A* Algorithm For Path Planning of Robotic Soccer Player

Tingsheng Liao Polat Utku Kayrak

Abstract—Path planning is one of the keys to win a robotic soccer game, and A* algorithm is one of the best approach for path planning. Most of existing works are using A* algorithm for static environment, but A* can also be applied to a dynamic system like a robotic soccer game. This paper investigates the feasibility of using A* algorithm to generate a best path for robotic soccer player to avoid it’s opponents in different scenarios. Our conclusion regarding the limit of performance in the extreme case to provide suggestions for robotic soccer path planning algorithm.

I. INTRODUCTION

In May, 1997. IBM Deep Blue defeated the human world champion in chess. Forty years of challenge in the AI community came to a successful conclusion. In the same year, NASAs pathfinder mission made a successful landing and the first autonomous robotics system, Sojourner, was deployed on the surface of Mars. Together with these accomplishments, RoboCup made its first steps toward the development of robotic soccer players which can beat a human World Cup champion team. [1]

The idea of robots playing soccer was first mentioned by Professor Alan Mackworth (University of British Columbia, Canada) in a paper entitled On Seeing Robots presented at VI-92, 1992. [2] A long term goal for robots is to develop multi-robot adaptive, co-operative, autonomous systems solving common tasks. Professor Mackworth chose soccer playing as one of the task in particular.

In this paper, we are only concerned with the path planning for avoiding robotic soccer player’s opponents. Our algorithm takes for granted that the position and goal of all opponents are correctly perceived. In the first part of this paper, an overview of A* algorithm is presented. We review the concepts of A* algorithm and propose our modification to allow A* algorithm to be used in dynamic environment. The second part of this paper present our simulation in different scenarios, and we discuss how does the performance of our algorithm varies with different scenarios. Finally, we give a briefly conclusion of our simulation.

II. REVIEW OF A* ALGORITHM

In 1968, Hart, Nilsson, and Raphael [3] proposed a proven optimization of Dijkstra’s algorithm dubbed $A^*$ (after multiple earlier version $A^1$, $A^2$, etc). $A^*$ is an informed search algorithm: it first searches the routes that appear to most likely lead to the destination. For this, $A^*$ uses a distance-plus-cost heuristic to determine the order in which to visit potential nodes on the route. $A^*$ greatly improves the time of Dijkstra’s algorithm and works well for path planning for robots navigating around fixed obstacles.

$A^*$ algorithm is one of the most effective algorithms to solve the path finding problem. It calculates the costs of each solution to find an optimal solution with minimal cost. The equation of total cost is

$$F = G + H$$

where $G$ is to the cost from the starting point to current (visited) one, and $H$ is the heuristic function estimating the cost from the current point to the destination. Therefore, the design of $G$ and $H$ will determine the total cost $F$ which can greatly affect the path $A^*$ finds.
The principle of the $A^*$ algorithm is to expand each possible node from start to destination and compare the cost of each path. Once there is an obstacle that makes the current path cost more than other available path, $A^*$ will go back to the new lowest path and expands it again. After repeating this finding process, $A^*$ will finally find a path that cost least once it expands to the destination. As shown in figure 1, the $A^*$ algorithm starts expanding nodes from $A$ to the goal $B$, and the grids with cross marks are the obstacles. Each expanded grid contains the information: $F$ at the top left, $G$ at the bottom left, $H$ at the bottom right, and the arrow pointing its parent. In this example, the heuristic function is the Manhattan method:

The Manhattan method [4] calculates the total vertical and horizontal distance between the current cell and the destination, ignoring obstacles, discarding any diagonal movement.

![Figure 1. Illustration of $A^*$ path finding algorithm. [5]](image)

<table>
<thead>
<tr>
<th>A</th>
<th>1.0</th>
<th>2.0</th>
<th>3.0</th>
<th>4.0</th>
<th>5.0</th>
<th>6.0</th>
<th>7.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2.0</td>
<td>3.0</td>
<td>4.0</td>
<td>5.0</td>
<td>6.0</td>
<td>7.0</td>
<td>8.0</td>
<td>9.0</td>
</tr>
<tr>
<td>2.0</td>
<td>3.0</td>
<td>5.0</td>
<td>6.0</td>
<td>7.0</td>
<td>8.0</td>
<td>9.0</td>
<td>10.0</td>
<td>11.0</td>
</tr>
<tr>
<td>3.0</td>
<td>4.0</td>
<td>6.0</td>
<td>8.0</td>
<td>10.0</td>
<td>12.0</td>
<td>14.0</td>
<td>16.0</td>
<td>18.0</td>
</tr>
<tr>
<td>4.0</td>
<td>5.0</td>
<td>7.0</td>
<td>9.0</td>
<td>11.0</td>
<td>13.0</td>
<td>15.0</td>
<td>17.0</td>
<td>19.0</td>
</tr>
<tr>
<td>5.0</td>
<td>6.0</td>
<td>8.0</td>
<td>10.0</td>
<td>12.0</td>
<td>14.0</td>
<td>16.0</td>
<td>18.0</td>
<td>20.0</td>
</tr>
<tr>
<td>6.0</td>
<td>7.0</td>
<td>9.0</td>
<td>11.0</td>
<td>13.0</td>
<td>15.0</td>
<td>17.0</td>
<td>19.0</td>
<td>21.0</td>
</tr>
<tr>
<td>7.0</td>
<td>9.0</td>
<td>10.0</td>
<td>11.0</td>
<td>12.0</td>
<td>13.0</td>
<td>14.0</td>
<td>15.0</td>
<td>16.0</td>
</tr>
<tr>
<td>8.0</td>
<td>9.0</td>
<td>10.0</td>
<td>11.0</td>
<td>12.0</td>
<td>13.0</td>
<td>14.0</td>
<td>15.0</td>
<td>16.0</td>
</tr>
</tbody>
</table>

(a)

![Figure 2. (a) Path finding using classic $A^*$ method (b) Path finding using fudge method [6]](image)

(b)

Obviously, the heuristic function $H$ is critical to determine the performance of the $A^*$ algorithm. By using different heuristic functions, the result can be completely different. Figure 2 shows how the paths differ by using two different heuristic functions.

### III. IMPLEMENTATION

The traditional $A^*$ algorithm mostly works in static environment, but it is not necessary. We can predict the future position based on their coordinates and bearing as reported by the sensor onboard. By designing a heuristic function that can describe a dynamic system like robotic soccer game, $A^*$ algorithm is capable to do the path planning to avoid moving opponents.

Figure 3 illustrates the concept of using $A^*$ algorithm to deal with dynamic environment. Instead of using traditional map, we use a map that varies with the increasing time step so that $A^*$ algorithm can avoid the potential encounters based on the prediction.
In order to calculate the cost function in a changing map, we introduce an additional equation to describe the costs generated by other opponents.

\[ H = \sum_{i=1}^{n} C - (|x_n - x_0| + |y_n - y_0|) \]

In this equation, C is a constant that usually chosen by the size of map. Therefore, when our players are getting closer to other opponents, the cost of heuristic function H becomes larger.

IV. SIMULATION

In our simulation, we consider three different scenarios to see how A* algorithm perform in these situations. In each scenario, there are five opponents to try to catch our player. Only our player is equipped the modified version of A* algorithm to allow it to find a best path to avoid it’s opponents and reach the destination, other opponents are just equipped the general A* algorithm guiding them from one position to another.

A. Scenario: Surrounding Opponents

The figure 4 shows that in this scenario, our player is surrounded by the opponents from four direction, and it has to avoid them to reach the destination on the top. For the opponents, they are moving toward to our player and try to catch it.

The simulation result is shown in figure 5, and our player (blue star) just barely pass the opponents within one time step to be caught.

B. Scenario: Opponents in a row

In this scenario, the opponents start on the same row and move forward to cut our player’s path as shown in figure 6. Figure 7 shows that our player can easily avoid the opponents to reach the destination in the middle.
C. Scenario: Crowded Opponents

In this scenario, the opponents are more crowded during the movement because all of them will across the original path of our player. Figure 8 shows the initial position of all robots.

As a result, our player is more conservative in this scenario. As shown in figure 9, instead of passing through the opponents our player choose the path to keep a long distance from opponents.

V. Conclusion

Our simulation result shows that A* algorithm is capable to solve the path planning problem for robotic soccer. However, in some scenario like the crowded opponents, the path is not ideal. This is due to the heuristic function we use would cumulate the cost of each opponents. It means when the opponents are getting closer, the entire area around them are block because of the cumulative costs. To solve this problem, a new heuristic function is needed to compute the cost equation of each opponent independently.
REFERENCES


Polat Utku Kayrak was born in Istanbul, Turkey in 1987 and completed his undergraduate studies in Koc University, one of the best engineering schools in Istanbul. His graduation project was on "Developing Signal Processing Algorithms for Mathematical Finance". He has been accepted as an undergraduate research assistant at Yale University in 2010 and worked on "Bio-Inspired Synthetic Vision" in Prof. Eugenio Culurciello’s group. He’s currently pursuing his Master’s Degree in Auburn University in Electrical Computer Engineering Department.

Tingsheng Liao is a graduate student at Auburn University, and is presently majored in Electrical and Computer Engineering for M.S. degree. He have done research in the past on Loci Control of Cs-113 Robotic Arm for Plane Motion. His chief research interests lie in the field of robotics and control system, and he is working on UAV collision avoidance as a research assistant.