Cerberus'09 Team Description Paper

H. Levent Akın, Çetin Meriçli, Tekin Meriçli, Ergin Özkucur, Can Kavaklıoğlu, and Barış Gökçe

Boğaziçi University
Department of Computer Engineering
34342 Bebek, Ístanbul, TURKEY
{akin, cetin.mericli, tekin.mericli, nezih.ozkucur, can.kavaklioglu, sozbilir}@boun.edu.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. The team competed in Robocup 2001-2008 except the year 2004. 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 the Robocup 2008 organization [1], a new league, named the Standard Platform League (SPL) [2], was introduced. In this league, the Nao humanoid robots manufactured by Aldebaran Robotics [3] are used as the standard robot platform and no hardware modifications are allowed as was the case for the 4-Legged League with Aibo robots. Cerberus competed in both the 4-legged and the 2-legged categories of the SPL in 2008 and made it to the quarterfinals, losing to the eventual champion in the 4-legged category.

The organization of the rest of the report is as follows. The software architecture is described in Section 2. In Section 3, the algorithms behind the vision module are explained. Self localization method is given in Section 4. The locomotion module and gait optimization methods used are explained in Chapter 5 and the planning module is described in Section 6.

2 Software Architecture

Software architecture of Cerberus has been completely rewritten in 2008. The existing modular architecture was transformed into a more general library architecture, where the code repository is separated into levels in terms of generality. Similar to the well known Model-View-Control architecture, the main goal of this new approach was to organize our code base into logical sections all of which are easy to access, manipulate, and debug. The rewrite process was originally

targeting the Aibo platform but the well designed architecture has made our initial development on Nao painless and quick.

Software architecture of Cerberus consists of mainly three parts:

- BOUNLib
- Cerberus Player
- Cerberus Station

2.1 BOUNLib

Past experience has demonstrated the previous modular approach to be sub-optimal in some cases. Especially considering issues such as reuse of source code for multiple architectures and also multiple purposes, making specific modifications to the special purpose modules becomes very time consuming and error prone.

We have started collecting general parts of our code base in a library structure called *BOUNLib*. Using this library will enable us to easily code for different platforms or different robots by reusing most of our code base.

2.2 Cerberus Station

BOUNLib library includes a versatile input output interface, called *BOUNio*, providing essential connectivity services to the higher level processes such as reliable UDP protocol, file logging, and TCP connections. Connections are made seamlessly to the sender, thus there is no need to write specific code for any application or test case.

Using *BOUNio* library enabled us to implement a very general version of our previous *Cerberus Station* using Trolltech's Qt Development Framework [4]. Using the well structured architecture of our runtime code and Cerberus Station, it is very easy to test new features to be added to the robot, which is a very vital resource for any research experiment.

Cerberus Station is designed to have the same features of old *Cerberus Station* and more, mainly aimed at visualizing the new library based code repository, some of which are listed below:

1. Record and replay facilities providing an easy to use test bed for our test case implementations without deploying the code on the robot for each run.
2. A set of monitors which enable visualizing several phases of image processing, localization, and locomotion information.
3. Recording live images, classified images, intermediate output of several vision phases, objects perceived, and estimated pose on the field in real time.
4. Log to file and replay at different speeds or frame by frame.
5. Locomotion test unit in which all parameters of the motion engine and special actions can be specified and tested remotely.

3 Vision

3.1 Image Processing and Perception

The purpose of the perception module is to process the raw image and extract available object features from the image (bearing and range if available). The input to the module is the image in YUV422 format and the output is the range and bearing features of objects seen on the field.

Color Quantization In the raw image format, each pixel is represented with a three-byte value and can be one of the 255^3 values. Since it is impossible to efficiently operate on such an input space, the colors are quantized into a smaller set of pseudocolors of white, green, yellow, blue, robot-blue, orange,red, and "ignore".

To efficiently get outputs from the trained GRNN network, a look up table is constructed for all possible inputs. To look up the color group of a pixel, Y, U, and V values are used to calculate the unique index and the value at that index gives the color group ID. In Figure 1, output of a trained GRNN is tested on a sample image.

Fig. 1. A classified image constructed with a trained GRNN.

3.2 Scanline Based Perception Framework

The most significant improvement over blob based systems is the increased reasoning possibilities. Using a single scanline, it is possible to detect a line segment by simply tracking the change in color of the pixels on the scanline. Furthermore, it is possible to combine information gained by multiple scanlines providing further spatial information about the structures available in the current image. In our previous blob based perception system such extensive and precise reasoning was not possible.

Scanline perception framework is also more scalable. The complexity of the system can be changed automatically in run time, by adjusting the number of scanlines per image. Since only the pixels on the scanlines are segmented, segmentation overhead of the blob based systems can also be avoided.

Inspecting a region of the image with a set of scanlines can provide a fast and accurate method of reasoning about the confidence of the perception. Such confidence values are very crucial for the performances of higher level modules. Further probabilistic methods will be employed to increase utility of these confidence values in the future.

In the following parts of this section, an overview of the implemented scanline based perception methods are presented.

Vision Object The *Vision* object instantiates other perception methods after generating some information which is required by all of the specialized perception classes.

The image received from robot's camera is first scanned with several scanlines. The number of these scanlines can be altered according to the viewing angle of the camera, however currently this feature is left as future work.

Once pixels corresponding to the scanlines are segmented, this segmented and ordered set of pixels are traversed once for important points, defined by each specialized perception method. Selected subsets of the segmented pixels are then sent to specialized perception classes for further processing and object detection.

Goal Perception The goal of the scanline based goal perception is to detect the goal bars individually so that multiple landmarks can be perceived from a single goal. A four stage procedure is designed to achieve this purpose, which can be used to perceive left and right goal bars individually. Due to the generic design of the perception stages and the underlying scanline framework, implementing top and bottom bars is only a task of mirroring left and right bar perceptors. Similarly beacon perception will be only a variation of the bar perception.

Tests of the perceptor are done using the new version of the *Cerberus Station*, a screen shot of which is shown Figure 3.2. Using this visual tool greatly enhances the testing procedure. Employing the *BOUNio* library makes it possible to observe run time performance as well as to inspect recorded logs within the goal perception tester tool.

Goal Target Perception Using the available scanline perception framework it is quite easy to implement simple perceptions quickly having all the robust properties of the scanline vision technique. Goal target perception presents a good example for such a case.

In order to score a goal, a goal target perception is required, so that the robot can shoot at the correct spot. Localization can generally be relied upon to figure out the direction of the goal, however it can not be trusted for precise shooting due to high level of noise.

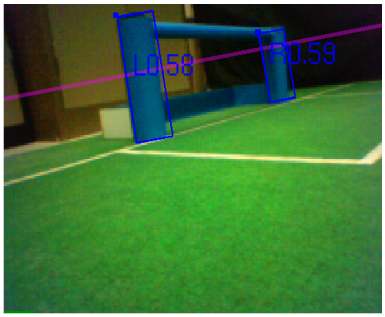

Fig. 2. Detected goal posts

Line Perception Line perception is implemented in a similar fashion with the goal target perception. The main vision object supplies the *important points* for line perception, which are, in this case, defined as the mid point of a green-white transition and a consecutive white-green transition on a single scanline.

"On Goal Line" Perception The most specialized perception is "on goal line" perception which provides a signal to the goal keeper to help the robot with its localization. Once the goal keeper roughly gets around the goal line, it can be quite hard to get localized precisely. Since the opponent goal is quite far away and generally obstructed, it can rarely be used in localization, which degrades the performance of the localization.

Distance Calculations To calculate the distance of a visually observed object, either a stereo vision system or using consecutive frames of single camera as stereo vision is needed [5]. The Nao robot has a single camera but the sizes of the objects in the environment are fixed and known. The distances of objects can be inferred by the size feature of the objects. In visual observations, the size of an object can be measured by the width or height of the object in pixels. As the object gets closer, the size of the object in the image increases and this increase has a nonlinear relationship with the actual distance of the object. In our approach, we represent this relationship with the function $y = a \times (x^b) + c$ where y is the distance, x is the width or height of the object in pixels, and a, b and c are parameters of the function.

4 Self Localization

Cerberus employs a vision based Monte Carlo Localization with a set of practical extensions (X-MCL) [6]. The first extension to overcome these problems and compensate for 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. This engine was used in both Aibo and Nao games.

4.1 World Modeling and Short Term Observation Memory

The perception module provides instantaneous information. While reactive behaviors like tracking the ball by head requires only instant information, other higher level behaviors and localization module needs more.

The planning and localization modules require perceptual information with less noise and more complete. The world modeling module should reduce sensor noise and complete the missing state information by predicting the state. This is a state prediction problem and we use the most common approach in the literature, which is the Kalman Filter [7].

In our problem, the observations are distance and the bearing of the objects with respect to robot origin, and the state we want to know is actual distance and bearing information. In addition to the dynamic objects like the ball, the state vector also includes distance change and bearing change information to ease prediction.

For any object, the observation is $z = \{d, \theta\}$ where d and θ are distance and bearing, respectively, to the robot origin. For the stationary objects, state is $m = \{d, \theta\}$ and the state evolution model is $m^1_{k+1} = I * m_k$ and $z_k = I * m_k$ where k is time and I is the unit matrix.

For the dynamic objects, the observation is the same but state is represented as $m = \{d, \theta, d_d, d_\theta\}$ where d_d is the change in distance in one time step and d_θ is the change in bearing likewise. The state evolution model is:

$$\begin{pmatrix} d_{k+1} \\ \theta_{k+1} \\ d_{d,k+1} \\ d_{\theta,k+1} \end{pmatrix} = \begin{pmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,1 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{pmatrix} \begin{pmatrix} d_k \\ \theta_k \\ d_{d,k} \\ d_{\theta,k} \end{pmatrix}$$

and observation model is:

$$\begin{pmatrix} d_{k+1} \\ \theta_{k+1} \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \end{pmatrix} \begin{pmatrix} d_k \\ \theta_k \\ d_{d,k} \\ d_{\theta,k} \end{pmatrix}$$

As can be observed from the model specifications, we omit the correlation between objects and separately execute filter equations for each object. If an object is not observed for more than a pre-specified time step, the belief state is reset and the object is reported as unknown. For our case, this time step is 270 frames for stationary objects and 90 frames for dynamic objects.

In the update steps, the odometry readings are used. The odometry reading is $u = \{d_x, d_y, d_\theta\}$ where d_x and d_y are displacements in egocentric coordinate frame and d_θ is the change in orientation. When an odometry reading is received, all the state vectors of known objects are geometrically re-calculated and the uncertainty is increased.

The most clear effect of using a Kalman Filter is that the disadvantage of limited field of view is reduced. In the Figure 3, a robot pans its head and is aware of three distinct landmarks at the same time.

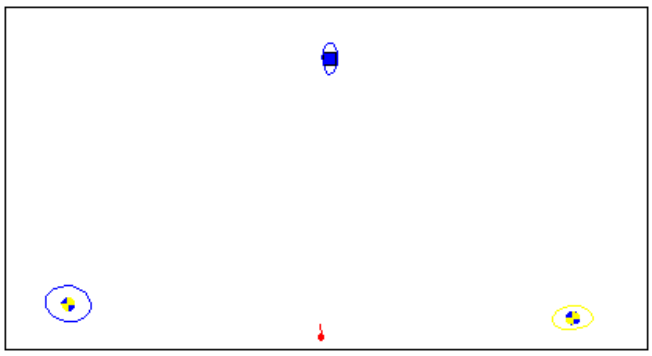

Fig. 3. A robot pans its head and is aware of three distinct landmarks at the same time.

4.2 Localization

The localization problem is a self pose estimation problem like in section 4.1, and the widely used approach is the Monte Carlo Localization (MCL) algorithm [8]. In this problem, the state to be estimated is $\mu = \{x, y, \theta\}$, where x and y are global coordinates and θ is orientation of the robot. In MCL algorithm, belief state is represented by a particle set and each element represents a possible pose of the robot. In Figure 4, a sample belief state is given.

5 Motion

Our motion engine has three different representation infrastructures, which allow different levels of abstractions. The first infrastructure contains a data structure, named “Body”, which stores the physical properties of the robot as well as the

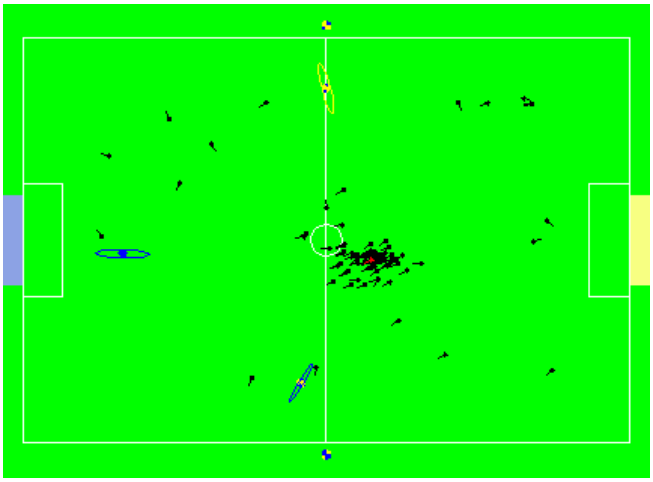

Fig. 4. Belief state of the robot in MCL algorithm. The best pose estimate is marked in red color.

functions used for performing related calculations. The "Body" is composed of several "Link"s, which are data structures for storing the joint positions, angles, and related kinematic description parameters. Being platform independent and generic, this infrastructure makes it possible to define new robot platforms via some configuration files and allows controlling the joints of the robot platform easily.

The second infrastructure is a hierarchical one. A root engine is defined at the very top level and the common properties and functions of each different motion engine are included in it. Different motion engines are inherited from the root engine, and platform and motion algorithm specific parts are defined. For Nao, four main features are implemented. The first one is to make the robot perform a static walking in which the robot tries to keep the ground projection of its Center of Mass (CoM) inside the support polygon. In addition to the static walking feature, a dynamic walking feature is also implemented. A signal generation-based algorithm, which is very similar to Central Pattern Generator [9] is used. The motion of the head is separated from the motion of the rest of the body and implemented as the third feature. The last feature is the motion player, which reads the sequential joint angles from pose definition files and plays them to realize some special actions, such as kicking the ball and standing up from a fallen position.

The third infrastructure is a container for some common motion-related functions. In addition to the matrix operations which are necessary for kinematics calculations, implementations to read configuration files are also included as common functions.

5.1 Bipedal Locomotion

Walking algorithms can be classified into two main groups; static walking and dynamic walking. The main principle of static walking is to preserve stability all the time during the motion, while dynamic walking is based on maintaining the balance by using dynamic properties of the motion. There are mainly three different methodologies;

- **Zero-Moment Point** (ZMP) Criterion [10] is very similar to CoM based static walking. The only difference is that holding ZMP inside the support polygon is enough to maintain the balance. Although it is a very common method, it is computationally very expensive.
- **Passive-Dynamic Walking** (PDW) [11] where CoM is carried along the motion direction, and the body moves along the motion direction because of the gravity. The important point is the timing of the foot contact of the swinging leg with the ground.
- Central Pattern Generators (CPGs) [9] method is based on the synchronous movement patterns of the joints. For this purpose, a signal is assigned for each joint, and the system is trained for synchronous patterns and balanced locomotion as a whole.

Model-Driven CPG-Based Biped Walking A walking method based on that of the champion of the Humanoid League in the RoboCup07, NimbRo [12], is implemented. They defined three important features for each leg; leg extension, leg angle, and foot angle. Leg extension is the distance between hip joint and ankle joint. It determines the height of the robot while moving. Leg angle is the angle between the pelvis plate and the line from hip to ankle. It has three components; roll, pitch, and yaw. The third feature, foot angle, is defined as the angle between foot plate and pelvis plate. It has only two components; roll and pitch. Using these features helps us to have more abstract calculations for the motion.

In order to find the leg angle and foot angle features, motion at each step is divided into five sub-motions; *shifting*, *shortening*, *loading*, *swinging*, and *balance*.

In shifting sub-motion, lateral shifting of the CoM is handled. For this purpose, a sinusoidal signal is simulated. The second important sub-motion is shortening signal and it is not always applied. During shortening phase, both a joint angle for the foot and a part of the leg extension value are calculated as a cosine function of the shortening phase value. The third sub-motion of the step is loading which is also not always applied. In this phase, only a part of the leg extension is calculated as that of shortening phase. Swinging is the most important part of the motion. In this part, the leg is unloaded, shortened and moved along the way of motion which reduces the stability of the system considerably. This movement has effects on each component of the leg and the foot angle features of the motion. As the last component of the step, balance, correction values for the deviations of the other operations are added to the system from the foot angle feature and the rolling component of the leg angle feature. At the end, the

corresponding parts of the sub-motions are added, and the values for the motion features are calculated.

Aside from the implementation inspired from the work of the NimbRo team, we have also developed a CPG-based custom algorithm for bipedal walking. In our design, the main walking motion starts from the hip, specifically the roll joint, which makes the body to swing from one side to the other. In order to keep the feet parallel to the ground while swinging, the ankle roll joint angles should be set to the negative of the value of the corresponding hip roll joint angle. The periodic movement of the hip is obtained by using a sinusoidal signal to be supplied as the hip roll joint angle. In order to realize this movement, the hip roll and ankle roll angles are set according to the following equations.

$$\theta_{hip_{roll}=A_{hip_{roll}}sin(period)}$$
$$\theta_{ankle_{roll}=-A_{ankle_{roll}}sin(period)}$$

This motion is the basis of the entire walking since it passes the projection of the center of mass from one foot to the other periodically, letting the idle foot to move according to the requested motion command.

In order to make the robot perform a stepping motion, the pitch joints on the leg chain should be moved. These joints again take sinusoidal angle values which are consistent with the hip roll angle. The following equations illustrate how the values of these angles are computed.

$$\theta_{hip_{pitch}} = A_{pitch}sin(period) + \theta_{hip^{rest}_{pitch}}$$
$$\theta_{knee_{pitch}} = -2A_{pitch}sin(period) + \theta_{knee^{rest}_{pitch}}$$
$$\theta_{ankle_{pitch}} = A_{pitch}sin(period) + \theta_{ankle^{rest}_{pitch}}$$

The A_{pitch} value determines how big the step is going to be. Obtaining backwards walk does not require much work but just reversing the iteration of the *period* value, which is defined as $0 < period < 2\pi$.

Similarly, making the robot move laterally is possible by setting the *roll* angles instead of the *pitch* angles together with the knee pitch, while turning around is possible by setting the $hipYawPitch$ joint angles properly. The amplitudes $A_{pitch}, A_{roll}, A_{yaw}$ are multiplied with the corresponding motion component, namely $forward, left, turn$ which are normalized in the interval $[-1, 1]$, to manipulate the velocity of the motion. In order to make the robot move omnidirectionally, the sinusoidal signals that are computed individually for each motion component are summed up and the final joint angle values obtained in that way. For instance, it is possible to make the robot walk diagonally in the north-west direction by simply assigning positive values to both the $forward$ and the $left$ components.

6 Planner

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 Figure 5

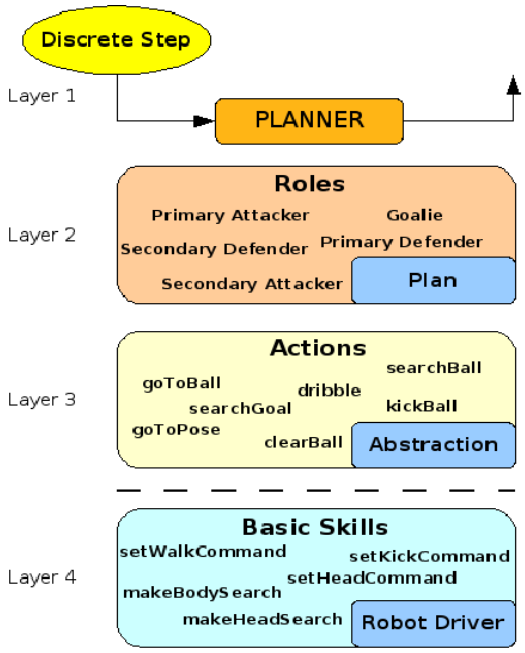

Fig. 5. Multi-layer Planner.

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*[13], *Chain of Responsibility Design Pattern* [14] and *Aspect Oriented Programming* [15].

For coordination among the teammates and task allocation, we employ a market driven task allocation scheme [16, 17]. 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 pick 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 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.

Acknowledgements

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

References

1. Robocup. www.robocup.org/.
2. SPL. Robocup standard platform league www.tzi.de/spl/.
3. Aldebaran-Nao. http://www.aldebaran-robotics.com/eng/nao.php.
4. Trolltech's Qt Development Framework. http://trolltech.com/products.
5. Thomas Lemaire, Cyrille Berger, Il-Kyun Jung, and Simon Lacroix. Vision-based slam: Stereo and monocular approaches. *Int. J. Comput. Vision*, 74(3):343–364, 2007.
6. K. Kaplan, 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, LNCS*, 4020:420–427, 2006.
7. Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.
8. Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2001.
9. C. Pinto and M. Golubitsky. Central pattern generators for bipedal locomotion. *J Math Biol*, 2006.
10. M. Vukobratovic and D. Juricic. Contribution to the synthesis of biped gait. *IEEE Transactions on Bio-Medical Engineering*, 1969.
11. Tad McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9:62–82, April 1990.
12. Sven Behnke. Online trajectory generation for omnidirectional biped walking. 2006.
13. Wikipedia Anonymous. Factory method pattern, 2007.
14. Wikipedia Anonymous. Chain-of-responsibility pattern, 2007.
15. Wikipedia Anonymous. Aspect-oriented programming, 2007.
16. H. Köse, Ç. Meriçli, K. Kaplan, and H. L. Akın. All bids for one and one does for all: Market-driven multi-agent collaboration in robot soccer domain. *Computer and Information Sciences-ISCIS 2003, 18th International Symposium Antalya, Turkey, Proceedings LNCS 2869*, pages 529–536, 2003.
17. H. Köse, K. Kaplan, Ç. Meriçli, U. Tatlıdede, and H. L. Akın. Market-driven multi-agent collaboration in robot soccer domain. *Cutting Edge Robotics*, pages 407–416, 2005.