Design and Implementation of a Real Time Planner for Autonomous Robots

Ümit Deniz Uluşar and H. Levent Akın
Department of Computer Engineering ,
Boğaziçi University,
34342 Bebek, İstanbul, TURKEY
uulusar@eng.marmara.edu.tr
akin@boun.edu.tr

Abstract

This work proposes a system design for real time planning for autonomous robots. Implementing a real time autonomous planning agent which can solve complex problems is a hard to achieve task where the planning system design must be adapted to process in real time domain and should be able to return satisfactory results. The planning approach is developed on the robotic soccer domain and uses fast forward planner and was tested using the Teambots simulation environment.

1 Introduction

Implementing a real time autonomous planning agent which can solve complex problems is a hard to achieve task. In order to achieve this task, planning system design should be adapted to process in real time domain. A real time system can be defined as a system which can run in the worst resource and runtime conditions and can produce the best possible results. However, a real time planning agent can not do many of the important needs of real time systems so for planning domain, we need a less strict definition – A real time planning system will be any system that can return statistically satisfactory results by the required time [1].

In this work, the robotic soccer domain was chosen since soccer is the most popular sport of the world with 22 players and millions of fans and robotic soccer is a state of the art testing ground for AI and robotics research with several international competitions like Robocup and FIRA. In addition, robotic soccer requires the joint work of many AI subfields like vision, planning etc. Although planning is an important subfield of AI it was not widely used in the robotic soccer due to the real-time nature of the domain. However, with the advances in the planning algorithms and the processing power available on the robots, it is now worthwhile to asses the feasibility of integrating planning systems with autonomous robot systems and make them play robotic soccer.

In this paper integration of a general purpose planner with a simulated soccer playing robot team is described. The rest of the paper is organized as follows; information on the planning algorithm used and the Teambots simulation environment is given in section 2. The details of the implementation are given in section 3. In section 4, the experiments are described. The results of the experiments are stated in section 5 and the conclusions are given in section 6.

2 Background

2.1 Planning and Planners

Planning can be called as the task of coming up with a sequence of actions that will achieve a goal. [2] Usually a planning algorithm has three inputs, the first one is the description of the world called as initial state, the second is the description of the agent's goal called as the goal state and the last one is the description of the actions that the agent can perform, called the domain theory. And the planner's output is a sequence of actions which takes the agent from initial state and ends up with the goal state. The representation of states actions and goals require a language which is expressive enough to describe as many domains as possible but restrictive enough for the planning algorithms to handle. The basic representation language of a classical planner is the propositional STRIPS language. This language represents the states of the world with conjunctions of positive literals [2].

The simplest way to build a planner is to use state space search. Many heuristic and brute force search techniques can be immediately applicable to state space search based planning systems. One can use various search algorithms like breadth-first, depth-first, A* search or techniques like enforced hill climbing which is a combination of hill climbing search and breadth first search methods [3]to find the action sequence from initial state to the goal state. There are two types of state space search algorithms. The first one is forward state space search, also called as progression planning. This type of planners operates by searching forward from initial state to the goal state. The second one is backward state space search, also called as regression planning. This type of planner searches from goal state to initial state.

Every good search algorithm requires a good heuristic function, which estimates the distance from a state to goal by looking at the effect of actions to the states. One of the widely used admissible heuristic – one that does not overestimate – relaxed problem, which is based on generating a relaxed problem from the original problem and solve it.

Planning Graphs is an add-on for the planning algorithms which can be used to give better heuristic estimates [2] which can be described as a sequence of levels that corresponds time steps in the plan. Each level contains a set of literals and a set of actions.

2.2 Fast Forward Planner

Fast Forward (FF) planner is a general purpose planning system developed by Hoffmann [3]. The planner is implemented in C and had won various prizes in planning competitions. The FF is described as forward chaining heuristic state space planner which uses relaxed task general purpose heuristic and Graphplan to support enforced hill climbing search algorithm. In addition, some pruning and optimization techniques are used to decrease the branching factor of the search space.

2.3 TeamBots

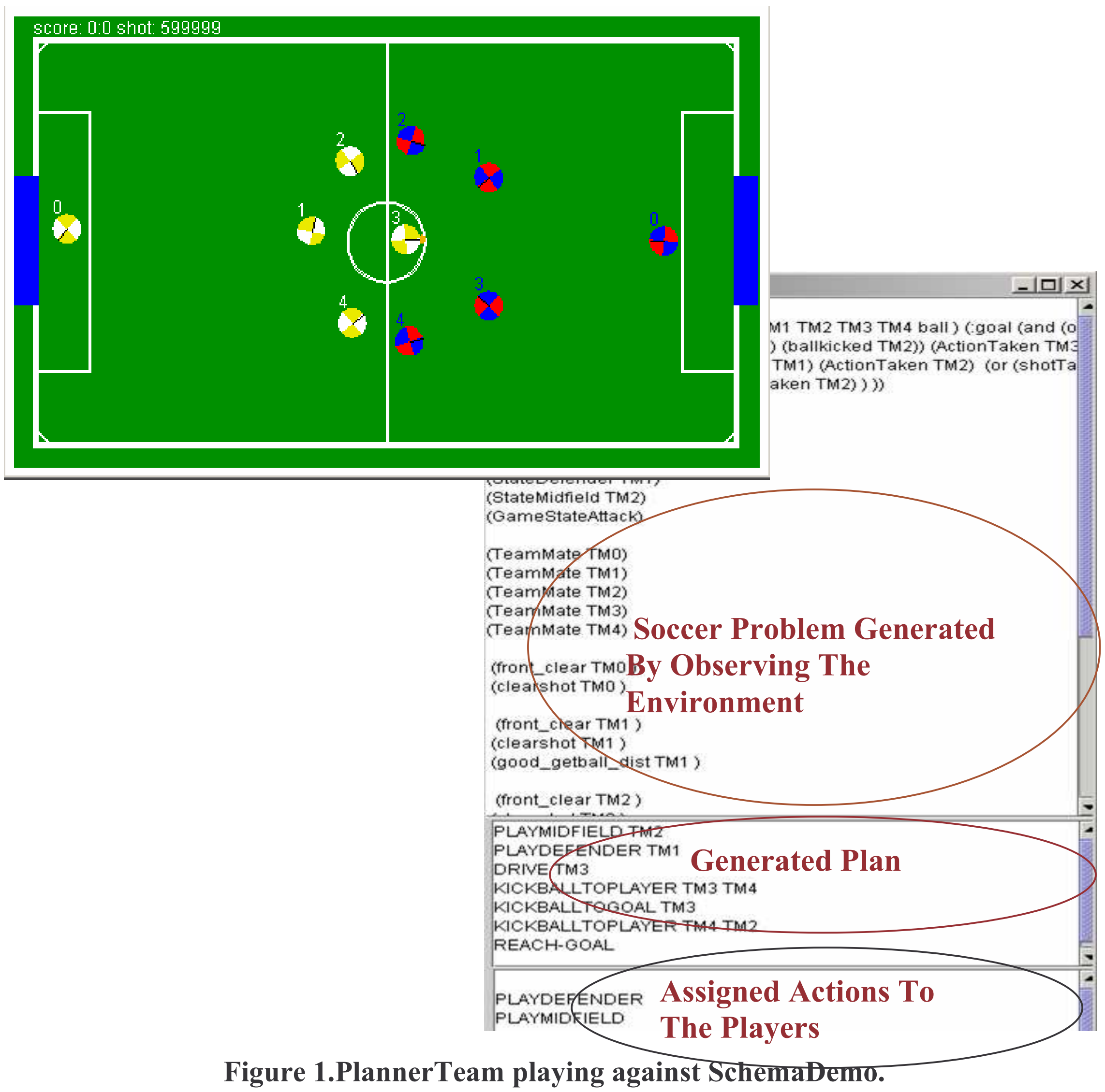
Teambots simulation environment is developed in Java programming language by Tucker Balch [5]. The simulation environment is designed for multiagent robotics. In each time step it asks the Robots to take their actions. Depending on the design the system, environment is dynamic or static. If all the planning process is done in each step then the simulation environment freezes until the planning process finishes. This type of system can be assumed as static. If there is another class extended from thread and the planning process is done by that thread then environment may be assumed as dynamic because agent does not have to wait for planner to finish planning process and while planning process takes place environment may change. There are various simulation domains and one of them is soccer. The simulation robot developers extend a base class called *ControlSS* and implement their robots.
The *TeamBots* simulation environment has the following environmental characteristics:

- **Fully Observable-** The environment is fully observable, every agent in the system can obtain the location of its teammates, opponents, ball, its own goal and opponent's goal.
- **Strategic-** The environment is deterministic except for the other agents.

- **Sequential -** The environment is sequential for two reasons, first the actions of the robot have effects on the future actions of robot and second, to take an action the simulation robot has to take many sub actions. For Example: If the agent wants to kick ball to goal then it has to come close to ball with a special angle so it can push ball to goal.
- **Multiagent** – Soccer requires team coordination and competition with an opponent team so the environment is multiagent and both competitive and cooperative.

3 Implementation

The *PlannerTeam* developed in this study is mainly composed of three modules. The first one is the fast forward planner and the second one is the problem file dynamically generated by the *PlannerTeam* class and the third component is the planning soccer domain.


score: 0:0 shot: 599999
Soccer Problem Generated By Observing The Environment
Generated Plan
Assigned Actions To The Players

```
M1 TM2 TM3 TM4 ball ) (:goal (and (o
) (ballkicked TM2)) (ActionTaken TM3
TM1) (ActionTaken TM2)  (or (shotTa
aken TM2) ) ))

(StateMidfield TM2)
(GameStateAttack)

(TeamMate TM0)
(TeamMate TM1)
(TeamMate TM2)
(TeamMate TM3)
(TeamMate TM4)

 (front_clear TM0 )
 (clearshot TM0 )

  (front_clear TM1 )
 (clearshot TM1 )
 (good_getball_dist TM1 )

  (front_clear TM2 )
```

```
PLAYMIDFIELD TM2
PLAYDEFENDER TM1
DRIVE TM3
KICKBALLTOPLAYER TM3 TM4
KICKBALLTOGOAL TM3
KICKBALLTOPLAYER TM4 TM2
REACH-GOAL
```

```
PLAYDEFENDER
PLAYMIDFIELD
```

Figure 1.PlannerTeam playing against SchemaDemo.

The *PlannerTeam* class observes the teammates, opponents and ball locations and updates its internal representation. With the obtained location information, it also updates environment variables listed below.

- *Clearshot*: If there is no player between the opponents goal and teammate.
- *Closeenough*: If the player is close to the opponents' goal.
- *closest_to_ball:* Determines the closest player to the ball.

- *goalarea_free:* Determines if the goal area is free. (There are at most 2 players in the goal area)
- *front_clear:* Determines if the front of the player is clear.
- *good_play_angle:* If the angle with the ball is suitable for action.
- *good_getball_dist:* Determines if the player is close enough to the ball.
- *good_getball_angle:* Determines if the player has good angle with the ball.
- *CanKick:* Determines if the player can kick the ball.
- *ActionTaken:* Determines if any action is set to the player. This is mainly required for goal definition.

The next process is to generate a planning problem file with the specified variables. Planning problem file is generated by the Planner class. The Planner class schedules replanning whenever necessary and obtains a plan by using FF planner. If the planner returns a plan, the planner class parses the plan and generates or updates the action list (see Figure 1).

3.1 The Soccer Planning Domain

The soccer domain determines the available actions of the simulation robot. In order to generate a plan, planners require a planning domain and a problem file. The domain is named with domain name which is ‘soccer’. An action is represented with four fields. The first one is for the name of the action. The second one is for the parameters; the third one is for the preconditions and the last one is for the effects of action (see Figure 2).

```
(:action KickBallToGoal
           :parameters (?player)
           :precondition  (and  (or  (good_getball_dist  ?player)  (CanKick
?player)) (clearShot ?player) )
           :effect   (and   (ballkicked   ?player)   (ActionTaken   ?player)
(shotTaken ?player) (not(CanKick ?player)) ) )
```

Figure 2. Example of a soccer action.

In order to visualize the available actions we can represent some of them similar to extended domain networks as shown in Figure 3. [4]

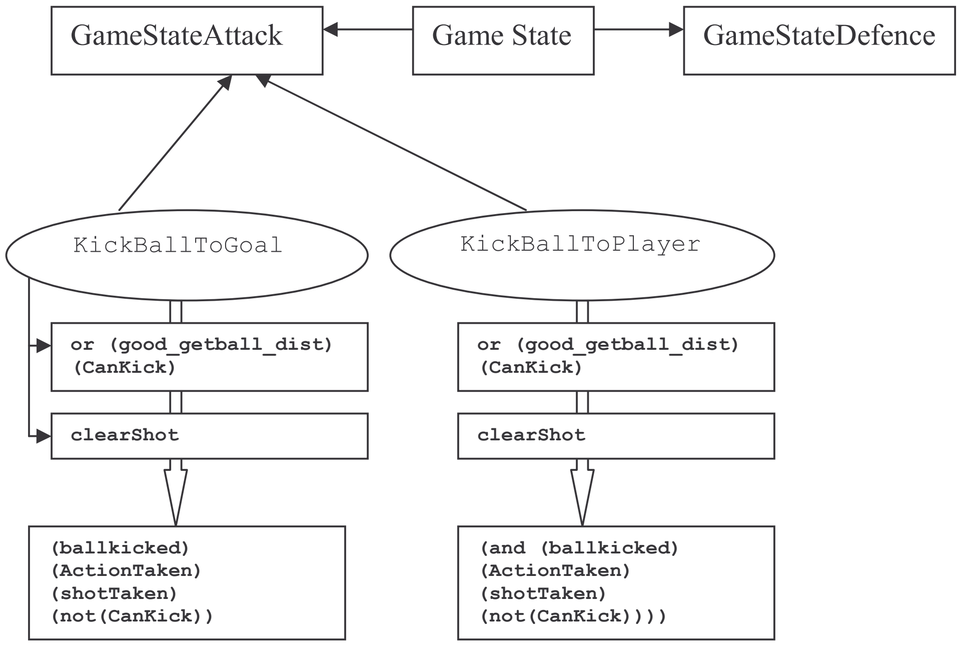

GameStateAttack
Game State
GameStateDefence
KickBallToGoal
KickBallToPlayer
or (good_getball_dist) (CanKick)
clearShot
(ballkicked) (ActionTaken) (shotTaken) (not(CanKick))
or (good_getball_dist) (CanKick)
clearShot
(and (ballkicked) (ActionTaken) (shotTaken) (not(CanKick)))

Figure 3. Plannnner Soccer Domain as extended domain networks

Other available actions are

- *Drive*: The attacker players can drive if their front is clear.
- *MoveToBall:* The closest player which is close enough moves to the ball.
- *GetBall:* If the player in a good position and angle gets the ball.
- *TakePositionAttack:* Depending on the strategy player takes attack position.
- *TakePositionDefense:* Depending on the strategy player takes defense position.
- *PlayMidfield:* Plays as a midfield player.
- *PlayDefender:* Plays as a defense player.

Both the action definitions and the goal can be changed dynamically. This leads us to generate adaptive domain and action definitions depending on the tactics of the opponents.

3.2 Planning Soccer Problem File

Planning Soccer Problem (PSP) file is generated by the planner class by observing the environment. PSP is composed of virtual environment variables which were listed before. After the problem file is generated, the *PlannerTeam* class asks the FF planer to generate a plan which can accomplish the goals. In most cases the FF generates a plan suitable for the current state but if it can not find a plan, the players start to chase the ball. The goal of the system is to score and is also defined in PSP as shown in Figure 4. We defined scoring as *shotTaken*. In order to take a shot the player should be in an appropriate position and with a suitable angle to the goal. Besides, all the active players should take at least one action.

```
(:goal  (and  (or  (ballkicked  TM3)  (ballkicked  TM4)  (ballkicked  TM2))
(ActionTaken  TM3)  (ActionTaken  TM4)  (ActionTaken  TM1)  (ActionTaken  TM2)
(or (shotTaken TM3) (shotTaken TM4) (shotTaken TM2) ) ))
```

Figure 4. Definition of goal

3.3 Team Coordination

Soccer is a complex game in which the soccer players have to consider many tasks at the same time. A soccer team has to defend its goal and if it is possible attack and score to the opponents goal. In order to achieve successful team coordination, team players should carry different roles and whenever necessary they have to position themselves at appropriate strategic positions on the field. Some roles assigned to the players are listed below.

- *StateAttackerLeft*: Is one of the attacker players.
- *StateAttackerRight*: Is the second attacker player.
- *StateDefender*: Is the defender player.
- *StateMidfield*: Is the midfield player.

The Planner generates no detailed actions so the *PlannerTeam* members should determine what is necessary to do to achieve the action given by the planner. Two of the complex tasks which can be given by planner are *takeAttackLocation* and the *takeDefenceLocation*. The player has to consider the optimum defense and attack positions [4].

3.4 Plan Commitment, Plan Failure and Re-planning

After a successful plan is generated, the actions of the players are stored in an action link list. When a player asks its action to the planner class, the planner class returns the defined action to the player. The player takes the abstract action, for example kick ball to player 3. It, then calculates the necessary moving direction and speed. If it is a kick ball action and the player has the ball then and the necessary ordering is acceptable then it kicks the ball.

Instead of detecting plan failures, we scheduled re-planning in every state change. We schedule re-planning when the planner *CanKick* variable changes or five pixel location change of players, opponents or ball. When re-planning is scheduled, the planning thread observes the environment and generates a problem file and asks the planner to generate a plan. Until a new plan is generated the planner stores and returns the previously defined actions of the players.

4 Experiments

The *PlannerTeam* played against three teams with different strategies. The first team is *AIKHomoG* which uses dynamic role assignment and potential fields for player movement. Most of the game period, two of the team players always stay in the two corners of the goal area and one player usually stays close to center. Generally, goals of *AIKHomoG* team are scored by our goalkeeper. The second team is *Kechze* which uses role assignment for strategy and geometric calculations as the assignment criteria [5]. Beside these techniques, *Kechze* team uses one of its players to block the opponents' goal keeper in left side of the goal and try to score that way. The third team is *FemmeBotsHeteroG* which defends its goal with two players.

4.1 Metrics

In order to compare the performance of the *PlannerTeam* with other teams we used five of the previously designed metrics [6] [7]. These metrics are; three possession metrics *BallPossessionHome*, *BallPossessionCenter*, *BallPossessionOpponent* measure the possession of the ball on the field during the game. *BallPossessionHome* and *BallPossessionOpponent* are the semicircle areas with radius of 9 times robot radius from the center of goals, the remaining area is called as *BallPossessionCenter* as shown in Figure 5. *Average of Our Scores* and *Average of Opponent* Scores are the remaining two metrics.

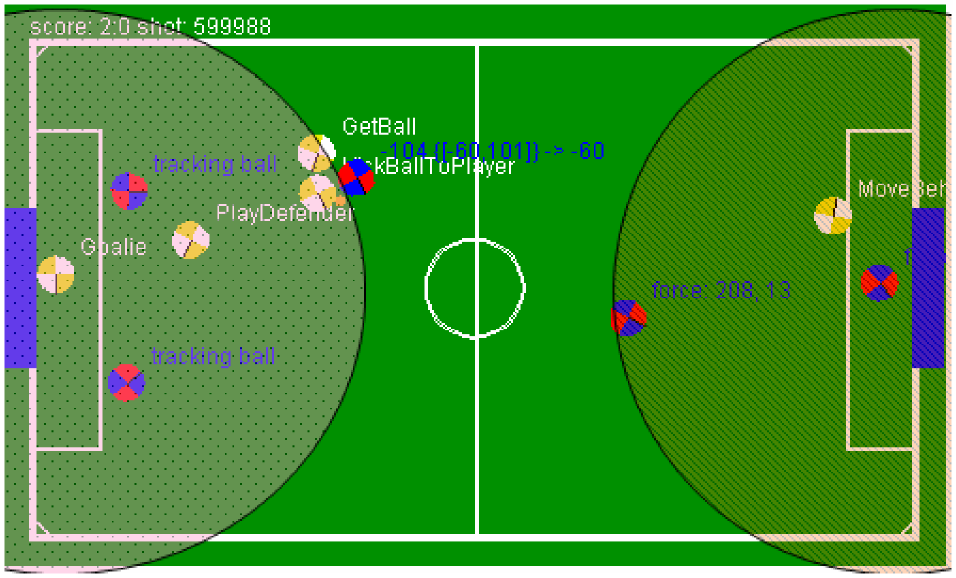


Figure 5. ***BallPossessionHome*****,** ***BallPossessionCenter*****,** ***BallPossessionOpponent*** **areas are shown. The** ***PlannerTeam*** **is playing against** ***AIKHomoG*****.**

5 Results

A total of 30 matches were played against the three opponent teams, 10 for each and the results are given in Table 1. We set the simulator *maxtimestep* parameter to 8 and timeout parameter to 240000 milliseconds. *Maxitimestep* statements set the maximum time (in milliseconds) that can transpire between discrete simulation steps. The games against *Kechze* and *FemmeBotsHeteroG* have better statistics than the game against *AIKHomoG*.

Table 1. Experiment Results

Opponent Team	Avg. Our Score	Avg. Opponent Score	Ball. Pos. Home.	Ball. Pos. Opp.	Ball. Pos. Center
AIKHomoG	0.4	0.5	0.352131	0.235115	0.412753
Kechze	0.9	0.2	0.126392	0.530549	0.343059
Femme	0.4	0.3	0.166722	0.394454	0.438825

6 Conclusions

In this work we integrated a general purpose planner with a multiagent simulation environment. Although our *PlannerTeam* is good at playing soccer we had many problems generating applicable plans. There are many reasons for that. First, the planner should return not only one plan but many plans. Secondly, in many cases a complete plan is not useful. Third the applicability of the plan is not considered by the planner. It always generates and returns the first plan it finds. On the other hand, this kind of agent planner integration is applicable to solve many problem types.

As a future work, we need to define a success ratio for each action and design a planner which considers the success ratio of those actions and generate all possible plans for a defined step length. Besides some state evaluation functions may be useful for positions in which a plan can not be generated. With this evaluation function the planner can chose the best possible state and return actions which leads us to that state. The planner domain and the goal definitions should be developed more.

References

1. A. Garvey and V. Lesser, *A Survey of Research in Deliberative Real-Time Artificial Intelligence,* UMass Computer Science Technical Report, 93–84 November 19, 1993.
2. S Russell and P Norvig. *Artificial Intelligence*: A Modern *Approach Second Edition.* Prentice-Hall International, 2003.
3. J. Hoffmann, "The FF Planning System: Fast Plan Generation Through Heuristic Search" *The Journal of Artificial Intelligence Research* 14, 253-302, 2001.
4. T. Wigel, J.-S., Gutman, M. Dietl, A. Kleiner, B. Nebel, "CS Freiburg: Coordinating Robots for Successful Soccer Playing", *Lecture Notes in Computer Science*, 2019, p.52, 2001.
5. T. Balch, "Teambots," http://www.teambots.org , 2003
6. K. Kaplan, "Design and Implementation of Fast Controllers for Mobile Robots," Master Thesis, Bogazici University, January 2003.
7. H. Kose, C. Mericli, K. Kaplan, and H. L. Akin, "All Bids for One and One Does for All: Market-Driven Multi-Agent Collaboration in Robot Soccer Domain*", ISCIS XVIII The Sixteenth International Symposium on Computer and Information Sciences* (ISCIS 2003), Springer-Verlag, Lecture Notes in Computer Science Series, 2003