Cerberus’14 RoboCup SPL Team Description

H. Levent Akın, Okan Aşık, Binnur Görer, Bahar İrfan, Ahmet Erdem
Boğaziçi University, Department of Computer Engineering
34342 Bebek, Istanbul, TURKEY
{akin, okan.asik, binnur.gorer, bahar.irfan, ahmet.erdem}@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 Boğaziçi University (BU), Istanbul, Turkey and Technical University Sofia, Plovdiv branch (TUSP), Plovdiv, Bulgaria. The team competed in Robocup 2001-2012 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.

From the very beginning, Cerberus has chosen to develop all components of the software to form the basis for a more general robotics research rather than to be used for soccer only. Through the years, the members of the team have done many PhD and MS Thesis studies related to SPL and published more than 40 papers in journals and international conferences, including the RoboCup Symposia.

The organization of the rest of the paper is as follows. The software architecture is described in Section 2. In Section 3 the details of the vision module are provided. Self localization method is described in Section 4. The locomotion module and gait optimization methods used are explained in Section 4.1. Finally, various approaches we use for the planning module are described in Section 5.

2 Software Architecture

Software architecture of Cerberus consists of mainly three parts:

- BOUNLib
- Cerberus Player
- Cerberus Station

Fig. 1 illustrates the general structure of our most generic software architecture, including the Cerberus code base.

1 The full list of Cerberus publications are available here: http://robot.cmpe.boun.edu.tr/cerberus/wiki/doku.php/publications
2.1 BOUNLib

Past experience has demonstrated the previously used modular approach to be sub-optimal in some cases. Reuse of source code for multiple architectures and also multiple purposes, and making specific modifications to the special purpose modules is very time consuming and error prone.

We collected the more general parts of our code base in a library structure called BOUNLib. Using this library enables us to develop software for different platforms or different robots easily by reusing most of our code base, as illustrated in Fig. 1.

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 [1]. It is very easy to test new features to be added to the robot using the well structured architecture of our runtime code and Cerberus Station. This is a very vital resource for any experiment involving robots.
The new Cerberus Station has the same features of its older version 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.

The screen shot in Fig. 2 demonstrates some of these features of the new Cerberus Station software.

![Cerberus Station Software](image)

Fig. 2. The new Cerberus Station software.

3 Vision

3.1 Image Processing and Perception

The purpose of the perception module is to process a raw image and extract available object features from it. The input to the module is the image in YUV422 format and the output is the range and bearing values of the perceived objects and landmarks.

**Color Quantization** We previously utilize a Generalized Regression Neural Network (GRNN) \[2\] for mapping the real color space to the pseudo-color space composed of a smaller set of pseudo-colors, namely, white, green, yellow, blue, robot-blue, orange,
red, and “ignore”. However, due to high running time, we train a decision tree using labeled images.

In order to obtain the outputs of the trained decision tree during games in a time-efficient manner, a look up table is constructed for all possible inputs. $Y$, $U$, and $V$ values are used to calculate the unique index and the value at that index gives the color group ID to determine the color group of a pixel. Fig. 3 shows a screen shot from the Labeler component of the Cerberus Station software, where it becomes possible to visually evaluate the resulting classification performance of the decision tree right after the labeling and training phases.

![Fig. 3. A classified image constructed with a trained decision tree.](image)

**Scanline Based Perception Framework** Considering that the cameras of the Nao robots provide higher resolution images and the processors are slower compared to those of the Aibo robots’, 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, scan lines are used to process the image in a sparse manner, hence speeding up the entire process.

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 5 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. Of course 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. In order to not lose information about those important objects, a convex-hull is formed for the starting points of the green segments. That way, we define the real 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 scan lines 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, and yellow. 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 the region analyzer module to be further filtered and processed for the detection of the ball, the field lines and intersections of them, and the goal posts. Fig. 4 shows the result of this processing, which takes less than 15 ms on the average. The thick red line represents the calculated horizon, the thin green line group represents the convex-hull, which corresponds to the green field border, thin red lines represent the white line segments to be further processed, the yellow line group represents the yellow goal post base, and the orange circle represents the detected ball. The egocentric positions of these objects are computed using the camera matrix and projecting them back on the field.

Fig. 4. Result of processing the image using scan lines.

3.2 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 \cite{1}, 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 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.

\section{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 [4]. 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 are modifying the algorithm to filter unidentified observations.

The new approach is inspired from FastSLAM [5] algorithm and Multi-Hypothesis tracking [6]. 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.

In our localization approach, we define 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 yellow goal bar observation might come from left or right yellow 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 [7]. The particles with wrong labels eventually die in the resampling steps.

![Snapshot of particles in the first scenario](image)

**Fig. 5.** Snapshot of particles in the first scenario

In Fig. 5 the robot stands still just outside the yellow penalty box, facing yellow goal, and scanning with its head continuously. The robot observes right and left yellow bars sequentially and does not know the identity. There are two possible label values in this scenario, and in the first few observations, both labels are maintained in the particle set. After more time steps, the true label dominate in the resampling states.

To overcome the unified goal bar color problem, with the assumption that initial position of the robot is known, the model described above works with minimal change. For the kidnapping situations, we plan to develop a binary hypothesis based approach. The methodology is as follows. After kidnapping (or falling), the robot makes an assumption
about the side of the first observed goal bar, and perform localization normally. After that it simultaneously tries to validate this hypothesis based on robot observations, and incoming messages from the teammates.

### 4.1 Motion

For bipedal locomotion, we use three different walking engines. While the first two engines are developed in the lab, we also use the walk engine developed by B-Human Team [8].

In the first bipedal walking algorithm, we defined two important features for each leg; leg extension, and leg angle. Leg extension is the distance between the hip joint and the 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. Using these features helps us have more abstract calculations for the motion.

Before finding motion features, a central clock ($\phi_{\text{trunk}}$) is generated for the trunk which is between $-\pi$ and $\pi$. Each leg is fed with a different clock ($\phi_{\text{leg}}$) with $ls \times \pi/2$ phase shift where $ls$ represents leg sign and it is $-1$ for the left leg while $+1$ for the right leg. The synchronization of the legs can be preserved in this way. In the calculations of motion features at a given time, the corresponding phase value is considered and the values for features are calculated by using these phase values.

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

In the shifting sub-motion, lateral shifting of the center of mass is handled. For this purpose, a sinusoidal signal is simulated. The second important sub-motion is the shortening signal and it is not always applied. During the 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. At the end, the corresponding parts of the sub-motions are added, and the values for the motion features are calculated.

Because balance is not guaranteed in the model and it is impossible to optimize the model with the maximum speed analytically, our biped walking is defined in terms of some parameters. After determining a feasible parameter set by hand, we applied an optimization algorithm, Evolutionary Strategies, to fine-tune the walking motion. Although both speed and balance is used in the fitness function, our walk engine is an open-loop engine during the game and it is vulnerable on the accumulation of balance disturbance. In order to compensate for these disturbances, we are working on how to obtain a feedback from the sensors and estimate the state of the robot. For this purpose, we log the readings of foot pressure and accelerometer sensors and simulate our walking style with the readings in Matlab. More details about this walking strategy can be found in [9].
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.

\[
\theta_{\text{hip roll}} = A_{\text{hip roll}} \sin(\text{period}) \\
\theta_{\text{ankle roll}} = -A_{\text{ankle roll}} \sin(\text{period})
\]

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.

\[
\theta_{\text{hip pitch}} = A_{\text{pitch}} \sin(\text{period}) + \theta_{\text{hip pitch}}^{\text{rest}} \\
\theta_{\text{knee pitch}} = -2A_{\text{pitch}} \sin(\text{period}) + \theta_{\text{knee pitch}}^{\text{rest}} \\
\theta_{\text{ankle pitch}} = A_{\text{pitch}} \sin(\text{period}) + \theta_{\text{ankle pitch}}^{\text{rest}}
\]

The \( A_{\text{pitch}} \) 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 \( \text{period} \) value, which is defined as \( 0 < \text{period} < 2\pi \).

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 \( \text{hipYawPitch} \) joint angles properly. The amplitudes \( A_{\text{pitch}}, A_{\text{roll}}, A_{\text{yaw}} \) are multiplied with the corresponding motion component, namely \( \text{forward, left, turn} \), 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 \( \text{forward} \) and the \( \text{left} \) components.

As the third walking engine, we use the omni-directional walking developed by B-Human Team [8, 10].

5 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 market based planner and a Dec-POMDP based planner. Currently, we use a finite state machine based planner as explained in Section 5.1.

5.1 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 [11]. FSC Designer enables easy development of finite state controller based behaviors by using already developed Transition and Action constructs as seen in Figure 6.

5.2 Goalie Planner

The goalie has its own finite state machine controller to execute its behavior. General structure of the goalie is composed of searching ball, tracking ball, blocking ball if it is
close enough and to localize itself during all of these behaviors. The localization problem to return to the center of the goal is solved partially by using corners, lines of the penalty area and odometry information. However, line and corner search and alignment processes take too much time when considering the speed of the game. The robot always has to check the ball of the position while localizing itself. As a result, we have to consider more visual sensory input to increase localization accuracy and decrease process time which has been planned as our direction in improving the performance of the goalie behavior.

6 Conclusion and Future Works

In conclusion, we aim to develop successful autonomous systems which are able to perform well in adversarial environments. We summarize our methods for solving certain problems in RoboCup Standard Platform League. In future, we aim to solve multi-agent planning, and kidnapping problems which are the top challenges of SPL. We perform research on the application of Dec-POMDP methods for robot soccer [12]. For solving the kidnapping problem, and in general the localization in symmetric field, we work on methods which effectively merge world models of teammates.

Acknowledgments

This work is supported by Boğaziçi University Research Fund through project 13A01P3.

References

