Robotics and Autonomous Systems 61 (2013) 1440–1449

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Path planning with obstacle avoidance based on visibility binary tree algorithm

Abdulmuttalib Turky Rashid [a,*], Abduladhem Abdulkareem Ali [b], Mattia Frasca [c], Luigi Fortuna [c]

[a] *Electrical Engineering Department, University of Basrah, Basrah, Iraq*
[b] *Computer Engineering Department, University of Basrah, Basrah, Iraq*
[c] *DIEEI, Faculty of Engineering, University of Catania, Catania, Italy*

HIGHLIGHTS

- A new algorithm for robot navigation, referred to as visibility binary tree algorithm is introduced.
- The construction of this algorithm is based on the visible tangents between robot and obstacles.
- The shortest path is run on top of the visibility binary tree.
- The performance is compared with three different algorithms for path planning.

ARTICLE INFO

Article history:
Received 19 November 2012
Received in revised form
5 July 2013
Accepted 15 July 2013
Available online 23 July 2013

Keywords:
Path planning
Obstacle avoidance
Visibility graph
Bresenham algorithm

ABSTRACT

In this paper, a novel method for robot navigation in dynamic environments, referred to as *visibility binary tree algorithm*, is introduced. To plan the path of the robot, the algorithm relies on the construction of the set of all complete paths between robot and target taking into account inner and outer visible tangents between robot and circular obstacles. The paths are then used to create a visibility binary tree on top of which an algorithm for shortest path is run. The proposed algorithm is implemented on two simulation scenarios, one of them involving global knowledge of the environment, and the other based on local knowledge of the environment. The performance are compared with three different algorithms for path planning.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Path planning and obstacle avoidance are two important aspects of autonomous mobile robot navigation. Based on the sensor information available, the approaches to path planning can be classified into global and local methods [1,2]. In global methods, the robot plans its trajectory on the basis of a global information on the environment [3]. This approach guarantees the convergence of the robot path to the target, and also indicates if the goal is reachable or unreachable. On the other side, planning approaches based on sensors providing limited (local) information, although of simpler implementation, do not guarantee the global convergence to the target [4], since the robot uses its sensors to locate nearby obstacles at each control cycle and to plan the next action to be executed.

Many techniques for path planning in the presence of obstacles employ a Voronoi diagram under the hypothesis either of a global [5] or a local knowledge scenario [6]. In the global knowledge scenario the assumption is that each robot has complete information about all obstacles in the environment, while in the local knowledge scenario the information of each robot is limited to its sensing range. In approaches based on the Voronoi diagram, the planned trajectory is either a piecewise linear trajectory or a smooth path. In the first case, the robots have to stop at each of the trajectory segments end, change its orientation according to the next segment and then restart again. This kind of motion is disagreeable and leads to additional waste of power. In order to get a smooth path, the use of different curves instead of linear segments has been proposed. An example is the use of Voronoi diagram for smooth path planning and better obstacle avoidance by iterative enhancement proposed in [7]. Another example is the use of Bezier curves. Smooths paths that are reliable with robot dynamics are generated by using Bezier curves of degree three [8]. Time optimality [9], navigation in presence of corridor constraints [10] and

* Corresponding author. Tel.: +964 7806325288.
E-mail addresses: muttalib_63@yahoo.com, abdturky@gmail.com (A.T. Rashid), Abduladem1@yahoo.com (A.A. Ali), mfrasca@diees.unict.it (M. Frasca).

0921-8890/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.robot.2013.07.010

curvature control [11,12] are other issues of navigation control addressed with approaches based on the use of Bezier curves. These approaches produce smooth paths but often require longer trajectories to the target.

Another class of methods for path planning is based on the use of graph search algorithms for shortest path. In such approaches, first the graph of possible paths is built and then a shortest path search algorithm is applied on such graph. The visibility graph can be used for this purpose: this is a graph of visible obstacle vertices (obstacles are assumed to be polygonal), where a vertex V is defined as visible from a vertex U if the segment VU does not intersect any obstacle edges in the environment [13,14]. A general limitation of methods based on graph is the computation time for shortest path calculation. For this reason, a simplification of the graph structure to be explored may result in an improvement of the efficiency of these methods. This is the idea underlying the introduction of other graphs instead of the visibility graph. Several works use, for instance, the so-called tangent visibility graph. The tangent visibility graph is defined as the set of possible trajectories obtained from visibility graph by retaining only the edges which are bi-tangent to convex obstacle vertices [15,16]. The tangent visibility graph has a lower number of vertices and edges than the visibility graph. This leads to a more efficient process of shortest path calculation for the graph. As concerns the algorithms for finding the shortest-path the Dijkstra's algorithm is used on a full visibility graph in [17]. The search can be optimized by running it on the tangent graph as in [18,19], instead on the visibility graph.

The aim of this paper is to introduce a new algorithm for path planning based on a further simplification of the graph structure used. The resulting graph, which we name *the visibility binary tree*, is derived from the tangent visibility graph. We assume that the robot and the obstacles have circular shapes, that the robot radius is R and that the obstacle radius is the sum of the physical space occupied by the obstacle and the radius of the robot. The visibility binary tree is built starting from all possible paths between the robot position and the target and optimizing the structure by reducing redundant edges. After this step, an ad hoc searching algorithm is run on this graph. In fact, thanks to the simplified structure, the searching phase is also optimized. A further contribution of this work is the use of a Bresenham algorithm for low level trajectory planning. This has the advantage of requiring few computational resources, so that the whole algorithm introduced in this paper is aimed at reducing the computational resources required for its implementation.

The rest of the paper is organized as follows. Section 2 describes the low level of trajectory planning of the robot. Section 3 develops the theoretical analysis, describing the visibility binary tree algorithm. In Section 4, the visibility binary tree algorithm has been tested in two scenarios. Finally, Section 5 draws the conclusions of the paper.

2. Low level of trajectory planning

In this section, we briefly discuss the low level trajectory planning of the robot. The low level trajectory planning aims at implementing the routines needed for a robot to follow a given trajectory, whereas this trajectory is the result of the high level path planning. In particular, in this section we briefly discuss the kinematics of the robot and the Bresenham algorithm used to implement robot motion along a straight line or an arc line.

Although Bresenham algorithms [20] were developed for drawing lines and circles on a pixelated display systems such as VGAs [21], in our work we apply them to implement the low level of trajectory planning. In pixelated display systems a line is defined as a set of points (pixels in the screen), in our approach we represent the trajectory of robot to be computed as a set of successive positions in the plane (points in the plane). So, we establish an

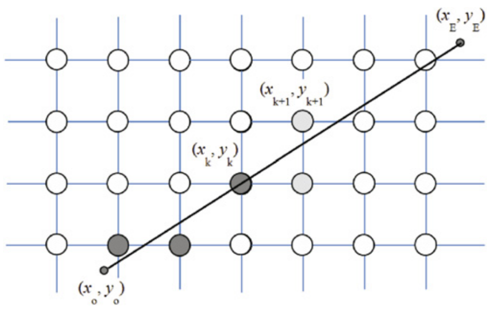

Fig. 1. Illustration of the Bresenham line algorithm.

analogy between pixels and points in the plane. To the best of our knowledge, this is the first time in which these algorithms are applied to trajectory planning. There are several useful characteristics in these algorithms motivating this choice: they are fast incremental algorithms and use only integer calculations; all multiplications are by 2 and thus can accomplished with a simple left shift instruction. These characteristics make these algorithms particularly suitable for implementation in hardware systems with limited available resources.

Let us first discuss the Bresenham line algorithm, which we use to implement planning of a straight line trajectory. Consider as in Fig. 1 a two-dimensional grid of points in the space where the robot moves. Assume that the robot initial position is (x_0, y_0), the direction to follow is given by the straight line shown in Fig. 1 and that the final end point is (x_E, y_E). The objective of the algorithm is to derive the sequence of positions in the grid in which the robot has to move to follow in an approximate way the depicted line. This is accomplished by moving at each step to the next position along the x axis (i.e., from x_k to x_{k+1}) and then by selecting which of y_k or y_{k+1} is the closest coordinate to the line (the points in the grid are indicated as (x_k, y_k) where k is an index labeling the points in the grid). Thus, the algorithm essentially has to choose at each step the value of the y coordinate. This is done by calculating at each time step a decision parameter p_k.

The algorithm can be described by the following steps:

1. start from the two line end points (x_0, y_0) and (x_E, y_E) and calculate the constants $\Delta x = x_E - x_0$ and $\Delta y = y_E - y_0$.
2. calculate the first value of the decision parameter as:
$$p_0 = 2\Delta y - \Delta x. \tag{1}$$
3. for each value of x_k along the line, perform the following test. If $p_k < 0$, the next point to be selected is (x_{k+1}, y_k) and:
$$p_{k+1} = p_k + 2\Delta y. \tag{2}$$
Otherwise, the next point to be selected is (x_{k+1}, y_{k+1}) and:
$$p_{k+1} = p_k + 2\Delta y - 2\Delta x. \tag{3}$$
4. repeat step 4 until (x_E, y_E) is reached.

Let us now discuss the Bresenham circle algorithm we used to derive a circular trajectory of the robot. The algorithm assumes that the circle is centered at the origin, so that the circle has a eight-way symmetry and it suffices to calculate the locations of the robot in one of the octants. We will refer to the original coordinates as (X, Y) and to the coordinates in the reference frame where the circle is centered in the origin as (x, y). Analogously to the case of the Bresenham line algorithm, at each step the new position x_{k+1} is selected and the algorithm has to choice which coordinate y_k or y_{k-1} is closest to the circle boundary. This is done by testing if the mid point between y_k and y_{k-1} is inside the circle or not, i.e., if $f(x, y) \triangleq x^2 + y^2 - r^2$ calculated at the mid point is negative or positive. This idea can be accomplished in an incremental way by the following steps (see Fig. 2):

Download English Version:

https://daneshyari.com/en/article/411342

Download Persian Version:

https://daneshyari.com/article/411342

Daneshyari.com