RoboCup-99 Team Descriptions
Small Robots League, Team *BigRed*, pages 14–23
http://www.ep.liu.se/ea/cis/1999/006/03/

Description of Cornell BigRed Small League RoboCup Team

BigRed

Raffaello D'Andrea, Jin-Woo Lee

RD, JL: Mechanical & Aerospace Engineering, Cornell University

Abstract. *In this paper, we describe Cornell BigRed, the team of Robot Soccer competition that we developed to attend the RoboCup competition. The project entails the construction of fully autonomous, fast moving robots which will work together as a team in an effort to compete against similar teams of robots in a robotic soccer match.*

We designed and built the robots and the image processing algorithm, and developed the game strategy for the many situations and the artificial intelligence for collaboration between the robots. To provide a realistic testing platform for our artificial intelligence system, we have constructed a simulation of the playing field. The simulator has provided a means of testing the artificial intelligence play-by-play even before our robots were fully constructed.

1 Introduction

As engineering systems become more and more complex, there is an increasing need from industry for engineers who not only have expertise in a particular engineering discipline, but who also possess diverse interdisciplinary skills, can integrate system components, and can ensure total system operability. This skill set and process is often referred to as Systems Engineering (SE).

In order to effectively teach SE principles to students, a project course that embodies many of the key elements of SE, is being developed. The project entails the construction of fully autonomous, fast moving robots which will work together as a team in an effort to compete against similar teams of robots in a robotic soccer match. This yearly competition is known as the Robot World Cup Initiative, or RoboCup.

This paper outlines the main features of the Cornell RoboCup team. The paper is organized as follows. In Section 2, we describe the basic system architecture. We describe the electrical and mechanical aspects of the project, followed by a description of the global vision system. In Section 3, we describe the feedback control and filtering structure. The artificial intelligence subsystem is described in Section 4. We conclude with a description of the

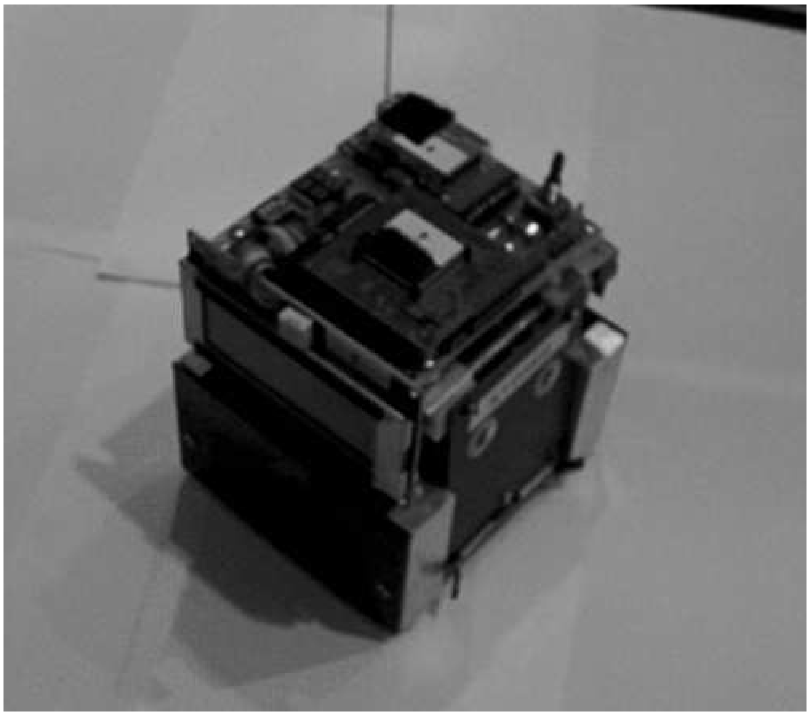

Figure 1: Picture of Robot

simulation model in Section 5, which has been developed to allow concurrent artificial intelligence and robot design.

2 System Architecture

The system consists of three main sub systems: the physical robots, the global vision system, and the artificial intelligence. Our global vision system is based on the Sony DXC-9000 camera, which captures the state of the field at a resolution of 640x480 and a frame rate of 60 Hz. A Matrox Genesis framegrabber, with an on-board DSP, receives the captured frame from the camera. A dedicated vision workstation identifies the ball and robot locations as well as the orientation for our team, and then pass the information to the AI workstation. Robot commands are dispatched from the AI workstation via a half duplex wireless transmitter. Each component of the system was designed to ensure a system sampling rate of sixty cycles per second.

2.1 Mechanical Design

The Cornell University team consists of two mechanical designs, one for our field player and another for our goalkeeper. All robots have a unidirectional kicking mechanism powered by a solenoid (two for the goalkeeper).

The field players have a design mass of 1.5 kg. Motors were selected to achieve a maximum linear acceleration of 5.1 m/s^2 and a maximum linear velocity of 2.5m/s using a six-to-one gear reduction. The desired maximum acceleration was arrived at using a weight transfer analysis assuming a coefficient of friction with the table of 1.0 and taking the height of our center of mess above the table as 6 cm.

The goalkeeper has a design mass of 1.5kg. Motors were selected to achieve a maximum linear acceleration of 5.8 m/s^2 and a maximum linear velocity of 1.25 m/s using a nine-to-one gear reduction.

2.2 Electrical Design

The on board electronics had several design goals. The main goal was for the robots to receive wireless left and right wheel velocity signals and have a local feedback loop to obtain the velocity quickly and accurately. The robot will also be able to receive different commands, such as engaging the kicking mechanism. The design had to be simple, low in power, small in size, and easy to use. The best way to handle the requirements was to implement everything all electronic with an on-board micro-controller. After a careful comparison of all the micro-controllers on the market we selected the Motorola 68HC16Z1.

The wireless communication design had to have enough bandwidth to send two 9 bit wheel velocities and a 2 bit action to five robot every 60 Hz. This is a total of $(2 \times 9 + 2) \times 5 = 100$ bits for a bandwidth of 750 bytes/sec.

2.3 Global Vision System

The vision system consists of two components, a DSP board for image processing and a host computer for target tracking. A digital camera is mounted above the field. A host computer with a frame grabber/DSP board performs the frame capture and all of the vision system processing. The extracted data is then sent over a network connection to the AI workstation.

The vision computer performs additional computation to identify the objects located by the image processing routines. The host computer is connected to the AI workstation with a UDP connection over 10 mbs Ethernet. The vision algorithm consists of four separate operations: frame capture, location, orientation, and identification.

The first task is to capture the image transmitted from the camera. The Matrox board takes a full 1/60th of a second to capture one frame at 640x480 resolution on 3 separate 8-bit color channels.

The second task is to locate the colored golf ball and ping-pong ball markers on the robots. The captured image is segmented by using a simple thresholding computation on each of the three color channels, and the resulting binary images are ANDed according to the color sought for each ball. Then the segmented images are searched for blobs. This search is localized based on the blob locations in the previous time step. These operations are performed as a separate thread of execution and runs concurrently with the capture of the next frame.

The third task is to determine the orientation of the team robots by pairing the location and orientation markers on each robot. The nearest orientation

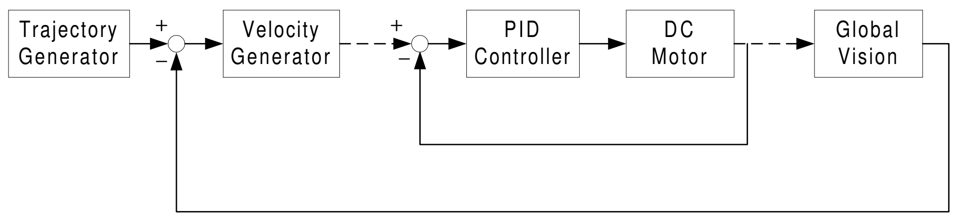

Figure 2: Schematic diagram for Robot Control

marker to each identification marker are paired together since the orientation markers are at a fixed and known distance from the orientation marker and smaller than the robot radius.

The fourth task is to determine the identity of each of the team robots. The robot locations are each paired with the robot locations in the previous frame to order the robot identification markers, which are initially unordered. The distances between robot identification markers for two successive frames is constrained to be within the distance specified by the robot top speed. A present robot identification marker is paired with a past identification marker such that the total global minimum difference is minimized.

Finally, the new robot and ball locations are sent to the artificial intelligence system via a UDP link. The system as currently implemented can achieve a throughput of 60 frames per second, with the total subsystem time elapsed (from shutter close to AI computer transmission) of approximately 30 milliseconds.

3 Feedback Control and Filtering Algorithm

Two feedback loops are employed for the robot's motion control. The first is a local feedback loop and the other a global feedback loop. The local feedback loop resides on the microcontroller of each robot and is in in charge of velocity control for the motors[1]. The global feedback control has position feedback loop through the global vision system. The global vision system gives the new position and the orientation of the robots. The desired trajectories to the target points and the real velocity of each motor of the robots are generated and then transmitted to the robots through the RF communication at every sixtieth of a second. Figure 1 is schematic diagram for the feedback control loop.

Data received from the vision system over the network contains noise which is rectified with filters for both the ball position and the team robot positions and orientations. Ball and the team robot filtering are performed with a $g-h$ filter[2], The basic equation of the $g-h$ filter for the robot's position and orientation is given by

$$\widehat{V}(t+1) = \widehat{V}(t) + \frac{h}{T_s}\left(X(t) - \widehat{X}(t)\right) \tag{1}$$

$$\widehat{X}(t+1) = \widehat{X}(t) + T_s \cdot \widehat{V}(t+1) + g \cdot \left(X(t) - \widehat{X}(t)\right) \tag{2}$$

where X is the measured position vector, $\widehat{X}$ and $\widehat{V}$ are the estimated position vector and the estimated velocity vector, T_s are the sampling time, and h and g are filter gains.

4 Artificial Intelligence Subsystem

The artificial intelligence subsystem is divided into two parts. The high-level AI takes the current game state (robot and ball positions, velocities, and game history) as input, and generates new targets and kicker commands for the robots as output. A target consists of a field location and a desired final state (time-to-target, robot orientation, and robot velocity). The low-level AI computes the trajectory to the target point and computes the wheel velocities to transmit to a robot.

4.1 Low-level AI (Trajectory Generation)

The job of low-level AI is to generate trajectories for a robot. It takes as inputs the current state of the robot and the desired ending state. The current state of a robot consists of the robot x and y coordinates, orientation, and left and right wheel velocities. A desired target state consists of the final x and y coordinates, final orientation, final velocity as well as the desired amount of time for the robot to reach the destination.

The low-level AI needs to be efficient and robust to imperfections in robot movement. Currently, our algorithm can run more than 60 times per one computation cycle, which is sufficient considering it only needs to be run at about four times per cycle (for four robots excluding the goalie).

This complex problem is solved by breaking the problem of trajectory generation into two parts. The first part generates a geometric path. The second part calculates wheel velocities such that the robot stays on the path and completes it in the allocated time.

4.1.1 Generating a geometric path

Our geometric path is represented by two polynomials in the x and y coordinates of the robots. The x coordinate polynomial is a fourth-degree polynomial and the y coordinate polynomial is third degree.

$$x(p) = \sum_{k=0}^{4} \alpha_k p^k \tag{3}$$

$$y(p) = \sum_{k=0}^{3} \beta_k p^k \tag{4}$$

The task is to solve for the nine polynomial coefficients for a particular path requirement. The 9 constraints on the polynomial path are: initial x coordinate, initial y coordinate, initial orientation, initial curvature (determined by the initial left and right wheel velocities), initial speed, final x

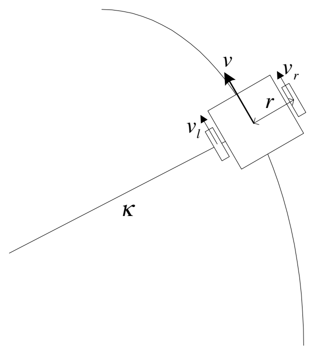

Figure 3: Trajectory Generation

coordinate, final y coordinate, final orientation, and final speed. It might seem strange that initial speed and final speed are constraints since they are not geometric features of the path, but we can actually transform them into the geometric feature of "path stiffness." For example, if the initial robot speed is very slow, the path beginning can change curvature very quickly (i.e. low "initial path stiffness") but if the initial speed is very high, the path beginning shouldn't change curvature too quickly (i.e. high "initial path stiffness").

4.1.2 Generating wheel velocities

Every point on the geometric curve has a curvature value, which defines a relationship between the left wheel velocity v_l and the right wheel velocity v_r at that point in the curve. This relationship is:

$$v = (v_l + v_r)/2 \tag{5}$$
$$v_l(1 + \kappa \cdot r) = v_r(1 - \kappa \cdot r) \tag{6}$$

where κ is the curvature of the path, and r is the half distance between the two wheels and v is the forward moving velocity of the robot(See Figure 3). Thus, we simply need to choose a forward moving velocity of the robot to solve for v_l and v_r at every point on the curve, which can then be sampled at the cycle rate of our AI system. Obviously, the forward moving velocity is constrained by the time-to-target as well as mechanical limits of the robot.

Even though each run of this algorithm generates a pre-planned path from beginning to end, it can be used to generate a new path after every few cycles to compensate for robot drift. The continuity of the paths generated is verified through testing. However, this algorithm breaks down when the robot is very near the target because the polynomial path generated might have severe curvature changes. In this case, the polynomials are artificially created (and not subject to the above constraints) on a case-by-case basis, and these are guaranteed to be smooth.

4.2 High-level AI

The artificial intelligence subsystem receives object coordinates, orientations, and velocities as inputs from the vision subsystem(See Figure 4).

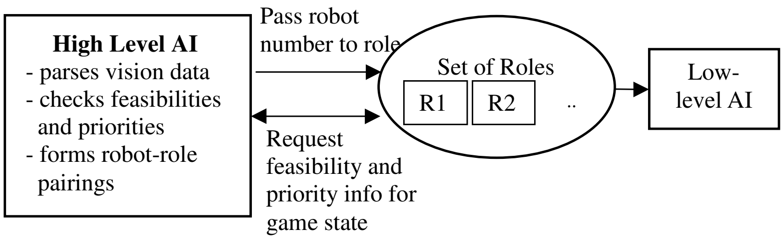

Figure 4: AI subsystem overview

This information defines an object state, which along with the game history defines the game state.

4.2.1 Roles

Definition A role is a set of functions that satisfy the following three properties:

- **Property A**
  - *Input:* the current game state
  - *Output:* a number representing the relative priority of execution for this role.
- **Property B**
  - *Input:* a robot number
  - *Output:* a number representing the relative feasibility of execution for this robot
- **Property C**
  - *Input:* the game state, and the robot number assigned to this role
  - *Output:* A target vector containing:
    1. target point on the field.
    2. desired ending orientation of the robot.
    3. desired no. of cycles in which the robot must reach 1 and 2.

Over time, a role specifies the behavior of a robot. A collection of offensive, defensive, and general roles, and the process of assigning them to robots sums up the high-level AI.

4.2.2 Example Roles

- **Defensive**
  - *Support Goalie:* cover areas of the goal not covered by the goalie
  - *Clear ball:* Go for the ball and knock it upfield
  - *Block opponent:* Block opponent's attackers

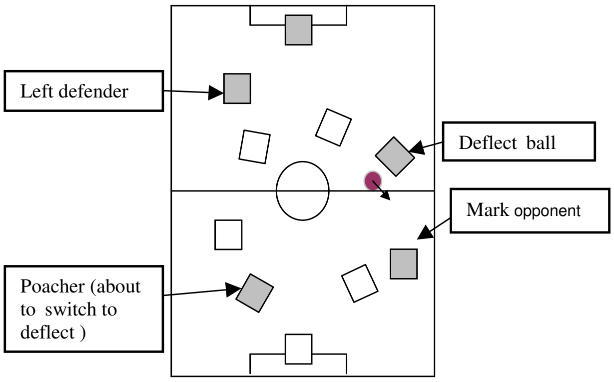

Figure 5: Example game state

- **Offensive**
  - *Shoot to score*
  - *Take penalty:* Wait for user input, rotate if required and use kicker
  - *Poach:* Wait in a suitable vicinity of the enemy goal
- **General**
  - *Mark opponent:* Keep position near specified opponent robot
  - *Intercept Ball:* Meet ball in order to 'trap' it
  - *Wait at specified position X:* Cover location X of the field
  - *Dribble ball to X*
  - *Shield ball:* protect ball from opponents till own robot arrives
  - *Various forms of 'Deflect ball':* knock ball into goal, to another robot (chain passes), or in a general direction.

4.2.3 Role-Role communication

Communication between roles allows them to dynamically adjust their behavior. Simple procedures (e.g. avoiding a shot path,) to complex maneuvers, (e.g. chained passes,) can then be implemented in a clean manner.

All roles have access to all current robot states, as well as current role-robot pairings. The role programmer must, of course, be aware of how other roles work to take advantage of the latter information.

4.2.4 Role-Robot Pairings

High-level AI maintains a list of roles that the four field players can assume. Every cycle, the high-level AI computes the priority and feasibility of each role as a function of the game state.

The high-level AI combines the roles' priority and feasibility information to generate role assignments for each robot. To perform this, it first sorts

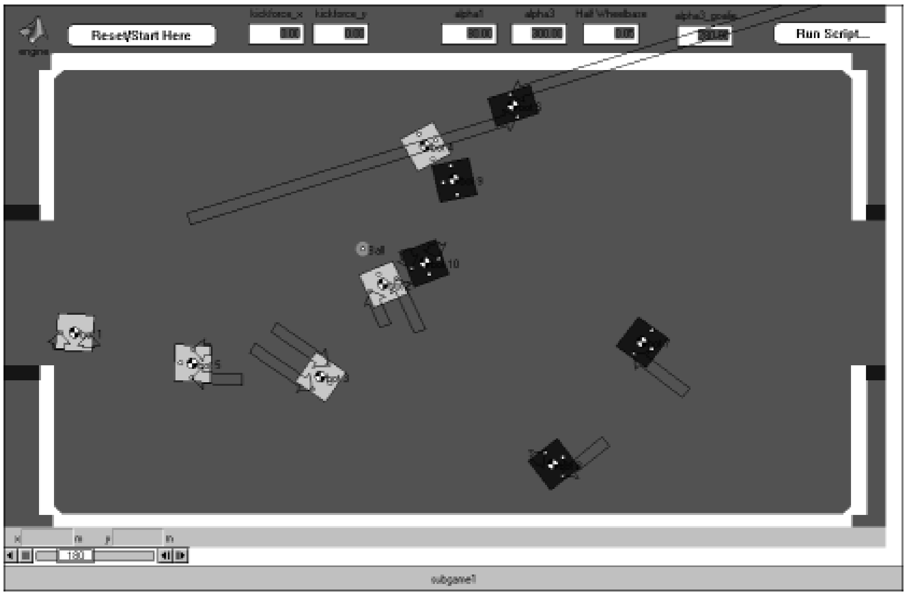

Figure 6: Testing platform for the Simulation

the roles by priority. Then, it rearranges the list of roles based on their feasibilities. Certain role-robot pairings are ruled out if determined to be completely infeasible.

For example, if the trajectory of the ball relative to a robot R is such that it is difficult to shoot it into the goal, not only will the feasibility value of this role for R be relatively low, it will reflect how difficult it is for R to shoot.

Finally, the high-level AI chooses the most suitable role assignments from the top of the list, and calls the respective role functions to give instructions to the robots.

4.2.5 Modularity

High-level AI has a general interface to all roles that is specified by the three properties mentioned earlier. Hence, a role programmer need only specify these three properties to add new roles to the system. However, role-role communication necessitates that roles are thoroughly documented, and their dependencies clearly specified.

5 Simulation

To provide a realistic testing platform for our artificial intelligence system, we have constructed a simulation of the playing field that models the dynamics of our environment.

The dynamic modeling of our system is performed by a Working Model 2D rendering of the complete playing field. The model includes two teams of five individual players, the game ball, and the playing field. Real world forces and constraints are modeled, including the modeling of the motion of the tires and the inertia of the robots and ball. Additionally, the physical interactions between the players and each other, the ball, and the playing environment are all modeled in the two dimensional environment.

The simulator accepts external input and output every 1/60th of a simulated second, the rate at which the information from the actual vision system is updated and robot commands are issued. To simulate the time lag and noise we encounter in our real world simulation, the Working Model parameters are passed into Matlab, where random noise, error, and delay are introduced to model the limitations of our the vision system. This information is then passed to the artificial intelligence module. Then, information that is normally fed back from the artificial intelligence module into our real-world robots is delayed and interpreted in Matlab before it is applied to the model of our system, to simulate the lag associated with our real world system.

The simulator has provided a means of testing the artificial intelligence play-by-play even before our robots were fully constructed. It highlights the real-world problems that exist in a dynamic system, and provides insight into solving these problems within an accurate representation of our playing environment. The simulator is a convenient, easy to use, and fairly accurate rendering of our real-world system.

6 Summary and Conclusion

As a whole, the system has successfully developed and built. In the mean time of building system, a simulator has been constructed to provide a realistic testing platform and it has provided a means of testing the strategy and artificial intelligence programming even before the real robots were constructed. RoboCup project provides insight into the same problems that we see in our real-world environment.

Acknowlegement

The authors would like to thank all of the members in Cornell RoboCup team for their help in the design and construction of the system, and Professor Bart Selman and Norman Tien for helpful advise and comments.

References

[1] J. L. Jones and A. M. Flynn, *Mobile robots: Inspiration to Implementation.* A K Peters, Wellesley, 1993.

[2] E. Brookner, *Tracking and Kalman Filtering Made Easy.* A Wiley-Interscience Publication, 1998.