Path finding algorithm for moving robots and obstacles avoidance

Steven Boeckx
Thomas More campus De Nayer
Sint Katelijne Waver, Belgium
steven.boeckx1@gmail.com

Patrick Pelgrims
Thomas More campus De Nayer
Sint Katelijne Waver, Belgium
patrick.pelgrims@thomasmore.be

Anzhelika Parkhomenko
Software Tools Department
Zaporizhzhya National Technical University
Zaporizhzhya, Ukraine
parhom@zntu.edu.ua

Dirk Van Merode
Thomas More campus De Nayer
Sint Katelijne Waver, Belgium
dirk.vanmerode@thomasmore.be

Olga Gladkova
Software Tools Department
Zaporizhzhya National Technical University
Zaporizhzhya, Ukraine
gladolechka@gmail.com

***Abstract*— In this paper existing path finding algorithms were analized and the best path finding algorithm for the Robocup project is presented.**

Keywords—path finding algorithm; mobile object; obstacles

I. INTRODUCTION

Robocup is a competition in robot soccer matches. The final goal is to form a team of humanoid robots that can win a match against the human soccer world champions by 2050 [1]. But it is not just a competition, it also has the goal to promote robotics and artificial intelligence (AI) research.

Each participating team builds its own robots. As part of a student project in Thomas More Campus De Nayer (Sint Katelijne Waver, Belgium) the Robocup Project was started. In the Small Size League the robots need to have a cylinder-like shape. The maximum size of the cylinder is 150mm in height and 180mm in diameter. The match is held between two teams of 6 soccer robots each. Building a successful team requires clever design, implementation and integration of many hardware and software sub-components into a robustly functioning whole [2].

The system architecture is shown on Figure 1. These robots are autonomously controlled by top-down oriented cameras and communicate wirelessly to each other to defeat their opponents.

All robots on the field and the ball are tracked by a standardized vision system – called SSL-Vision - that processes the data provided by two cameras that are attached to a camera bar located 4 m above the playing surface.

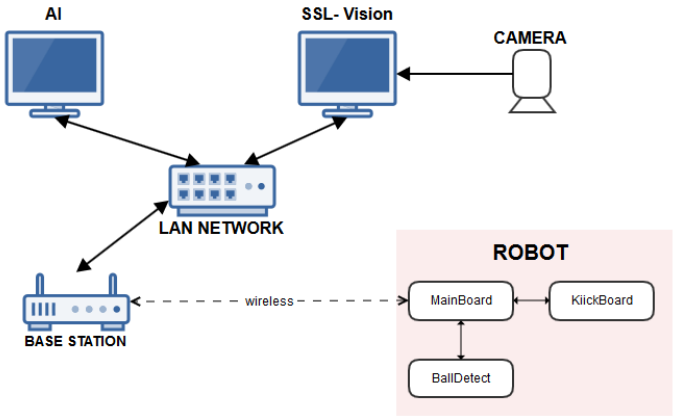

Fig. 1. The system architecture

The robots are tracked using a custom pattern on top, which encodes the team color (circle in the middle) and a unique robot id – the four circles around (Fig.2).[3]. The standard marker papers are: Pink, Green, Blue, Yellow.

Fig. 2. Robot small sized league

The camera data is processed by a central computer. This position data of both teams is broadcasted into a network. Each team has a central computer, which receives the data, processes it and sends commands to the robots with a technique of their choice. [4]

O,e of the biggest problems in Robocup is to calculate the path between two points on the field without crashing into other obstacles/robots. Not only preventing to hit the obstacles

is a big task, but also doing this in a limited time frame so that the AI has enough time to calculate other tasks like role assignment, sending data over the network, and so on.

II. Investigation of the Path Finding Algorithms

Research has shown that path finding algorithm can be divided in grid based algorithms and graph based algorithms.

A. *Grid based algorithm*

The first path algorithm is grid based. The idea is to work with a square grid and to make a list of nodes. Like the name suggest you divide the field in even squares. Every square is defined with a specific state of that place on the field, for example: empty, obstacle, height differences, etc.

The advantage of this type of algorithm is that there is no need for intensive calculation. But this does not provide any benefit in this case, because we need to have a big resolution to cover the field. For example, if we want to use this type of algorithm for Robocup we need to make the following calculation if every square is 0.5cm: the playfield is 6m x 3m. If we calculate the total of needed squares then we need 12000 x 6000 x 1bit (defining if there is an object yes or no) = 72 000 000 bits or 9Mbytes of information. This is too much therefor the use of this type of algorithm was discarded.

B. *Graph based algorithm*

Graph based algorithm uses math to calculate a path. This has an advantage and a disadvantage. The advantage is that there is no need for a big memory map to save all the properties (obstructions, empty, …). The disadvantage of this algorithm is that it needs more processing power to do the calculations.

The graph-based algorithm was chosen in this case, as the algorithm will be calculated on a PC, so the processing power is not a big issue.

III. Path Finding Algorithm

A. *Path finding algorithm*

To begin with 2 classes were made: node and tree. An object tree exists of many nodes. The number of nodes is undefined; this is just how many the algorithm needs. The maximum amount of nodes that can be used is predefined, just in case the algorithm did not reach the goal after a long time. In case this happens the algorithm starts over again.

The algorithm starts with asking in which direction it needs to grow, in the direction of the goal or in the direction of a random place. This will be chosen with a random number that can be between 0-100. The random number will then be compared with the ratio (like discussed in “RRT, rapidly-exploring random tree”). Those two elements will define what the target is.

For example: if the ratio is random 30% and goal 70%, then the code will select the target being random, if the generated number is smaller or equal to 30. If the generated number is greater than 30, the target will be the goal.

After that the code will look which node is the closest to the target that is defined in the previous step. This is simply done with the Pythagorean Theorem.

Then the tree will be “virtually” extended towards the target. The starting place of extension is from the node that is the nearest to the target.

After the new node is calculated, the algorithm checks if there is a collision with objects on the field. In case that there is no collision, the node will be added to the tree. In the other case the node will be discarded.

The last step is to check if the goal is reached. If yes, the algorithm ends, else it just starts again from the beginning until the goal is reached.

B. *Testing*

The tree is displayed in the developed application by black lines. The nodes are the circles.

After the goal is reached you can easily determine how the tree reached the goal. This can be done just by asking the last node (the one that reached the goal), what its parent is. So from which node it extended from. If you then ask the same at the parent from the node and then at the parent’s parent from the node and repeat this until the starting point is reached, you will get the path. This the red line in the application.

For control of the defining factors, like the ratio, the growth speed and the distance between the nodes, sliders were added to easily modify them (Fig.3).

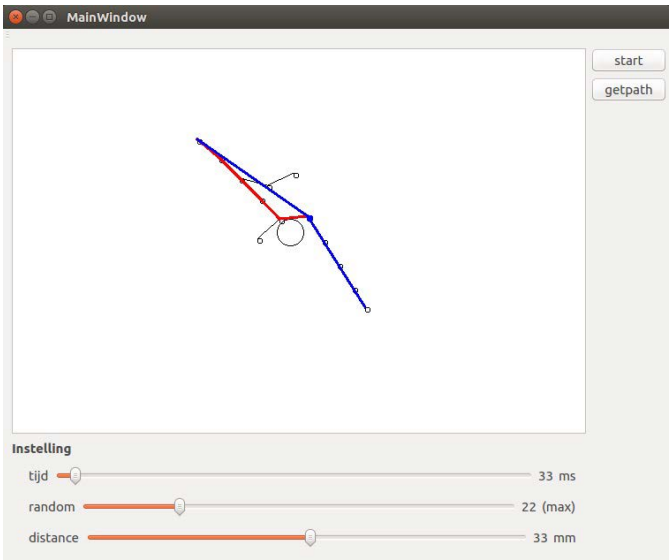

Fig. 3. Screenshot path finding test application

The blue line in the application (Fig.3) is determined by the smoothing algorithm. The path needs to be smooth out because the generated path has unnecessary corners that only extend the total length of the path.

C. *Smoothing algorithm*

The algorithm starts by defining 3 nodes: starting node, current node and previous node (Fig.4). The starting node and

the previous node are the last node in the path (the one nearest to the goal).

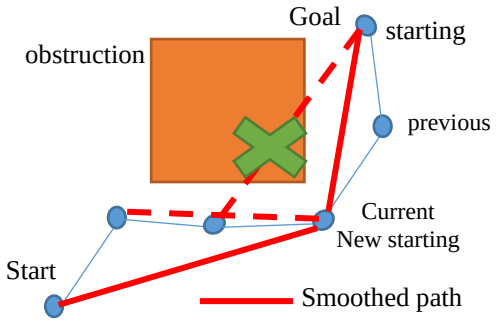

Fig.4. Smoothing algorithm operation diagram

Then the previous node is equal to the current node. After that the parent off the current node is the current node. Then checking on collision is done between the starting node and the current node. In case there is, the previous node will be the new starting node, this node will also be added to the "smoothed path".

This will be done until the start point has been reached.

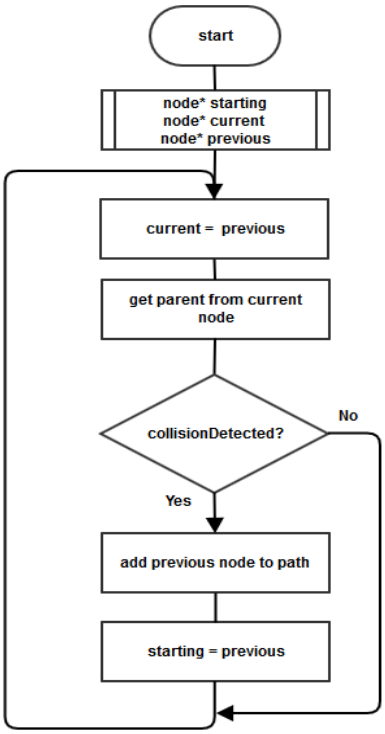

Fig.5. Smoothing algorithm flowchart

D. Final application

Finally the described path finding algorithm was tested in a real situation. To do this, an additional application in QT was developed which reads the coordinates from the robots and the ball by the use of UDP (Fig.6). The ball in this application is the goal and the coordinates of the robot itself is the starting point. The other robots on the field are the obstructions what the robot needs drive around for reaching the goal/ball.

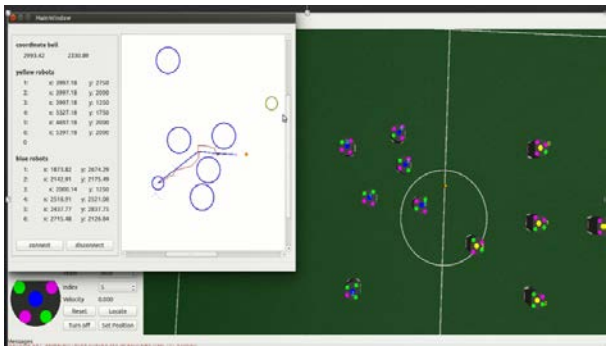

Fig. 6. Screenshot AI application in QT and GrSim

Similar as in the first application we can recognize the black, red and blue line defining the steps from the path finding algorithm. We can see that the robots are on the same place in the application as in the simulator (only horizontal mirrored). The orange dot is the ball, the blue circles are the first team and the green circles the second team.

IV. Conclusion

At this moment assembling of one robot is complete. The QT framework was used for software realization of the path finding algorithm and algorithms of obstacles bypass. The GrSim simulator [5] was used for testing. Results of the simulation showed that the system works correctly. For now it only operates in simple conditions, with the other robots stationary. The work for more complex situation with a lot of moving objects is being continued. The next stage is testing the system on a real life field together with other robots.

Acknowledgements

We would like to express our gratitude towards Patrick Pelgrims, Jurre de Weerdt and Wim Dams, of the EmSys Research Group, for research and manufacturing support, towards Thomas More Mechelen-Antwerpen for enabling us to do this research and towards VDAB Haasrode for the manufacturing of all the mechanical parts.

References

[1] Daniel Waigand and Gunther Berthold, "Cooperative shoot and pass behavior of mobile robots in the context of the TIGERS-Mannheim SSL Robocup-project". 23.12.2010. URL: https://tigers-mannheim.de/download/papers/2011-Pass-and-shoot-Weigand_Berthold.pdf (visited on 26.08.2017)

[2] Small Size League. URL: http://wiki.robocup.org/wiki/Small_Size_League (visited on 26.08.2017)

[3] RoboCup SSL-Vision Wiki. URL:https://github.com/RoboCup-SSL/ssl-vision/wiki (visited on 26.08.2017)

[4] Nicolai Ommer AI Architecture and Standard Game Strategies in RoboCup SSL. URL: https://tigers-mannheim.de/download/papers/2013-AI_Architecture_and_Strategies-Ommer.pdf (visited on 26.08.2017)

GrSim – Overview [Online]. Available: https://github.com/mani-monaj/grSim/blob/master/INSTALL.md