The University of Pennsylvania Robocup 2003 Legged Soccer Team

David Cohen, Yao Hua Ooi, Paul Vernaza, and Daniel D. Lee

General Robotics Automation, Sensing and Perception (GRASP) Laboratory
University of Pennsylvania, Philadelphia, PA 19104
WWW home page: http://www.cis.upenn.edu/robocup

Abstract. This paper presents the software design for a team of soccer playing robots developed at the Univ. of Pennsylvania for the 2003 Robocup competition. The software was developed from scratch a few months before the competition and featured several novel innovations. Lower level sensory and motor functions were first prototyped in Matlab before being ported onto the robots. High level behaviors and team coordination were implemented using an embedded Perl interpreter. These tools allowed the rapid development of the team that showed continuous improvement during the competition, ultimately resulting in a 2nd place finish.

1 Introduction

This paper presents the software structure for a team of soccer playing Aibo robots recently developed at the Univ. of Pennsylvania. Although we have competed in the Robocup competition in the past, this year featured new participants, including a new faculty advisor and several interested undergraduate students from the Engineering school. It was decided not to use any of the existing legacy code from previous teams, and to develop a new team from scratch for 2003. The effort started in earnest during the 2003 spring semester, leaving only a few months of development before the competitions. Thus, it was decided that the software would be prototyped as rapidly as possible, emphasizing simplicity in design over complex algorithms and complicated parameter tuning.

The resulting software architecture proved to be useful, since it was possible to rapidly improve the robot play during the week of Robocup competition in Padua. During that week, several new kicks, a new goalie, penalty kick behavior, and a new team coordination algorithm were implemented. This allowed the team to ultimately finish in 2nd place. The algorithms for the three challenges were also implemented only a day (and night) before that competition, resulting in a 4th place showing in the challenge results.

2 Software Architecture

The software architecture for the robots is shown in Figure 1. This archicture is implemented by compiling several OPEN-R modules using the Sony OPEN-R API [1]:

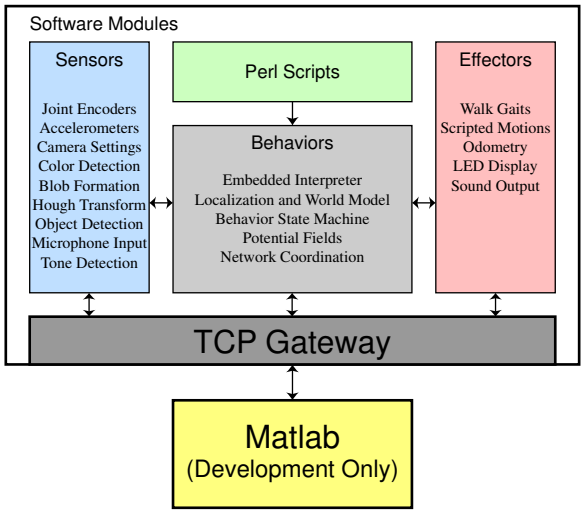

Fig. 1. Univ. of Pennsylvania software architecture for sensory and motor processing, as well as implementation of various behaviors for the Robocup competition.

ROBOTCOM.BIN Main module that includes the embedded Perl interpreter, vision and sensor interfaces, walk routine, and world model.

EFFECTOR.BIN Module that accepts arrays of joint angles and sequences the motors and other effectors such as LED's.

SNDCOMM.BIN Module that can play and record PCM sound samples.

In order to simplify development, all interprocess communications are performed by passing a shared memory matrix structure between the modules:

```
template <class T>
class ShmArray : public OShmPtrBase {
...
}
```

This structure allows for efficient communication as well as unifying the input and output interfaces of every module that is developed. Modules can also easily pass information to modules on other robots by sending these data structures through the supplied TCPGateway interface.

2.1 Matlab Interface

Rapid development of lower level sensory and motor functions is facilitated by using Matlab to first prototype the algorithms [2]. Matlab is a high-level numerical scripting language that allows complex algorithms to be coded in a small number of lines. Since Matlab uses matrices as its basic data structure, we developed several functions that allowed Matlab to send and receive matrices over the wireless network to the various OPEN-R modules running on the robots.

For example, the joint angles for a new kick are first computed using a Matlab routine. This results in a $16 \times N$ matrix in Matlab that describes the motion of each of the 3 joints in the 4 legs as well as the 4 head angles (pan, tilt, roll, mouth). This matrix is then sent to the Effector module on a running robot to be actuated in real-time. These angles are then modified in Matlab and replayed on the robot to quickly improve the performance of the motion. This interface allows for the development of a new kick in only a few hours. Several novel motions such as back flips, rolls, and upside-down walking routines were also developed using this software tool.

2.2 Perl Interpreter

High level behaviors are implemented using an event-driven state machine. To facilitate rapid development, we decided to implement these behaviors using Perl scripts rather than in the native C++ programming environment. Perl is a high-level "kitchen-sink" programming language that incorporates an optimized interpreter that compiles scripts into intermediate opcodes for efficient performance [3].

To enable this capability on the robots, we embedded a Perl interpreter to run on the robots [4]. Unfortunately, it is not possible to simply configure Perl for the Aibo robots using the normal build process. Due to the limited operating system environment on the robots, our Perl port is based upon the scarcely-documented microperl build in the 5.8.0 release of Perl. The interpreter is built as a static library that is linked into the ROBOTCOMM module. The resulting library references several functions such as "fork" in the standard UNIX API that are not available on the robots, so several dummy functions also needed to be implemented in order for the module to be linked properly.

To enable the Perl interpreter to interact with the sensory and motor routines implemented in the OPEN-R modules, the interpreter was extended so that it is able to call C functions exported by the modules. A small language called XS available in the Perl distribution was used to automate this task [1]. Through this interface, a Perl script is able to access data variables in the OPEN-R routines and activate functions that set status LED's, sequence motions and walk routines, and communicate with other robots to coordinate team behaviors.

The use of the Perl interpreter allows us to develop and modify behaviors for the robots using a standard high-level scripting language in the robots' runtime environment. Thus, we could test a new behavior by sending a new script to robots over the network and execute them on demand without having to reboot the robots. Additionally, due to Perl error handling, a bad script will rarely result in a system crash. This dramatically reduces the severity and duration of downtime during development time.

[1] see the perlxs UNIX man page for more information

3 Vision

Most of the algorithms used for processing visual information from the robots' CMOS cameras are similar to those used by other teams in the past [5]. Since fast vision is so crucial for the robots' behaviors, these algorithms were implemented from scratch to gain as much performance speed as possible. This speed was critical during the competition since we used several robots with older, slower processors during the preliminary rounds.

3.1 Object Recognition

The main processing pipeline involves segmenting the highest-resolution color images from the camera, forming connected regions, and recognizing various objects from the statistics of the colored regions. The color segmentation routine classifies individual pixels in the image based upon their YCbCr values. Based upon a number of training images, a Gaussian mixture model is used to segment the YCbCr color cube into the following colors:

- Orange (Ball)
- Pink (Marker)
- Green (Marker)
- Cyan (Marker and Goal)
- Yellow (Marker and Goal)
- Blue (Robot)
- Red (Robot)
- Green (Field)
- White (Lines and Border)

Once the pixels in the image are classified according to their colors, they are merged into connected components using techniques that have been previously described [5]. This is accomplished by first run-length encoding the images, and then merging these run-lengths into connected regions.

After the image has been segmented into these connected regions, the regions are classified into relevant objects by comparing various image statistics of the regions. These statistics include the bounding box of the region, the centroid location, and the major and minor axes lengths. In this manner, the location of the ball, markers, and goals are detected. There was not enough time before the competition to implement a robot detection algorithm, and our behaviors are based solely on the reported locations of our robots and the ball.

3.2 Line Recognition

One new aspect of our visual processing routines is the detection of field lines. This decreased the need for our robots to actively search for field markers, enabling them to chase the ball more effectively. The first step in line identification is to find white pixels in the medium resolution camera images that neighbor

pixels of field green color. Once these pixels are located, a Hough transform is used to search for relevant line directions.

In the Hough transform, each possible line pixel (x, y) in the image is transformed into a discrete set of points (θ_i, r_i) which satisfy:

$$x \cos \theta_i + y \sin \theta = r_i \tag{1}$$

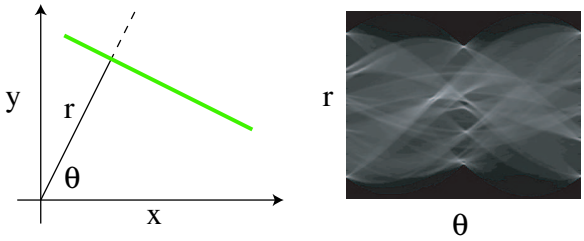

Fig. 2. Hough tranformation for field line detection in images.

The pairs (θ_i, r_i) are accumulated in a matrix structure where lines appear as large values as shown in Figure 2. To speed the search for relevant lines, our implementation only considers possible line directions that are either parallel or perpendicular in the field to the maximal value of the accumulator array. Once these lines are located, they are identified as either interior or exterior field lines based upon their position and then used to aid in localization.

4 Localization

The problem of knowing the location of the robots on the field is handled by a probabilistic model incorporating information from visual landmarks such as markers, goals, and lines, as well as odometry information from the effector module [6]. Recently, probabilistic models for pose estimation such as extended Kalman filters, grid-based Markov models, and Monte Carlo particle filters have been successfully deployed. Unfortunately, complex probabilistic models can be difficult to implement in real-time due to a lack of processing power on board the robots. We address this issue with a new pose estimation algorithm that incorporates a hybrid representation that reduces computational time, while still providing for a high level of accuracy. This new algorithm models the pose uncertainty as a distribution over a *discrete* set of heading angles and *continuous* translational coordinates. The distribution over poses (x, y, θ), where (x, y) are the two-dimensional translational coordinates of the robot on the field, and θ is the heading angle, is first generically decomposed into the product:

$$P(x, y, \theta) = P(\theta)P(x, y|\theta) = \sum_i P(\theta_i)P(x, y|\theta_i) \tag{2}$$

We model the distribution $P(\theta)$ as a discrete set of weighted samples $\{\theta_i\}$, and the conditional likelihood $P(x, y|\theta)$ as simple two-dimensional Gaussians. This approach has the advantage of combining discrete Markov updates for the heading angle with Kalman filter updates for the translational degrees of freedom.

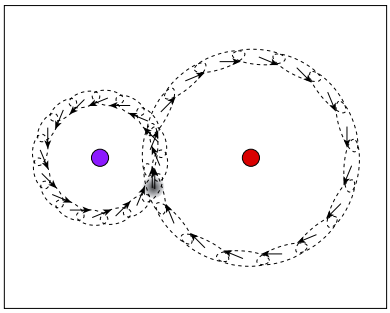

Fig. 3. Hybrid probabilistic representation used for localization.

When the algorithm is implemented on the robots, they are able to quickly incorporate visual landmarks and motion information to consistently estimate both the heading angle and translational coordinations on the field as shown in Figure 3. Even after the robots are lifted (kidnapped) by the referees, they are able to quickly relocalize their positions when they see new visual cues.

5 Motion

The motion of the robots is controlled by a parameterized walk routine in addition to predetermined scripted motions. The implementation of the walk routine is similar to other teams, in that inverse kinematics is used to generate the motion of the robot feet in a rectangular pattern relative to the body [7]. The four legs are phased according to a dynamic trot gait with a 50% duty cycle. The parameters for the walk were optimized by tracking the motion of the robot using a magnetic tracker [8], and gradually modifying the parameters to maximize speed and stability.

5.1 Kicks

The kicks are fully scripted motions, each starting from a standard position that is compatible with the end of a walk cycle. Matlab was used to tweak the joint angles in order to develop a set of kicks that could be used to kick the ball forward as well as sideways using both the legs and head. The criteria for developing kicks were as follows:

1. They shouldn't damage the robot or turn it off under any circumstances.

2. They had to be very quick to execute (typically under 2 seconds).
3. They featured complete ball control at every instant until the ball was released.
4. If they failed to release the ball, they should return to a state where the robot maintained control of the ball.
5. They were optimized for either power or accuracy.

The final set of kicks used in the competition were quite successful in scoring from almost every position on the field, as well as knocking the ball from the other team's possession.

6 Behaviors

The behaviors of the robots on the field are implemented using an object-oriented Perl script. This script incorporated an event-driven state machine that generates different actions depending upon one of possible four roles that the robot was assigned:

Attack The attacking robot goes directly to the ball, as long as it wasn't in the defensive penalty box.

Defend The defending robot positions itself between the ball and defensive goal area.

Support The supporting robot positions itself in an area away from the ball that would assist in scoring.

Goalie The goalie stays near the defensive goal to clear the ball when it comes close.

6.1 Potential Fields

Potential fields are used in all the roles to guide the robots to their optimal positions on the field. As shown in Figure 4, the positioning of the robots are dependent upon the ball location relative to predetermined field locations. For example, the optimal position for the Support role is on an intermediate point between the ball and an offensive field position. Similarly, the Defend role is situated on an intermediate point between the ball and a defensive field position. With the exception of the Goalie, a repulsive potential field in the defensive penalty area prevents the robots from becoming an "illegal defender."

6.2 Role Switching

Although the Goalie remains staticly assigned, the three robots that are initially assigned the Attack, Support, and Defend roles can fluidly switch roles. Any of these three robots can actively switch to the Attack role if it clearly sees the ball and if it is not being suppressed from switching by a timer. Once a new robot switches to an Attack role, it sends a message over the network to the other two robots who take either a Support or Defend role based upon their relative

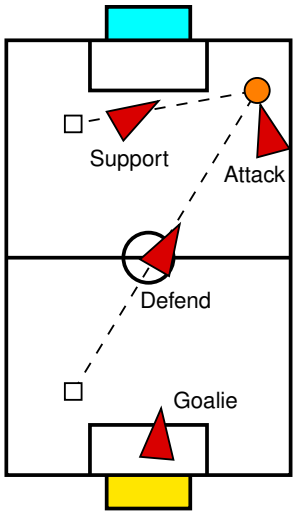

Fig. 4. Potential fields draw robots to their relevant field positions around the ball according to their roles.

position. The more forward positioned robot becomes Support, and the backward positioned robot takes the Defend role. At this point, the suppression timer on the Support and Defend robots are reset so that they cannot immediately take the Attack role.

As long as the Attack robot sees the ball and is making progress towards it, it will periodically send messages to the other robots to reset their suppression timers. By adjusting the values of the timer periods based upon relative distance to the ball, it is possible to ensure that the nearest robot to the ball ultimately becomes and stays the Attack robot. This simple role switching scheme worked well during the competition, with only a few bad role switches due to network congestion and delays.

7 Summary

The University of Pennsylvania 2003 Robocup team resulted in several innovations regarding software architecture and algorithms. In particular, rapid prototyping and development was made possible by incorporating high-level languages such as Matlab and Perl in the software design. These tools made it possible to develop new algorithms for line detection, localization, and team coordination. This development also coincides nicely with the team's research goals of investigating learning algorithms for sensorimotor processing and distributed robotics. By releasing the source code under the GNU public license, it is hoped that in the future, other teams will be able to build upon the successes of this team.

References

1. OPEN-R SDK. http://www.openr.org.
2. MATLAB. http://www.mathworks.com.
3. Comprehensive Perl Archive Network. http://www.cpan.org.
4. Tim Jennes and Simon Cozens. *Extending and Embedding Perl*. Manning Publications, 2002.
5. Manuela Veloso, Scott Lense, Douglas Vail, Maayan Roth, Ashley Stroupe, and Sonia Chernova. CMPack-02: CMU's Legged Robot Soccer Team, 2003. http://www.openr.org/robocup/code2002SDK/CMU/cmu_teamdesc.pdf.
6. S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141, 2000.
7. Spencer Chen, Martin Siu, Thomas Vogelgesang, Tak Fai Yik, Bernhard Hengst, Son Bao Pham, and Claude Sammut. The UNSW Robocup 2001 Sony Legged League Team. 2001. http://www.cse.unsw.edu/robocup/2002site.
8. Ascension Technology Corporation. http://www.ascension-tech.com.