Cerberus’15 Team Report

H. Levent Akın
Okan Aşık
Binnur Görer
Bahar İrfan
Metehan Doyran
Berna Erden
Kadriye Yasemin Usta
Artificial Intelligence Laboratory
Department of Computer Engineering
Boğaziçi University
34342 Bebek, İstanbul, Turkey
{akin, okan.asik, binnur.gorer, bahar.irfan,
metehan.doyran, berna.erden6, kadriye.usta}@boun.edu.tr

December 2015
# Contents

1 Introduction ........................................... 1

2 Software Architecture ............................... 3
   2.1 Building and Running Cerberus 2015 Code ........ 5

3 Vision ................................................. 7
   3.1 Color Segmentation .............................. 7
   3.2 Scanline Based Perception Framework .......... 8
      3.2.1 Robot Perception .......................... 9
      3.2.2 White Goal Perception .................... 10
   3.3 World Modeling and Short Term Observation Memory .... 10

4 Self Localization .................................... 13
   4.1 Field Model based Non-unique Observation Identification ... 14

5 Motion ................................................. 15

6 Planner ............................................... 16
   6.1 Finite State Controller based Planner ............ 16
   6.2 Multi-Robot Role Allocation ..................... 17

7 Coach Robot .......................................... 19

8 Technical Challenges ............................... 21
   8.1 Corner Kicks Challenge ......................... 21
      8.1.1 Assignment of Roles ....................... 21
      8.1.2 Passer Behavior .......................... 22
      8.1.3 Striker Behavior ......................... 22
      8.1.4 Result at the RoboCup .................... 23
   8.2 Many Carpets Challenge ......................... 23
      8.2.1 Motion and Ball Search Behavior .......... 23
      8.2.2 Result at the RoboCup .................... 23
8.3 Realistic Ball Challenge ........................................ 23
     8.3.1 Ball Search Behavior .................................... 24
     8.3.2 Result at the RoboCup .................................. 24

9 Conclusion .......................................................... 25
Chapter 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 Boğaziçi University (BU), Istanbul, Turkey and Technical University Sofia, Plovdiv branch (TUSP), Plovdiv, Bulgaria. The team competed in Robocup 2001-2015 except the year 2004. Since 2005, 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.

Through the years, the members of the team have done many PhD, MS, and BS thesis studies related to Standard Platform League (SPL) and published more than 40 papers in journals and international conferences, including the RoboCup Symposia\(^1\).

Until 2015, we have been developing all required software modules as a part of general robotics framework in our lab. Last year, the team shifted its focus from developing and maintaining all the software modules to provide hands-on experience for undergraduate students. Therefore, starting from the last year, we started using the B-Human code architecture and adapted our cognition code base for the B-Human infrastructure [1]. Our team played in the quarterfinals in last year’s competition.

In this document, we aim to summarize the work that has been done in our labs. The organization of the rest of this report is as follows. Firstly, we elaborate

\(^1\)The full list of Cerberus publications are available here: http://robot.cmpe.boun.edu.tr/ cerberus/wiki/doku.php/publications
on the software architecture in Section 2; presenting the overall framework and explaining the building and running procedures for those who are interested in studying our code. Later, we one-by-one present the modules that we developed. Technical details of the vision module are provided in Section 3. Localization module and the self localization method we employ in this module is described in Section 4. The details of the motion module is given in Section 5. Various approaches we use for the planning module are described in Section 6. Finally, we also present our coach robot in Section 7.
Chapter 2

Software Architecture

Before we started using B-Human infrastructure, the overall architecture as shown in Figure 2.1, was in fact an instance of a classical *sense-plan-act* paradigm. The old architecture started with reading sensory information including images from the camera, readings of the inertial sensors, and current positions of the body joints. Detailed information about our old architecture can be found in [2].

![Figure 2.1: The overall control architecture of the Cerberus robot soccer team.](image)

In order to use the API of NaoQi, we have adapted the shared memory communication infrastructure and walk engine of the B-Human’s code [3]. From our previous code base, we took the perception module, self localization, world modeling and planner units and integrated them with the B-Human infrastructure. In order to achieve this, we needed to follow the same formalism as B-Human used.
Therefore, we provided our modules with the necessary input/output mechanisms through wrapper classes we developed. These classes wrap our modules to enable the two architectures work together. The mechanisms wrapper classes provide are called *representations*, as B-Human infrastructure defines them. Under each module we develop, the following representations are produced by the corresponding wrappers:

1. CerberusPerception:
   - NaoVisionModule
     - BallPercept
     - GoalPercept
     - LinePercept
     - RobotPercept
     - FieldBoundary
     - ObstaclePercept

2. CerberusModelling:
   - ModellingWrapper
     - RobotPose
     - ObstacleModel
     - CombinedWorldModel
     - FieldCoverage
     - ObstacleClusters
     - SideConfidence
   - CerberusWorldModelProvider
     - CerberusWorldModel
   - CerberusBallModelProvider
     - BallModel
3. CerberusPlanner:

- PlannerWrapper
  - ArmMotionRequest
  - MotionRequest
  - HeadMotionRequest
  - ActivationGraph
  - BehaviorLEDRequest

As mentioned earlier, these representations are used for the purpose of integrating our modules with B-Human infrastructure[1].

2.1 Building and Running Cerberus 2015 Code

Our code for the last year’s competition is publicly available in Cerberus online repository[4]. As we moved on to use the infrastructure provided earlier by B-Human team, we expect it to be compiled and run successfully in Windows and OS X environments, just as described in [1]. On the other hand, Cerberus members developed all the software on Ubuntu 14.04 64-bit and therefore, we do not recommend trying our code on any other operating systems.

One can download the code from Cerberus online repository and save it in a folder where she likes. To carry on with the building procedure, firstly, the dependencies of the B-Human infrastructure must be downloaded and installed. These dependencies are listed below:

- qt4devtools
- libglewdev
- libxml2dev
- clang
- graphviz
Upon installing these, the developer can go ahead with the compilation procedure by entering into the *Make/Linux* folder and running the *make* command in this directory. The software developed by Cerberus is fully integrated with the infrastructure and should be compiled successfully.
Chapter 3

Vision

The purpose of the perception module is to process a raw image and extract available object features from it. Raw image is in YUV444 format with the resolution of 640 x 480 for the top camera and 320 x 240 for the bottom camera.

We first use a color table based color segmentation and extract colored segments for possible object features. These segments are combined to create region of interests. Finally, these regions are analyzed for the detection of field objects.

3.1 Color Segmentation

We previously utilized a Generalized Regression Neural Network (GRNN) [5] for mapping the real color space to the pseudo-color space composed of a smaller set of pseudo-colors, namely, white, green, blue, orange, red, and “ignore”. However, due to high training time, we switched to decision tree training using labeled images in RoboCup 2014. However, the success of decision tree is generally determined by the chosen labeled images which cannot be guaranteed on competition site. Therefore, in RoboCup 2015, we use the color table approach developed by B-Human team [1]. This approach creates a pre-calculated color table by color thresholding in the HSV color space. To reduce the size of the color table, 3 bits are shifted right in the Luminance channel of the color.
3.2 Scanline Based Perception Framework

Considering that the cameras of the Nao robots provide higher resolution images and the processors are slower, it becomes infeasible to process each pixel to find the objects of interests in the image due to computational efficiency and real-time constraints. Therefore, scanlines are used to process the image in a sparse manner, hence speeding up the entire process.

Algorithm 1 The image processing pipeline.

- calculate horizon
- create vertical scanlines
- find field border points
- extract possible obstacle spots
- create convex hull of field border points
- adjust scanlines to field border
- extract possible goal post region of interest
- apply Sobel-Feldman filter in vertical direction on Luminance channel
- adjust scanline for body contour
- extract white and orange segments
- build regions from extracted segments
- perceive ball using regions
- perceive lines using regions

The process starts with the calculation of the horizon based on the pose of the robot’s camera with respect to the contact point of the robot with the ground, that is the base foot of the robot. After the horizon is calculated, scan lines that are five pixels apart from each other and perpendicular to the horizon line are constructed, such that they originate on the horizon line and terminate at the bottom of the image. The first step after that is to scan through these scan lines to find where the green field starts, which is done by checking for a certain number of consecutive green pixels along the line. That results in a green region where all non-green parts that are close to the edges of the field ignored, such as the goal posts and balls that are on the border lines. To be able to process these objects, a convex-hull is formed for the starting points of the green segments. That way, we define the green field borders where all objects of interests fall inside; hence, we can basically ignore, say all orange regions, that are outside the field borders. That provides a natural
way of pruning false percepts without having to process them beforehand. After the field borders are constructed, the shorter scanlines are extended back to these borders, so that it is possible to use them to detect the goal posts and balls that are close to the borders.

After all these constructions and corrections, each scan line is traced to find colored segments on them. After only one pass over these scan lines, we end up with groups of segments with the colors we are interested in, namely, orange, white, blue. The next step is to build regions from these segments, based on the information on whether two consecutive segments “touch” each other, that is they are on two consecutive scan lines and either of them has a start or end point within the borders of the other one. Two consecutive touching segments are merged into a single region. For white segments though, there are some additional conditions, such as change in direction and change in length ratio. These additional constraints guarantee that all field lines are not merged into a single, very big region, but instead into smaller and more distinctive regions. After the construction of these regions, they are passed to the so called region analyzer module to be further filtered and processed for the detection of the ball, the field lines and intersections of them.

3.2.1 Robot Perception

As the field border is constructed, we identify possible obstacle spots according to the deviation of the border point on the image. If the border point deviates from two of its neighbors in both directions, this border point is marked as an obstacle spot. After collection all possible obstacle spots, we validate the spots by clustering near obstacle spots. We identify these obstacles as either robot, goal post or unknown object. We identify our own robots by checking the shirt color by vertical scan over the obstacle box. If we cannot identify an obstacle as a robot, we apply edge detection to identify the obstacle as goal post as detailed in Section 3.2.2. If we still cannot identify the obstacle, it will be used as an unknown obstacle. An example image where red and blue robots are identified can be seen Figure 3.1.
3.2.2 White Goal Perception

We identify the possible goal post spots by detecting anomalies in the green border set. After extracting obstacle spots and not being able to identify them as robots, we apply following sanity checks to identify them as possible white goal posts:

- The distance of the possible goal post from the border of the field.
- 90% percent white color rate.

After having found the region of interest for white goal posts, we apply Sobel-Feldman filter[6] to the region on the Luminance channel of the image. We extract the edges by thresholding the resulting image. Edges are clustered using k-means clustering. If there is any cluster having 60% of all edge points, we identify this obstacle as a white goal post as seen in Figure 3.2.

3.3 World Modeling and Short Term Observation Memory

The perception module of Cerberus provides instantaneous information. While the reactive behaviors like tracking the ball with the head requires only instantaneous information, other higher level behaviors need more than that.

The planning module requires perceptual information with less noise and in a more complete manner. 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, the Kalman Filter [7], for solving this problem.

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

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

For the dynamic objects, the observation is the same but the state is represented as $m = \{d, \theta, d_d, d_\theta\}$ where $d_d$ is the change in distance in one time step and $d_\theta$ 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 the 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 the objects and use filter equations for each object separately. 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 \textit{unknown}. For our case, this time step is 270 frames for stationary objects and 90 frames for dynamic objects.

In the update steps, 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 \( d_\theta \) is the change in orientation. When an odometry reading is received, all the state vectors of known objects are geometrically re-calculated and the associated uncertainty is increased.

The most obvious effect of using a Kalman Filter is that the disadvantage of having a limited field of view is reduced. As the robot pans its head, it can be aware of distinct landmarks which are not in the same field of view at the same time.
Chapter 4

Self Localization

Cerberus employs vision based Monte Carlo Localization (MCL). In the MCL algorithm, the belief state is represented by a particle set and each element represents a possible pose of the robot. We use MCL with a set of practical extensions (X-MCL) which is detailed in [8] and inspired from FastSLAM [9] algorithm and Multi-Hypothesis tracking [10]. In FastSLAM, each particle has its own world model (i.e. map). In Multi-Hypothesis tracking, there are multiple Gaussians where each relies on a different data association sequence and their numbers are limited by pruning. Until last year, we used the output of the world modeling module as input to the localization module. Namely the filtered landmarks are used as observations for the localization module. However, this year we also implement a field model approach to identify unidentified observations by associating them with the most likely one.

Our multi-hypothesis tracking approach defines landmark groups which a non-unique observation might be observed from. For example a T type field line intersection might be observed from 6 different landmarks. Similarly a goal bar observation might come from left or right goal bar. We define a label for each non-unique observation which indicates the identity in its group. We augment the discrete label variable to the particles. Each particle now represents robot pose and a label variable associating non-unique observations to landmarks in the map. This model is an instance of Switching Observation Model [11]. The particles with wrong labels eventually die in the resampling steps.
4.1 Field Model based Non-unique Observation Identification

When the robot perceives a non-unique observation such as goal bar, we identify this percept according to the current estimated position of the robot. We maintain a field model based on the current estimated position of the robot, each observation is mapped to the field model and most likely observation is calculated. We order the observations based on orientation difference and distance difference for the current field model. We first prune the likely observations based on a threshold for orientation difference. Then, we choose the one which has the least distance according to the field model as shown in Algorithm 2.

Algorithm 2 The field model based observation association.

determineSide()
sanityChecks()
matchingObservation ← none
minDist ← maxNumber
for all fieldObjects do
    if oriDiff(currentObservation, fieldObject) < oriThreshold then
        if dist(currentObservation, fieldObject) < minDist then
            matchingObservation ← fieldObject
            minDist ← dist(currentObservation, fieldObject)
        end if
    end if
end for
calculateParticleWeights()
resample()
injectParticles()
estimatePose()
Chapter 5

Motion

Until RoboCup 2015, we used our inhouse developed motion infrastructure and our own static motion engine. We used B-Human walking engines. Starting from RoboCup 2015, we started to use B-Human motion infrastructure [1]. We use the walking engine and get up motions as provided by B-Human code, however, we changed kick motions’ parameters for longer kicks.
Chapter 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 decides the new actions. The main aim of the planner is to sufficiently model the environment and update its state. Additionally, the planner should provide control inputs according to this model. Previously, we developed a Dec-POMDP based planner. Currently, we use a finite state machine based planner architecture as explained in Section 6.1. We also extend the behaviors by integrating market-based multi-robot role allocation mechanism as explained in Section 6.2.

6.1 Finite State Controller based Planner

The Finite State Controller (FSC) based planner makes use of the formal model of the problem. At every planner step, the robot is in a particular state and we want our robot to take the best action in that state. FSC is based on the conventional Hierarchical Finite State Machine model, however, we changed some aspects to use it in high-level robot planning. There are states which correspond to the environment states. Transitions take place according to the current environment observations. There are also actions which will be taken when the robot is at a particular state. The robot can execute many actions in a particular state and these actions may override each other according to their priority. The most
powerful part of this planner architecture is that once we code particular transitions or actions, they can be reused in different behaviors. We have developed a GUI tool called FSC Designer for this purpose [12]. FSC Designer enables easy development of finite state controller based behaviors by using already developed Transition and Action constructs as seen in Figure 6.1.

Figure 6.1: Snapshot of FSC Designer

6.2 Multi-Robot Role Allocation

We define a team formation for four field players. A place in the formation corresponds to a role, such as striker, attacker, defender and wings. Each role have a corresponding rectangular field defined on the field. The robot that is assigned to a role can only move in that rectangular field except when it is the closest robot to the ball. We used market-driven algorithm[13] to decide which robot gets to the ball. The robots calculate their costs for getting the ball role considering their distances to the ball, their orientation, and whether there are obstacles on the way or not. Then, a robot is assigned to the “on ball” role when it has the lowest cost among others. The costs are calculated according to the distance of the robot to
the position of the ball on the field, considering obstacle avoidance and orientation changes. This calculation and assignment is done in a distributed fashion by using teammate messages shared between players. Then the rest of the robots are assigned to roles, that are statically determined before the game starts according to the robot numbers.

We also used **ball attraction parameter** to determine which robot should go where in their each rectangular field. The robots are attracted towards the ball depending on their roles. For example, a robot which has the role defender is less attracted to the balls that are closer to the opponent goal than its own goal, and a robot that is assigned to the attacker role is much more attracted to that ball. The exact position of the robots are determined according to the position of the ball. Since, it would result in oscillation in role assignment, the allocation mechanism is run at once every second.

As a future work we want to add dynamic role allocation to our system. This means doing all the role assignments with the market-driven algorithm.
Chapter 7

Coach Robot

We developed a coach robot to assist the team with strategic decisions. Since it is a newly defined concept, primarily we first construct a SimRobot simulator scene for being able to conduct experiments on it. The scene obeys the rules of 2015 [14] as shown in Figure 7.1.

![Screenshot from B-Human’s simRobot simulator](image)

Figure 7.1: Screenshot from B-Human’s *simRobot* simulator

In the behavior part, the coach robot just sits on the table and scans all the field with its head moves for creating a world model of all the field. With modified vision, the robot perceives robot positions, ball positions and other landmarks such as field lines. We need to change vision because the coach robot is on a table
which gives a height difference from other robots. After constructing the world model of all robots, it calculates the soccer metrics [15] and updates its statistics.

According to changes in the statistics, the robot will select a predefined tactic with the help of the decision (classification) tree. There are mainly three formations, i.e. offensive, defensive and neutral. These different formations determine how likely the team will be on other team’s half field. The selected formation is represented as a human-readable message and broadcast by the GameController since the robot can not communicate with the other robots directly as the Rule Book states [14]. Then, the calculated formation is carried out by the dynamic role allocation module of the planner.
Chapter 8

Technical Challenges

In this section, we describe our contributions to the technical challenges of SPL in the RoboCup 2015. The technical challenges were namely, Corner Kicks Challenge, Many Carpets Challenge, and Realistic Ball Challenge. We have participated in all three challenges. We scored an overall of 19.44 points in the challenges. For Corner Kick and Realistic Ball challenges, we received 5 points each. For the Any Carpet challenge, we received 9.44 points for touching the ball on two of the carpets.

8.1 Corner Kicks Challenge

The aim of this challenge was to test the passing skills between teammates, the level of team play and the detection of opponent teams robots. The rules for this challenge can be found in [16].

8.1.1 Assignment of Roles

In this challenge the first robot to touch the ball should be the one closest to the ball, hence the robots should assign roles for themselves. In our approach, the robot that is less than 90 cm to the ball is assigned as the first robot to touch the ball and the passer, $R_1$, and the other robot is assigned to the role of the striker, $R_2$. 
8.1.2 Passer Behavior

For the first time, \( R_1 \) kicks the ball towards the static position of \( R_2 \) robot, at an angle

\[
\alpha = \arctan\left(\frac{\Delta R_{12y}}{\Delta R_{12x}}\right) + \pi/2 - (\arctan\left(\frac{\Delta R_{1Oy}}{\Delta R_{1Ox}}\right) + 5)
\]

where \( \Delta R_{12x}, \Delta R_{12y} \) represent the initial position difference of \( R_1 \) and \( R_2 \) in \( x, y \) coordinates, and \( \Delta R_{1Ox}, \Delta R_{1Oy} \) represent the distance between the leftmost obstacle and \( R_1 \) in \( x, y \) coordinates. 5 degrees counterclockwise is added to the angle to the closest obstacle, such that \( R_1 \) would throw the ball to in front of \( R_2 \) and if there is any other obstacles. After kicking, \( R_1 \) waits until the ball is closer to it than 30 cm. If \( R_1 \) is kicking the ball for the second time, it should orient towards,

\[
\alpha' = - (\arctan\left(\frac{\Delta R_{1Oy}}{\Delta R_{1Ox}}\right) + 5)
\]

that is, in front of its teammate or any other left-most obstacle, such that \( R_2 \) would see the ball. If it cannot see any obstacle, it would rotate 5 degrees counterclockwise towards the field. Our vision algorithm could not separate a teammate from an opponent team’s robot, hence we chose to implement an algorithm in this way.

8.1.3 Striker Behavior

\( R_2 \) waits for 15 seconds to pass such that \( R_1 \) would touch the ball and pass it to itself. After 15 seconds, it searches the ball with head and body search and goes to the ball, if it can find it. For kicking the ball, \( R_2 \) orients towards the goal, while avoiding any obstacles. It’s angle towards the goal was

\[
\beta = \arctan\left(\frac{\Delta R_{2Gy}}{\Delta R_{2Gx}}\right) - (\arctan\left(\frac{\Delta R_{2Oy}}{\Delta R_{2Ox}}\right) + 5)
\]

where \( \Delta R_{2Gx}, \Delta R_{2Gy} \) denote the distance between \( R_2 \) and the goal post in \( x, y \) coordinates, and \( \Delta R_{2Ox}, \Delta R_{2Oy} \) represent the distance between the leftmost obstacle and \( R_2 \) in \( x, y \) coordinates.
8.1.4 Result at the RoboCup

We could not score any goals in this challenge, due to the calibration problems in our vision algorithm, therefore, we could not test our algorithm on the field before the challenge.

8.2 Many Carpets Challenge

The aim of this challenge was to test the motion skills of robots on different carpets. The rules for this challenge can be found in [16].

8.2.1 Motion and Ball Search Behavior

We have used B-Human’s walking and kick engines [1], however, we have modified the parameters in the engines and decreased our speed of walking and kicks.

The robot first turns around itself for 10 seconds, which approximately is a one full turn for the robot, such that it can localize itself when it detects the goal post. Then it searches for the ball, and goes towards the ball and kicks the ball towards the goal as in our game strategy.

8.2.2 Result at the RoboCup

Our robot was successful in walking and kicking without falling in all three carpets, however, our walking speed was very low for the one minute limit. Hence, we could not score any goals, but we achieved to touch the ball in two of the carpets.

8.3 Realistic Ball Challenge

The aim of this challenge was to detect if the robots can detect more realistic balls that have a different size or color than the standard ball used in SPL. The rules for this challenge can be found in [16].
8.3.1 Ball Search Behavior

The robot uses a head search for detecting the ball while moving towards the center line for six seconds. If during the walk, it had not detected any balls in its search, it turns around itself at the center line. If no ball is detected at the turn, then it moves 30 cm forward, while searching for a ball, and turns around itself after it moves 30 cm or at most 10 seconds. This continues until a ball is found. If a ball is detected, then the robot goes towards the ball, and orients itself towards the closest goal and kicks the ball. After it kicks the ball, the robot moves on to the next position, that is, it moves 30 cm forward, and searches for the ball during its walk.

For the vision algorithm, our team calibrated the color table used in the game for detecting our own ball, which was dark blue.

8.3.2 Result at the RoboCup

Due to the color table we were using and the lighting conditions in the challenge field, our robot saw a ball in its shadow, hence it tried to kick its shadow. Therefore, we could not score any points in this challenge.
Chapter 9

Conclusion

This document summarizes our work for the RoboCup SPL 2015 competition. In it, we present our approaches for solving certain problems in the RoboCup SPL domain. We elaborated on our software architecture and how it is integrated with B-Human infrastructure[1]. A technical inspection of our modules is also presented. Moreover, additional studies are mentioned, such as the work related to the technical challenges and the coach robot.

Overall, our goal of developing successful autonomous systems which are able to perform well in adversarial environments persists. In this regard, we aim to solve the multi-agent planning, and kidnapping problems which are the top challenges of SPL, in near future. We are currently doing research on the application of Dec-POMDP methods for robot soccer [17]. For solving the kidnapping problem, and in general the localization in symmetric fields, we are working on methods which effectively merge the world models of the teammates. Also, we are studying actively on the computer vision algorithms to adapt to the changing rules of SPL.
Acknowledgments

This work is supported by Boğaziçi University Research Fund through project 7361.
References


