Modelling and Analysis of the behavior of a Robotic Agent in Robot Soccer using Petri Nets

Sunil Kumar Rajendran
Graduate Student, Electrical Engineering
George Mason University
Fairfax, Virginia

Venkata Sasikiran Veeramachaneni
Graduate Student, Electrical Engineering
George Mason University
Fairfax, Virginia

Dr. Syed Abbas K. Zaidi
Research Associate Professor
Department of Electrical and Computer Engineering
George Mason University
Fairfax, Virginia

***Abstract*—Robot soccer is a league where teams are formed by autonomous robots. RoboCup is an annual international robotics competition which aims at promoting robotics and AI research, by offering a publicly appealing, nevertheless formidable challenge. Established in 1997, the ultimate goal is to develop a robot soccer team that beats the human world champion team by 2050. Robot Soccer is one of the leagues organized by RoboCup and has been subcategorized based on the size of robots. With plethora of simulation software available in the market, it is possible to conduct research and run simulations on different models and processes involved among the robotic agents. However, when all the processes are viewed on higher level, they are all discrete events and a common framework which can handle such discrete events and be customizable according to research is needed. This paper proposes a robot soccer framework of one robotic agent that can be customizable based on the needs of the researching team. This paper particularly focuses more on the humanoid league of RoboCup.**

Keywords—Robot soccer, framework, petri nets, walking, kicking, ball, obstacle avoidance

I. Introduction

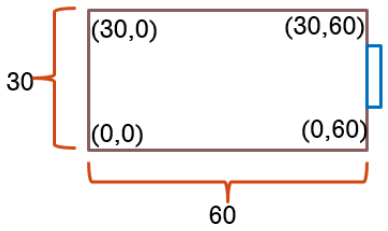
In this paper, we attempt to address the need of a platform which involves mainly about the study of robot soccer team with respect to discrete events. We modelled and analyzed the behavior of a robotic agent in robot soccer. This involved discretization of all the necessary components and actions related to the robotic agent: soccer field, ball dynamics, walking steps, kicking the ball, to name a few.

Using Petri-Nets we were able to translate our discretized model into a state-event model. Several scenarios have been tested like: robot walking towards the ball and kicking the ball towards the goal in the presence of obstacles in either paths. The entire model is built based the consideration that all the actions and movement are restricted to a 2-dimensional platform.

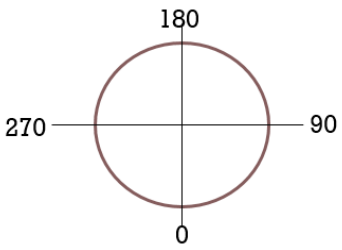
II. System

A. Overview

Firstly, a simplified version of real world robot soccer has been considered. A single robot is placed in a soccer field of length 60 units along the y-axis and breadth 30 units along the x-axis. The goal is considered the line segment between the coordinates (12.5,60) and (17.5,60). Since only one robot's behavior is modeled here, only the opponent's goal has been considered. However, the robot sees the goal with respect to the center point of the goal which is (15,60) and later decides which part of the goal to kick the ball. The axes have been inverted in the below picture for better viewing purpose.

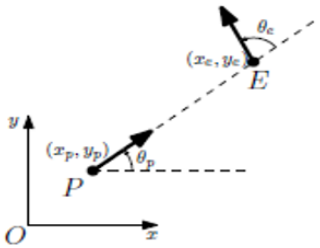


The ball is represented by a token with 'x', 'y', and 'field' values where first two represent the 2-d coordinates and the field value is a Boolean variable which is marked true if the ball is within the field and marked false otherwise. Robot is represented by a token with 'x', 'y', 'theta' values where the first two represent the 2-d coordinates and the theta value represents the orientation of the robot which basically ranges from 0 to 360 degrees.



B. *Functionality*

The process commences firstly by a robot tracking the ball followed by walking towards it. Once the robot reaches the ball, it kicks the ball towards the direction of the goal point. Each time the robot kicks the ball, the robot keeps moving by TravelDist units and the ball's position is incremented by BallDist units. Here ball dynamics have been simplified as such the ball stops after it has traveled BallDist units, hence discretizing the inertial movement.



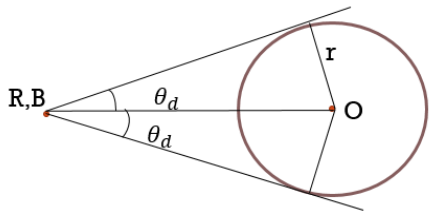
Robot path is planned based on the classical pursuit dynamics. Below are the equations which were used to calculate the new 'x' and 'y' coordinates when robot follows the path towards the ball and the goal.

$$\begin{aligned}\dot{r}_p &= u_p \\ \dot{r}_e &= u_e\end{aligned} \quad u_p(t) = v_p \frac{r_e(t) - r_p(t)}{\|r_e(t) - r_p(t)\|}$$

$$\dot{x}_p = \cos\theta_p, \quad \dot{x}_e = \nu\cos(\theta_e + \theta_p),$$
$$\dot{y}_p = \sin\theta_p, \quad \dot{y}_e = \nu\sin(\theta_e + \theta_p).$$

This process repeats until the ball is kicked into the region of the goal point, which is the circular region around the ball point with a radius of BallDist units. When the goal point is within the ball's observable region, it chooses the final goal point with x-coordinate generated by a Gaussian function. The function has been written such that value returned is mostly within the goal post's value and with less possibility of missing the goal. The y-coordinate is the goal point's y-coordinate.

In the event of an obstacle faced, it deviates and chooses a new path. If the ball is within the obstacle's region which is marked by a circular region of radius ObsRad units then the either of the points which fall on the circumferential region of the obstacle is considered and the maximum theta is considered. Similarly, when the robot is in possession of the ball, ObsKickRad is considered so that the ball is kicked outside the circular region of the obstacle.



Still-tracking of ball takes one time unit and parallel-tracking of ball does not take any time as it is a concurrent process. Once the robot scores a goal, the system reaches its final state.

C. *Parameters*

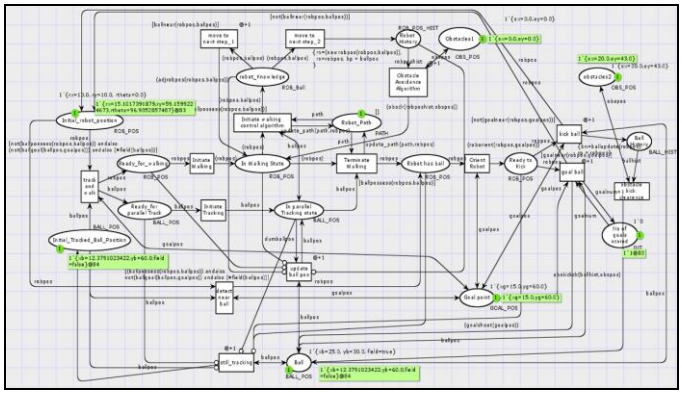
Below are the parameters that we used in our system:

No.	Parameter	Value
1	TravelDist	1 unit
2	BallDist	2 units
3	ObsRad	1 unit
4	ObsKickRad	2 units

III. Petri Net

A. *Structure*

The places in the petri net represent the states of the robot and/or ball. Numerous functions have been written in SML language to reduce the complexity of the petri net model and at the same time not all the functionalities have been coded because some had to be implemented using the petri-net itself to maintain the framework. Several color sets have been defined like robot's location, ball's location on the soccer field. Inhibitor arcs have been used to restrict ball position update still tracking and parallel tracking functions do not conflict with one another.



Petri-Net Model of one Robotic Agent

B. *Functions*

The table below contains the list of functions used and a short description of its purpose:

Function	Description
convrad(x)	Converts given value to radians
convdeg(x)	Converts given value to degrees
stepx(theta)	Returns step value in x axis – Math.cos
stepy(theta)	Returns step value in y axis – Math.sin
incX(x,theta)	Increments the x or y value
decX(x,theta)	Decrements the x or y value
update_path(path,	Updates path list with new robot

robotposition)	position
thetacalc(xr,yr, xb,yb)	Returns the angular difference between the robot and the ball
newrobpos(robpos, ballpos)	Returns the new robot position given the robot and ball position
adjrobpos(robpos, ballpos)	Adjust the robot's position if the ball is very near to the robot
ballnear(robpos, ballpos)	Boolean function checks if the ball is near the robot
ballpossess(robpos, ballpos)	Boolean function checks if the robot possesses the ball
roborient(robpos, ballpos)	Orients the robot such that to face the goal when is possession of the ball
ballupdate(robpos)	Updates ball's coordinates after the robot kicks
goalnear(robpos, ballpos)	Boolean function checks if robot and ball are near the goal
goalshoot(goalpos)	Returns 'x' value generated by Gaussian distribution with center of the goal as mean and with a variance of 1.0
devtheta(robpos, obspos)	Returns an angle which deviates the robot from the obstacle
add/subtheta(robpos, real)	Returns the added or subtracted theta value during obstacle avoidance
newrobpostest(robpos, obspos)	Boolean function checks if robot's newly calculated position is in the obstacle's circular region
add/subdevpos(robpos, obspos)	Adds or subtracts the robot position while deviating
obsclr(robposhist, obspos)	Given robot's previous position, calculates the new position of the robot when an obstacle is present
devkicktheta(robpos, obspos)	Returns the deviated kick theta
newballpostest(ballpos, obspos)	Boolean function checks if the projected ball coordinates fall in the obstacle's circular region
add/subballpos(robpos, obspos)	Adds or subtracts the ball's position while deviating
obskickclr(ballhist, obspos)	Given ball's previous position, calculates the new position of the ball when an obstacle is present nearby, after the robot kicks

Certain functionalities have been partitions and categorized in order to reduce the coding complexity and encourage code reuse. Below are a few screenshots of the function which calculates the new robot coordinates when the robot is near an obstacle. The function 'obsclr' uses three other functions to calculate the new coordinates: 'newrobpostes', 'adddevpos', 'subdevpos'.

```
fun obsclr(x:ROB_POS_HIST,y:OBS_POS)=
  if  not(((Math.pow ((#rx(#ro(x)) - #ox(y)), 2.0))+
        (Math.pow ((#ry(#ro(x)) - #oy(y)), 2.0))) < (Math.pow (ObsRad, 2.0)))
    then
     (if (((Math.pow ((#rx(#rn(x)) - #ox(y)), 2.0))+
        (Math.pow ((#ry(#rn(x)) - #oy(y)), 2.0))) < (Math.pow (ObsRad, 2.0)))
       andalso
       not(((Math.pow ((#xb(#bp(x)) - #ox(y)), 2.0))+
        (Math.pow ((#yb(#bp(x)) - #oy(y)), 2.0))) < (Math.pow (ObsRad, 2.0)))
     then
       (if (newrobpostest(adddevpos((#ro(x)),y),y))
        then
          subdevpos(#ro(x),y)
        else
          adddevpos(#ro(x),y)
       )
     else
        {rx=(#rx(#rn(x))),
         ry=(#ry(#rn(x))),
         rtheta=(#rtheta(#rn(x)))
        })
  else
     {rx=(#rx(#rn(x))),
      ry=(#ry(#rn(x))),
      rtheta=(#rtheta(#rn(x)))
     }
  ;
```

A screenshot of 'obsclr' function

```
fun adddevpos(x:ROB_POS,y:OBS_POS)=
      {
      rx=(#rx(x)) + trunc(TravelDist*
           (Math.cos (convdeg
            (addtheta(x,devtheta(x,y))))),
      ry=(#ry(x)) + trunc(TravelDist*
           (Math.sin (convdeg
            (addtheta(x,devtheta(x,y))))),
      rtheta=addtheta(x,devtheta(x,y))
      }
  ;
```

```
fun subdevpos(x:ROB_POS,y:OBS_POS)=
      {
      rx=(#rx(x)) + trunc(TravelDist*
           (Math.cos (convdeg
            (subtheta(x,devtheta(x,y))))),
      ry=(#ry(x)) + trunc(TravelDist*
           (Math.sin (convdeg
            (subtheta(x,devtheta(x,y))))),
      rtheta=subtheta(x,devtheta(x,y))
      }
  ;
```

Screenshots of 'adddevpos' and 'subdevpos' functions

```
fun newrobpostest(x:ROB_POS,y:OBS_POS)=
      (((Math.pow (((#rx(x)) - #ox(y)), 2.0))+
        (Math.pow
          (((#ry(x)) - #oy(y)), 2.0))) <
          (Math.pow (ObsRad, 2.0)))
    ;
```

A screenshot of 'newrobpostest' function

IV. Analysis

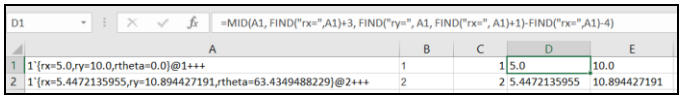
The robot's and ball's path coordinates have been generated in the petri-net model with unique time units assigned to each movement of the robot's and ball's tokens. This data has then been imported to Microsoft Excel spreadsheet and parsed using various formulae to obtain individual data elements like the coordinates and time units from the token bindings.

```
1`{rx=10.0,ry=10.0,rtheta=0.0}@1
+++
1`{rx=10.6,ry=10.8,rtheta=53.13010
23542}@2+++
1`{rx=11.2,ry=11.6,rtheta=53.13010
23542}@3+++
1`{rx=11.8,ry=12.4,rtheta=53.13010
23542}@4+++
1`{rx=12.4,ry=13.2,rtheta=53.13010
23542}@5+++
1`{rx=13.0,ry=14.0,rtheta=53.13010
23542}@6+++
```

Robot's path token bindings

```
1`{xb=12.3791023422,yb=60.0,field
=false}@84+++
1`{xb=15.1017391879,yb=59.15992
24673,field=true}@82+++
1`{xb=15.1017391879,yb=59.15992
24673,field=true}@81+++
1`{xb=15.1017391879,yb=59.15992
24673,field=true}@80+++
1`{xb=15.3421960356,yb=57.17442
9958,field=true}@79+++
1`{xb=15.3421960356,yb=57.17442
9958,field=true}@78+++
```

Ball's path token bindings

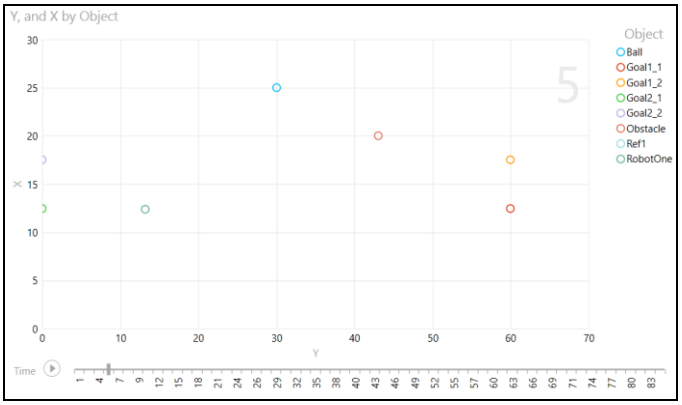


Formula on Excel to parse the data from the bindings

Object	Size	X	Y	Time
RobotOne	0	5	10	1
RobotOne	0	5.447213596	10.89442719	2
RobotOne	0	5.894427191	11.78885438	3
RobotOne	0	6.341640787	12.68328157	4
RobotOne	0	6.788854382	13.57770876	5

Result of parsing the data

Using Power View function in Excel, a scatter/bubble chart has been generated where each point or bubble on the chart represents an object on the robot soccer field. The axes have been inverted for better viewing. The number viewed on the top right corner is the number of time units passed.



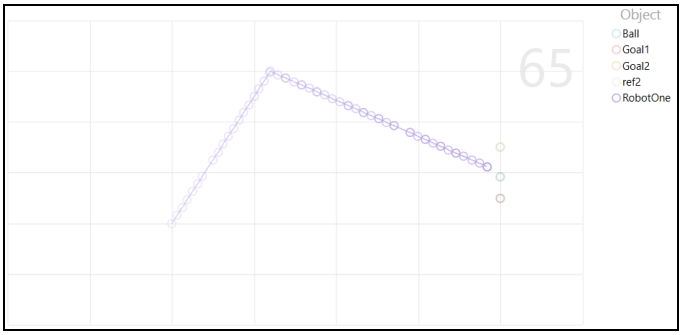
A bubble/scatter chart representing the objects with legend

Four different scenarios have been tested. Firstly, a robot reaching the ball and kicking it towards the goal, then with static obstacles present in between the robot, ball and goal.

A. Test Scenario 1:

This analysis is to study the path taken by the robot to reach the ball and subsequently reach the goal.

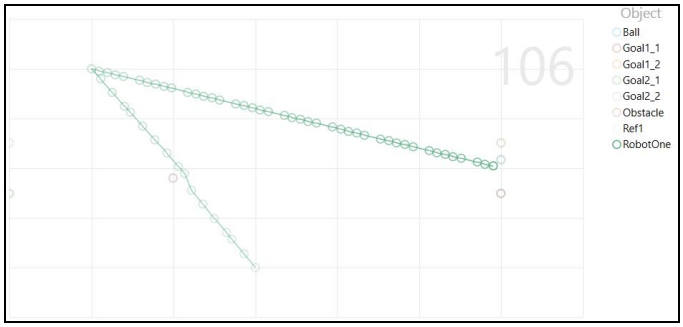
Object	Coordinates
Robot	(10,20)
Ball	(25,32)
Goal Post 1	(12.5,60)
Goal Post 2	(17.5,60)



B. Test Scenario 2:

This analysis is to study the path taken by the robot to avoid the obstacle when it is going to reach the ball.

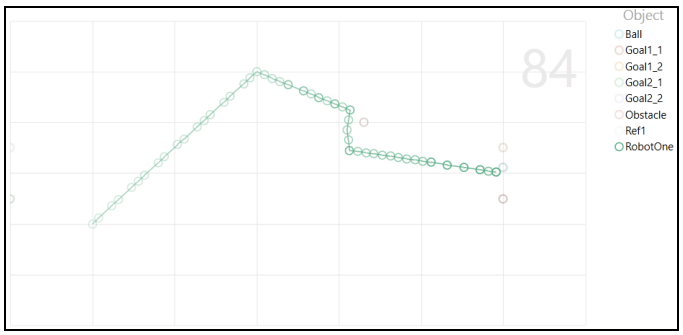
Object	Coordinates
Robot	(5,30)
Ball	(25,10)
Obstacle	(14,20)
Goal Post 1	(12.5,60)
Goal Post 2	(17.5,60)



C. Test Scenario 3:

This analysis is to study the path taken by the robot to avoid the obstacle when it kicks the ball towards the goal.

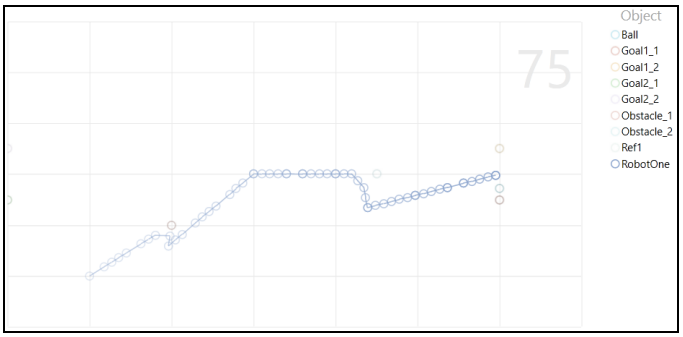
Object	Coordinates
Robot	(10,10)
Ball	(25,30)
Obstacle	(20,43)
Goal Post 1	(12.5,60)
Goal Post 2	(17.5,60)



D. Test Scenario 4:

This analysis is to study the path taken by the robot to avoid both the obstacles when it walks towards the ball and kicks the ball towards the goal.

Object	Coordinates
Robot	(5,10)
Ball	(15,30)
Obstacle 1	(10,20)
Obstacle 2	(15,45)
Goal Post 1	(12.5,60)
Goal Post 2	(17.5,60)



V. CONCLUSIONS

From the above results, we have shown the analysis of the behavior of a robotic agent in robot soccer as a discrete event system. From our observations, the robot's behavior with or without obstacles has been consistent and has met our expectations. This project can be further extended to a more detailed framework modelling every event in robot soccer and thus widening the scope of users who can use this for different research purposes within robot soccer.

REFERENCES

[1] P.F. Palamara, V.A. Ziparo, L.Locchi, D. Nardi, P. Lima, H. Costelha, "A Robotic Soccer Passing Task Using Petri Net Plans" In Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), 2008.

[2] Limor Marciano, "CPNP: Colored Petri Net Representation of Single-Robot and Multi-Robot Plans" Department of Computer Science, Bar-Ilan University.