RoboCup SSL Team Description, IRL RC

B. Sujith Kumar[1], Pratik Agarwal[2], P. Abhimanyu[3], Prem Bhargav[4]
and Dr. K. Madhava Krishna[5]
Robotics Research Lab, International Institute of Information Technology
[1] *sujithkumar@students.iiit.ac.in*
[2] *ag.pratik@gmail.com*
[3] *abhimanyu_p@students.iiit.ac.in*
[4] *bvvp_bhargav@students.iiit.ac.in*
[5] *mkrishna@iiit.ac.in*

***Abstract*—This paper describes the robotic system of IRL RC (IIIT-H Robotics Lab RoboCup) team participated in RoboCup,09 India. The whole system is divided into three main parts: Mechanical, Electrical and Software systems. A summary of all these systems is given in different sections. Path planning for mobile robotics has always been of interest to researchers. Rapidly Exploring Random Tree (RRT) has been used in the past decade extensively for path planning. Its advantage is obviously generating a path in the most randomly distributed environment. This paper also gives an algorithm for generating a smooth path using RRT in an environment containing rapidly moving obstacles.**

***Index Terms*—AI, RRT, Path Planning, Robot Soccer.**

I. Introduction

ROBOTICS is an area where many technologies and fields can be merged. It also develops the ability to solve real time problems. In this paper we explain about our robotic system that was developed for RoboCup SSL. RoboCup Small Size League is a competition where a team of five autonomous soccer playing robots compete with the other team and it was introduced for the first time in India as a national wide competition in 2009 by IIT-Kharagpur. This paper focuses on our SSL robotic system (past and present) touching upon many relevant aspects.

This paper is organized as follows: Section II describes the mechanical system of the robot. Section III describes the electrical and electronic elements involved. Section IV describes the software system focusing on two important parts namely the vision system and the decision system. Section V discusses our path planning algorithm. Section VI deals with our current work.

II. Mechanical System

The robot is omnidirectional with four custom made Swedish wheels. Any two wheels are separated by an angle of 120^O or 60^O. Each wheel contains sixteen symmetrically aligned similar rollers and each roller is parallel to the axis of rotation of the wheel to give an extra degree of freedom. These wheels are attached to stepper motors of the type 16PU-M301-G1. The robot's CAD is given in Fig. 2 and the original robot's picture is given in Fig. 1. The robots well fit into the restrictions of SSL rules. Each robot has a diameter of 179 mm and a height of 150 mm.

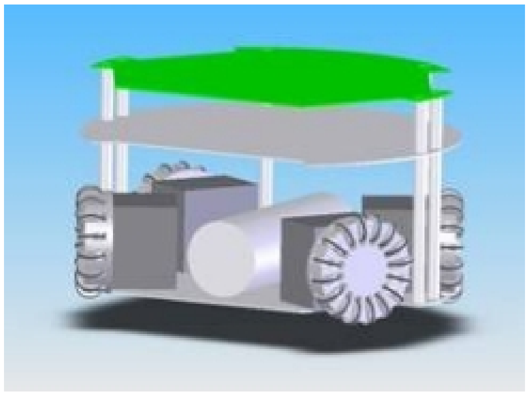


Fig. 1. CAD model of the Robot

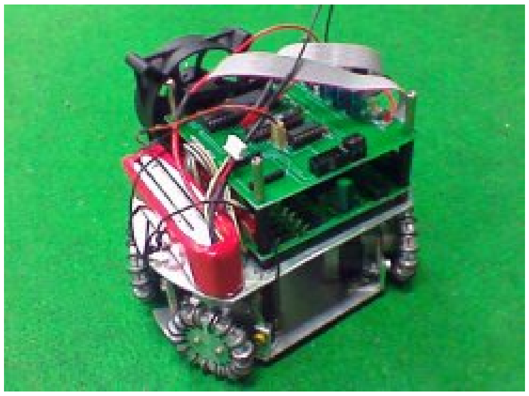


Fig. 2. Omni direction robot

III. Electrical System

The central controller consists of one ATMega16 and one ATMega8 microcontrollers. ATMega16 communicates with the Maxstream XBee Seies 2 modules at a baud rate of 9600 bps. These modules are low power RF devices which work on Zigbee protocol. Each of the RF devices is embedded in each bot and they communicate with the XBee module connected to the Decision System. The command from the Decision system to the bot contains the information about the speed of each wheel, dibbling on/off and kicking on/off. ATMega8 generates required signals for the dribbling and kicking. It acts a slave to the main controller ATMega16. The stepper motors are driven by L298-L297 pairs. The driver circuit for the stepper motor consists of four L298-L297 pairs connected to ATMega16. The servo motor and the dribbler roller motor are driven by one L293D connected to ATMega8.

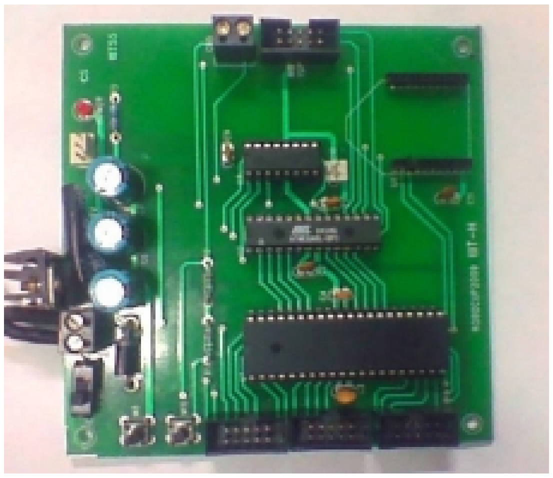


Fig. 3. Central controller

Our custom made double layered PCBs for the central controller and the driver circuits are given in Fig. 3. The whole electrical system is driven by a 11.1 V, 2200 mAh Li-Po battery. To cool the electrical system we use a CPU fan which consists of 12V DC brushless motor.

IV. Software System

Our software system is divided into two different entities; one is Vision System and the other one is Decision System. They communicate with each other using UDP through socket programming. Their descriptions are given below.

A. Vision System

We are using Unibrain Fire-I firewire digital camera (IEEE 1394) which supports 30 fps in the mode YUV 4:1:1 with a resolution of 640 X 480. We are using openCV open source library for all the vision process. Our algorithm first finds the centers of the robots and the ball in a frame using thresholds in YUV space discarding the Y channel to compensate the effects caused by the illumination changes. Our algorithm is based on the algorithm described in [1] which can determine the ids of the robots and their orientations robustly. After finding the centers of each robot the vision system has to differentiate the bots using to the extra coloured markers around the center markers. These center markers of are used to detect the robots. Example of different robot tops are shown in Fig. 4. It also has to determine the orientations. We've implemented an algorithm which can do both these jobs at a shot. It involves the following steps:

1. Taking circular data around the center markers and converting them into 1-dimensional signals
2. Identifying the bots using Minimum of Minimum Mean Squares
3. Determining the orientation of each robot using Minimum Mean Squares obtained in Step2.

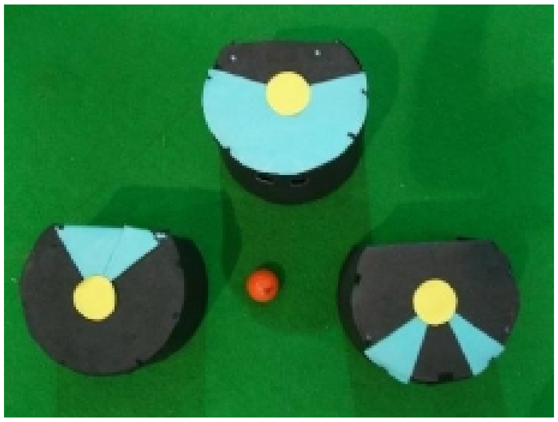


Fig. 4. Markers on top of each robot

This system sends the bots' positions, their orientations and the ball's position to the Decision System in a UDP using socket programming in Linux. We've chosen UDP because it is connectionless and it does not do error correction and flow control as in TCP which is slower compared to UDP.

B. Decision System

The decision system receives the information, described above, from the vision system and sends appropriate commands to the robots wirelessly. Our decision system along with the controlling of the on filed bots is divided into four layers. The bottom layer, layer 1, consists of sending particular control commands, like frequency of the stepper motor steps, to the bots using serial communication and the XBee devices. Next level layer, layer 2, uses the layer 1 and sends particular control commands like setting the wheel velocities independently. Layer 3 consists of commands which are used to move the robot at a particular speed in a particular direction with particular angular velocity about its centre. The strategic part is in Layer 4. The advantage of doing this is that even if we change the robot's type (differential drive or omni directional) we have to change only the Layer 1 while the remaining layers will remain intact.

V. RRT

Path planning for mobile robotics has always been of interest to researchers. Rapidly Exploring Random Tree (RRT) [2] has been used in the past decade extensively for path planning. Its advantage is obviously generating a path in the most randomly distributed environment. Below we have explained the RRT algorithm and modified it for generating a smooth path using RRT in an environment containing rapidly moving obstacles.

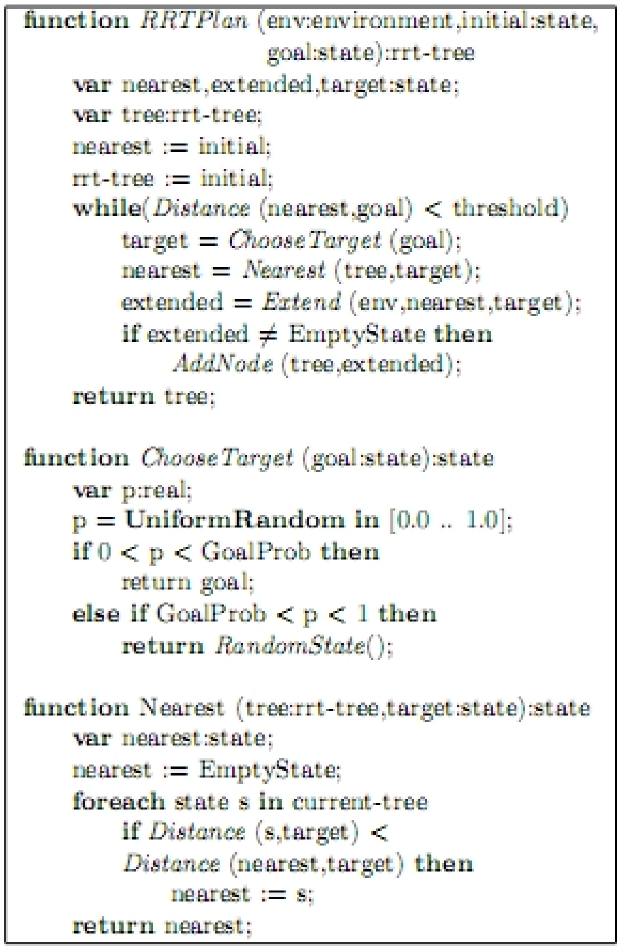
We have generated this algorithm for mainly assisting us in path planning for our non holonomic robot for RoboSoccer. RRT [2] gives a path plan according to the present situation of obstacles. So for a dynamic environment such as roboSoccer where robots can move with a velocity higher than 5 m/s in a confined region of 6m by 4m the best way to have an obstacle free path is to generate the path again and again for a fixed interval of time.

There are techniques such as using goal bias [3] and a way point bias [6] to reduce the computation but still the need to generate the path each time is some times redundant. [6] had addressed the problem of path planning in dynamic environment using RRT by introducing a way-point cache and by implementing a beta search but it by no means reduces the burden of generating the paths repeatedly. Our algorithm improves the planning significantly for robots in a rapidly changing dynamic environment significantly. RRT inherently gives us a path which is based on random points and hence the path achieved will always be very crooked and jagged. We present here an algorithm which minimizes the generation of new paths and it also smoothens the path followed by the robot.

Our algorithm computes the path using a way point cache but only when the current path is not feasible. It then tells the robot to go furthest point on the computed path till which it can go in a straight line without hitting any obstacles according to current situation. After each time interval it computes the next furthest point which is possible to reach. Once it reaches this point it recomputes the path similar to way point cache method as described by [6].

A. RRT Goal Bias

In a naive RRT path generation we find a random point on the map and find the nearest node on our tree to this point. Initially the tree contains only the start point as the root node. We attach a new node to this closest node of the tree if it is possible to move by a fixed distance (METRIC) in the direction of this random point from our closest node. This is the most general RRT algorithm. Next a goal bias was introduced to bias the path generated to the destination. Here we generate the path with random points with a probability p and with the goal as probability $1-p$. The algorithm for this is mentioned beside.



```
function RRTPlan (env:environment,initial:state,
                  goal:state):rrt-tree
    var nearest,extended,target:state;
    var tree:rrt-tree;
    nearest := initial;
    rrt-tree := initial;
    while(Distance (nearest,goal) < threshold)
        target = ChooseTarget (goal);
        nearest = Nearest (tree,target);
        extended = Extend (env,nearest,target);
        if extended ≠ EmptyState then
            AddNode (tree,extended);
    return tree;

function ChooseTarget (goal:state):state
    var p:real;
    p = UniformRandom in [0.0 .. 1.0];
    if 0 < p < GoalProb then
        return goal;
    else if GoalProb < p < 1 then
        return RandomState();

function Nearest (tree:rrt-tree,target:state):state
    var nearest:state;
    nearest := EmptyState;
    foreach state s in current-tree
        if Distance (s,target) <
        Distance (nearest,target) then
            nearest := s;
    return nearest;
```

Fig. 5. Algorithm for Goalbiased RRT

B. RRT Waypoint Bias

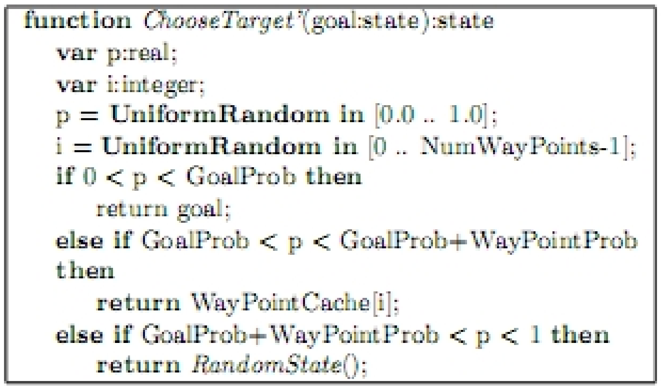
This algorithm was further improved by [6] by implementing a waypoint cache. They added a third probability and hence for replanning the tree extended by a probability p towards a random point, a probability q towards the goal and a probability of 1-p-q towards a randomly chosen waypoint which is a point on the previous path generated. They recompute the path after each time interval.

C. Modified Smooth RRT

We use the same algorithm as described by [6] for generating the path for the first time or if the destination changes. We introduce a point call the current destination. Current destination is a furthest point on the generated path till which the robot can go in a straight line without a collision. We will move the robot with a velocity so that it can reach this point. The advantage of this is that too many turns are avoided. At each time interval we just need to update the current destination and assign the required velocities to the robot.

Here the path in red shows the actual path generated by the RRT algorithm. The black circle is the furthest point where the robot can reach by travelling in a straight line without hitting any obstacle in the current situation. We give this point as the current destination. As the situation changes the current destination will also change. At each time cycle we find the furthest current destination which can be achieved according to the current configuration. We use this generated path repeatedly till using the same path we can reach the required destination. The velocities required to reach this current destination(black circle) is calculated and achieved by the bot.

The first obvious advantage of this method is that smoothens the path traversed by the robot. The robot tries to travel in a straight line as much as possible. The second advantage is that regeneration of path is minimized as much as possible. It is possible that some time in future it may be feasible to go from a certain location to another which currently is obstructed. By using this method we maybe able to direct a shortcut from one way point to another though the path generated on the rrt does not have it.



```
function ChooseTarget(goal:state):state
    var p:real;
    var i:integer;
    p = UniformRandom in [0.0 .. 1.0];
    i = UniformRandom in [0 .. NumWayPoints-1];
    if 0 < p < GoalProb then
        return goal;
    else if GoalProb < p < GoalProb+WayPointProb
    then
        return WayPointCache[i];
    else if GoalProb+WayPointProb < p < 1 then
        return RandomState();
```

Fig. 6. Algorithm for WayPoint biased RRT

[5] uses a similar technique to smoothen the path generated from RRT by putting the waypoints on a stack. He converts this path into line segments which reduces the jaggedness into few straight lines. But his algorithm is for static obstacles. We do not need to convert the whole path into few straight lines because we never end up travelling the generated path to the end as the environment changes dynamically.

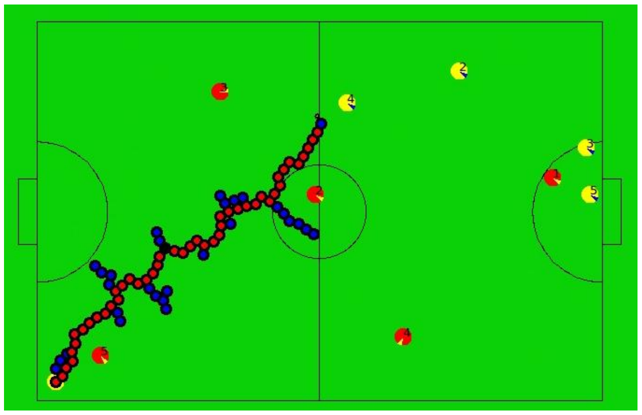


Fig. 7. RRT on Simulator

VI. Simulator

The simulator in another very important piece of code written in QT. It helps us simulate a game. The wireless module which in a real situation would communicate with the robots thinks it is still communication with the robot but instead it communicates with the simulator. The simulator has a physics engine which finds out if the robot undergoes a collision with the ball or with the wall or with other robots. It simulates a real environment as close as possible.

The results were tested on a simulator designed in QT. The simulator was written to test AI and strategy module and all the other codes without actually causing any harm to the hardware. The simulator incorporates a physics engine to simulate the collisions of robots with robots and the robots with the ball. When a ball collides with robot the simulator predicts the new velocities of the ball depending upon the parameters of the collision. The simulator was also used to test the path planning algorithm as shown in Fig. 7. The simulator receives the commands sent by the controller for the actual robots and simulates the environment. We send the velocities of the various robots and the simulator predicts the next position or performs collision restitution. In a real situation the strategy module would only interact with the vision module for data but for testing purposes we use the simulator.

VII. Current Work

Currently we are working on improving the hardware. On the software side we are working towards coming up on an intelligent strategy to play soccer. We will be using a Play and skill based strategy. Play represent the high level strategy. It can be compared to the formations we have in a normal game of soccer. In our method each robot will be assigned a role, where a role can be defender, attacker goalie etc. Since each of our robots are equally capable of performing each role the roles are dynamically assigned to each robot. Each robot can perform a group of skill. Each robot is assigned skills to be performed for the role assigned to it. For example the goalie may be assigned to keep tracking the ball and minimizing the open area for the goal. The attackers may be assigned to shoot if they are open or move to an open position. We can have various plays and various skills which we are still experimenting with. This technique is based on the technique used by the Cornell team [7].

References

[1] Shichi Shimizu, Tomoyuki Nagahashi, and Hironobu Fujiyoshi, "Robust and Accurate Detection of Object Orientation and ID without Color Segmentation".

[2] S. M. LaValle, Planning Algorithms. Cambridge University Press,2006.

[3] S. LaValle, "Rapidly-exploring random trees A new tool for path planning," Iowa State University, Dept. of Computer Science, Tech.Rep. 9811, 1998.

[4] S. M. LaValle, J. J. Kuffner, and Jr., "Randomized kinodynamic planning," The International Journal of Robotics Research, vol.20,no. 5, pp. 378400, 2001.

[5] Zoltan Deak Jnr., Professor Ray Jarvis, "Robotic Path Planning using Rapidly exploring Random Trees" Intelligent Robotics Research Centre Monash University

[6] Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: Proceedings of IROS-2002, Computer Science Department (2002)

[7] Raffaello D'Andrea., Tamas Kalmar-Nagy., Pritam Ganguly., Michael Babish, "The Cornell RoboCup Team".

[8] Veloso, M., Bowling, M., Achim, S., Han, K., Stone, P.: The CMUnited-98 cham- pion small robot team. In: RoboCup-98: Robot Soccer World Cup II, Springer Verlag (1999)

[9] Browning, B., Bruce, J.R., Bowling, M., Veloso, M.: STP: Skills tactics and plans for multi-robot control in adversarial environments. In: Journal of System and Control Engineering. (2005)

[10] Extended Team Description for RoboCup 2009, B-Smart. Deutsches Forschungszentrum fur Kunstliche Intelligenz GmbH, Germany.

[11] CMDragons 2009 Extended Team Description, Carnegie Mellon University, USA.

[12] Plasma-Z Extended Team Description, Chulalongkorn University, Thailand.

[13] Michael Bowling, Brett Browning, Allen Chang and Manuela Veloso: "Plays as Team Plans for Coordination and Adaptation"