Robot Path Planning

Christiane Cancado, Jason Zadwick, Allen Rush

Santa Clara University

June 9, 2015

Table of Contents

Figures and Tables

Abstract

The process of Robot path planning has been studied for many years, and several generations of algorithms have been proposed and explored. The basic idea is to find a most efficient path through a maze, consisting of a map of obstacles. The map of obstacles is assumed to have a reachable solution; in other words, there exists at least one valid solution to the problem. The baseline is a method from the paper "An Efficient Dynamic System for Real Time Robot-Path Planning", by Willms and Yang. It is a discrete distance propagating planning algorithm.

This project explores the features and capabilities of the distance propagating algorithm, and describes the limitations of it. By extending the algorithm to cover some of the limitations, an improved version is described. The new version is called "Check and Reset" and extends the principles of the distance propagating algorithm to account for the possible collisions that can be encountered. The new algorithm is evaluated for completeness and some boundary conditions, as well as compared to the original distance propagating algorithm. It is expected that the improvements will be in time and/or distance relative to the original algorithm.

Introduction

Objective

The objective of this project is to explore improvements to the robot path planning algorithm described in the paper "An Efficient Dynamic System for Real Time Robot-Path Planning" by Willms and Yang. Robot path planning is an important tool for determining efficient and error (collision minimization) solutions for self-guided robots. The basic idea of this algorithm is to optimize a path based on distance propagating the path. In this example, the robot is simulated in a rectangular maze, but the solution can be extended to virtually any similar type of path (2D).

The new algorithm, called "Check and Reset", is designed to overcome the limitations of the distance propagating algorithm by applying probabilities and heuristic for executing an improved avoidance algorithm.

Problem

The problem that is addressed in robot path planning is to find a reasonable path through a maze consisting of a collection of obstacles. The best path is one that a) avoids all obstacles and b) executes in minimum time (or distance, cost, etc.). In the static case, this can be done by globally understanding the landscape and constructing a graph that can be easily traversed to find an optimal solution. However, in the (more typical) case of moving obstacles, one must be prepared to encounter obstacles at positions that were not known at the time of start. In this case, some update decisions need to be made regarding the future path, once a collision is made. This is collectively known as dynamic programming, implying that the decisions for the future route are made dynamically, based on current observations or conditions.

Theoretical Bases and Literature Review

Robot planning is well documented going back to the 1970s, and continues with even more recent work in the last few years. There are many different approaches to be used and [4] explores how much and when they have been used. Comparing BUG algorithms, Voronoi diagram and Silhouette as the most famous used in the 80's until the majority use of Fuzzy as a first choice in the last decade. [3] on the other hand, exposes the strength and weakness of the different methods with specific configurations. Among the methods revised and compared, Dynamic path planning always adds complexity issues, making the solution harder to be found.

The main paper that is used for this study is “An Efficient Dynamic System for Real Time Robot-Path Planning”, by Willms and Yang. It describes a bottom-up dynamic programming algorithm for determining a path through the maze that is based on calculating the distance to the target by propagating distance information through the free workspace by means of a discrete time evolution. This method has the advantage that once the dynamic system equilibrates the path chosen by the robot is always the shortest path. If the goal is for the robot to take a shortest path motion, then the distance propagating method outperforms neural nets which only propagate excitatory information and thus runs into degenerate type cases which the shortest path is through a narrow corridor in which the obstacles give a lot of inhibitory signals; the robot may then take a longer but more active path in the free workspace.

To implement the distance propagating algorithm, a distance accumulation is assigned to each possible path, and obstacles are penalized the heaviest in terms of potential distance. Once the best path is picked, the robot can proceed along the prescribed path. If, in the process it experiences a collision, it backs up to the most previous successful point and recalculates the next best path from that point. However, no penalty is assessed for actually colliding with the

obstacle. In this case, the desired improvement is to determine an update strategy that eliminates (or reduces to a small enough probability) that a collision will actually occur. In Figure 2, the best path is shown along with the static, or original position of the obstacle. If the obstacle moves, as shown in Figure 3, then the robot must recalculate the best path. In a worst-case scenario, it may need to re-trace its steps, potentially all the way to the beginning, if the obstacle movement changes completely the best path route.

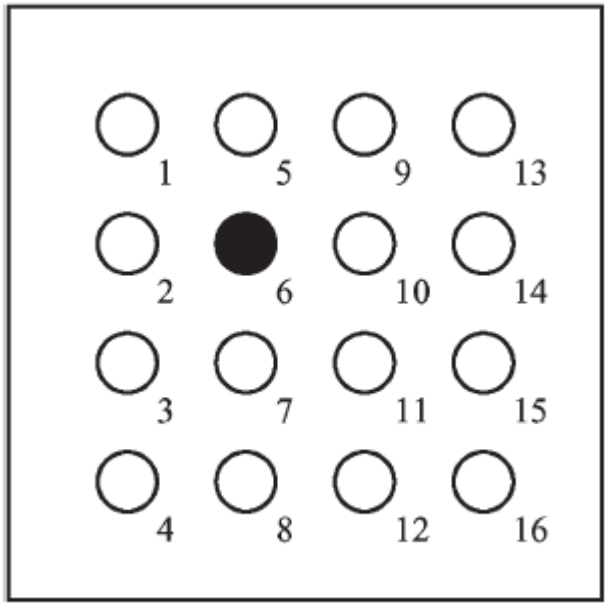

Figure 1 Grid with Obstacle at location 6

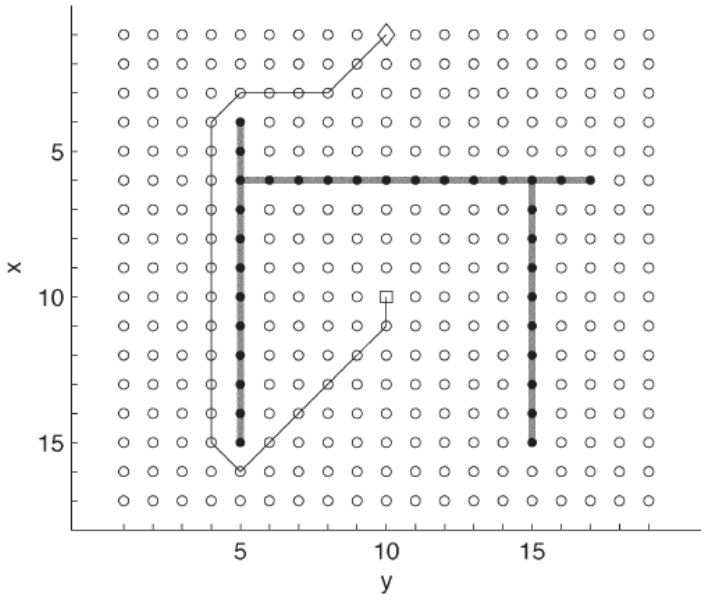

Figure 2 Robot path with obstacle

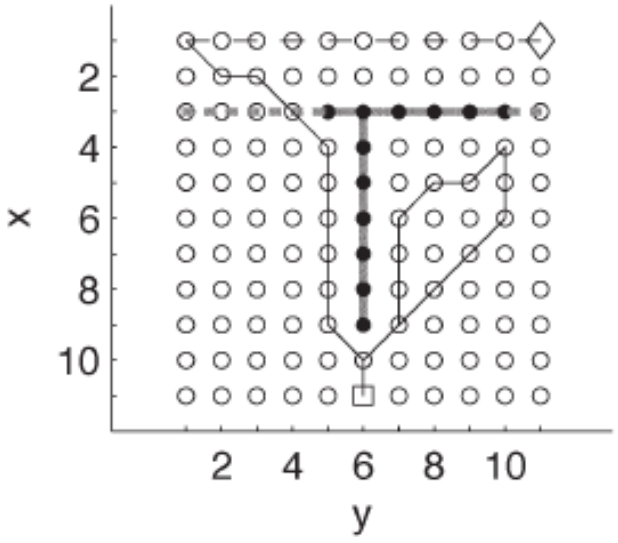

Figure 3 Map with Obstacles that have moved relative to original position

Previous work to find a solution to this problem has been summarized in many papers, and a selection is summarized in this report. In [2], the method of path planning based on triangulation is derived, starting with Voronoi diagrams. There are four traditional methods for creating path plans: potential field, which from [7] a series of equations describe the operational

space in terms of attraction (target) and repulsion (obstacles) forces. The idea of repulsive force potential provides a basis for creating a path through the maze that avoids obstacles by observing their extended force field and planning to either steer around or encounter the edge and change course. In either case, the obstacle is avoided in terms of collision. The second method is reverse gradient which converges to the shortest path. The third is based on a path through an annular outline of the free space between obstacles. The final method is configuration space, which is the basis for this study. Configuration space tries to find a path that is furthest from the obstacles. The most reliable and conservative method that implements this strategy is the Voronoi diagram. While this provides a good path that avoids obstacles, it has two drawbacks that make it less than ideal: the static Voronoi calculation has a probability of failing as obstacles move (the original points are no longer valid), and the size of the obstacle is not considered when calculating the Voronoi diagram.

The proposal in [1] is to use Delauney triangulation as a way to reduce the inefficiencies of the Voronoi diagram. DT is derived from the original Voronoi diagram as shown in figure 5. In addition, values are assigned to the nodes such that an efficient path can be found that overcomes the previously described limitations. In Figure 6, the path is shown with value assignments represented by different size nodes.

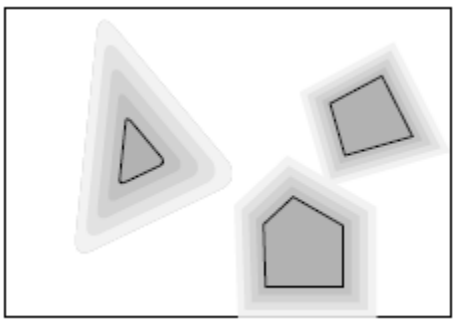

Figure 4 Potential fields of obstacles. Note density decreases further from center

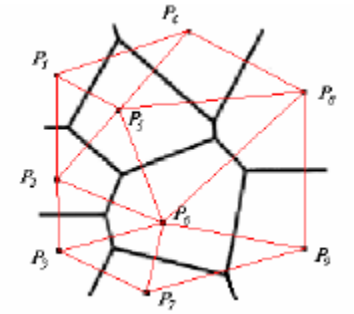

Figure 5 Delauney Triangulation derived from Voronoi Diagram

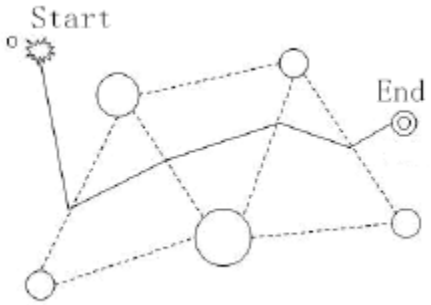

Figure 6 Path assignment based on Triangulation with value assignment

Hypothesis

When we consider the situation where obstacles are allowed to move, the triangulation method needs to be re-visited at points along the prescribed path due to the probability that it can be on a collision path with the previously cleared obstacle. The hypothesis for the improved path planning method is based on the idea that improvements in accuracy can be achieved by resetting the probabilities to new initial conditions such that with probability 1, a new path can be calculated without a possibility of collision. This is similar to dead reckoning with waypoint GPS updates. If you know a path with certainty (no obstacles) you have a high confidence that you will continue with no collisions. However, over time, the confidence will decrease, as obstacles

increase in probability that they can intercept the chosen path. Using probabilities and heuristics, it is possible to choose a “check and reset” waypoint that allows you to re-establish a new best path based on updates that provide current obstacle positions.

Path Planning

The process of path planning is based on finding a connection between a collection of vertices (representing the vertices of the obstacles). The method chosen is Dijkstra’s method. The input graph G contains the vertices of the obstacles and a starting point vertex. Figure 7 shows the pseudo-code for determining the path.

Input: Graph $G = (V, E)$
1 $(\forall x \neq s) dist[x] = +\infty$ //Initialize dist[]
2 $dist[s] = 0$
3 $S = \emptyset$
4 $Q = V$ // Keyed by $dist[]$.
5 **while** $Q \neq \emptyset$ **do**
6 $u = extract\_min(Q)$
7 $S = S \cup \{u\}$
8 **foreach** *vertex* $v \in Adj(u)$ **do**
9 $dist[v] = min(dist[v], dist[u] + w(u, v))$
10 //”*Relax*” operation.

Figure 7 Dijkstra's algorithm for finding the path from start to target

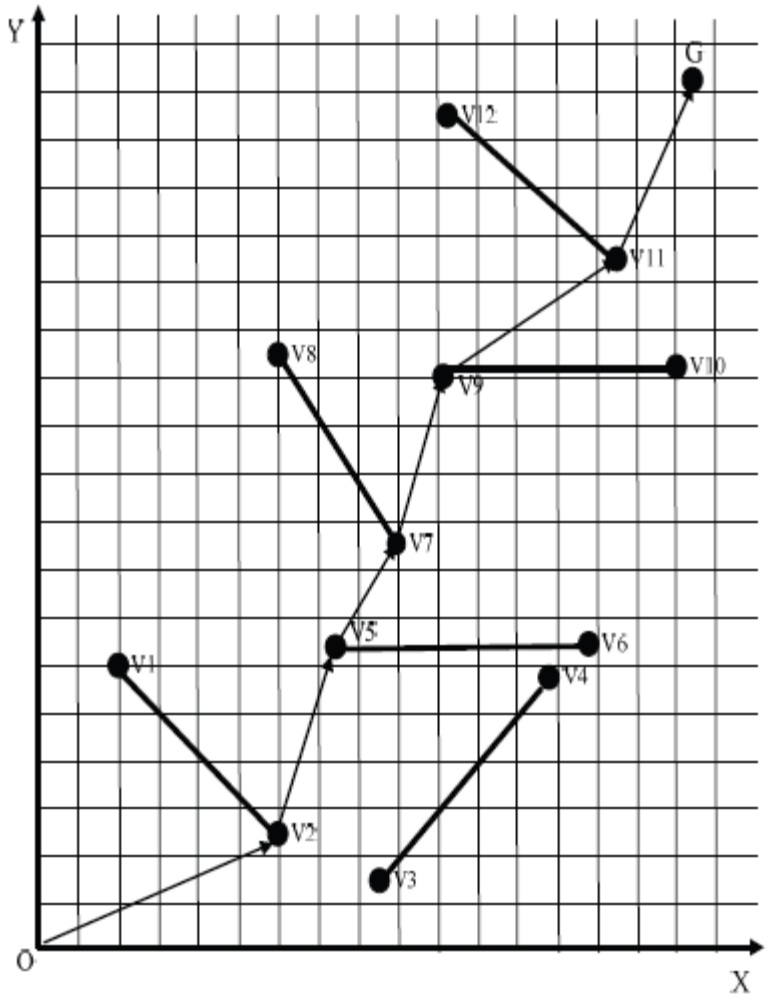

Figure 8 Path resulting from Dijkstra's method

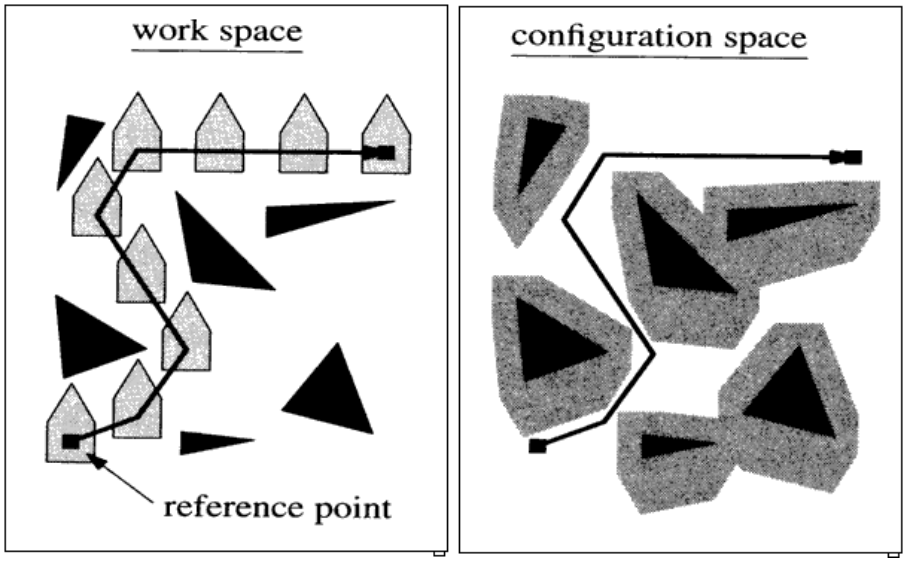

Figure 9 Map with Obstacles growing with probability of position uncertainty

Figure 9 shows the effect of an obstacle "growing" over time. At t0, there is an efficient path from start to finish, with obstacles identified and located. The configuration space will tend to constrict the available path more and more as time passes. However, the robot is also making progress along its best path solution. So, at some point in time say t0+k, the robot determines that it is close to being intercepted with a pre-determined probability based on the original position of the robot and the maximum possible velocity. At this point, the robot can stop and reset the map, taking into account the updated obstacle locations. At this point, it can be considered a new starting point, and a new best path can be calculated as before. The robot continues along the new path, repeating the process until it reaches the goal

In principle, this will be an improvement over the distance propagating, as it avoids altogether the possibility (within probability thresholds) of encountering a collision.

There are several options and extensions that can be considered to either improve the basic approach or add more interesting challenges. For example, one can consider multiple robots using the identical algorithm and communicating with each other about the update state that was reached based on the check and reset operation. In this way, all the robots can decide what, if any, updates they may need to make to their respective paths.

Another option is to consider the variable of speed between the obstacles and the robot. In the model, we consider only the narrow constraint that the obstacles are moving with a maximum speed Vo, and the robots are moving at constant speed Vr, which is greater that max(Vo). In the more relaxed case, the robot can have variable speed, and it may be greater or less than the Vo.

The last option is to use a Kalman filter to make course adjustments and predictions of obstacle path intercept. In this way, the KF predictor acts like an early warning system to the robot so a more effective decision can be made on when to execute the check and reset.

Methodology

The approach for modeling the proposed improved algorithm is based on a simulation of a robot traveling through a grid, populated with randomly placed obstacles. The robot has a start point (x,y) and a target point (x,y). After the grid is populated with obstacles, a best path is found by attempting to follow a straight line from start to target.

The simulation is visualized by using the OpenGL API. It contains many useful functions and a framework for displaying graphics on the screen, and easily supports screen updates as a function of object motion, as is required by the robot simulation.

Robot motion is managed by two matrices – FuturePath, and CurrentPath. The FuturePath matrix contains the grid point of the planned path, and the CurrentPath contains grid points of the path already travelled. The loop updates these matrices with an appropriate delay so the progress is visible (slow).

Obstacles are managed by two matrices – ObstMat, and ObstMatMove. For ObstMat, it is initialized with a fixed number (50 in the simulation) of randomly placed obstacles. At update time, the grid points are expanded function grow_obst(num). This simulates the probability that the robot has moved anywhere in a prescribed but limited range. So, for example, a grid starting with {1} at some point (x,y) will grow to {1,1,1;1,1,1;1,1,1} after one call to grow_obst, and so forth. After some time, this range will have a very high probability of intercepting the path prescribed by FuturePath. The ObstMatMove contains the actual movement of the obstacles. It is

updated every cycle by moving a fixed distance in a random direction (can be 0). In the worst case scenario, the movement will equal the range of the growth determined by grow_obst().

Once a potential collision is detected (function collision_prob(path_list), the obstacle grid is reset to the actual obstacle grid that has moved (ObstMatMoved). This also has the effect of resetting the probability range to 1. There are several helper functions and data structures to assist in this. The path is enumerated on a path_list structure- a linked list that takes the FuturePath grid and creates a list of every point on the path. The function collision_path(current_point) takes the pointer to the current position in the path_list) as the position of the robot on the path. If this position matches (collides with) any potential obstacle boundary, then the routine returns the number of the obstacle. A waypoint list is maintained to use as input into the function find_clear_path. When a collision (potential) occurs, the waypoint stack is updated by removing the top of stack and replacing it with the current position – this will become the new starting point. The find_clear_path will determine a new path to the target just as in the beginning. If a conflict occurs – the new path is already on a collision path – the routine will recursively call for a new path with additional waypoints determined by finding nearby obstacles and splitting the difference in distance. This is equivalent to using a modified Delaunay triangulation, by using midpoints of a Voronoi graph. This process continues until the robot reaches its destination.

Implementation

The implementation of the new method is evaluated by creating a simulation of a simple grid with a point robot and point obstacles. The choice of point robot and obstacle is for

convenience; the theory of the proposed method can be easily extended to cover obstacles of various shapes as well as robots that are rigid bodies.

A grid of 500x500 is used to represent the path environment. Each point in the 2 dimensional grid is a potential path point or can be occupied by an obstacle. The robot moves along a path that consists of a vector of x,y coordinates.

Initially, the grid is empty. It is populated with a target, some obstacles, and the starting point for the robot. Once the initial path is established, the robot is allowed to follow the prescribed path. Since we are using a discrete grid, the actual path will be an estimation of the actual path along any of the diagonal path segments.

As the robot moves, the obstacles can also move, in a random direction. Over time, the obstacles can reach a point that intersects the prescribed path solution. As the robot encounters waypoints it can determine if the probability is high enough to warrant a check. Once decided, the check will re-establish the ground truth for the actual position of the updated obstacles. The updated obstacle grid may or may not force a re-calculation of a new path solution. This cycle continues until the robot reaches the target.

Data Analysis and Discussion

One of the most important results that can be evaluated from this method is the ability of the algorithm to produce a collision-free path through the maze. When there is a set of static obstacles this is trivial – the initial path is guaranteed to be collision-free. Under obstacle movement scenarios, the ability of the system to recover and continue is the metric that is evaluated.

There are several variables that affect the performance of the simulation. The natural influence is the number/density of obstacles. The number of 50 was a reasonable balance – any

fewer and the probability of collision was reduced to an insignificant level. Any more, and the collision rate was too high and the simulation time became too high to evaluate meaningful results.

The growth rate affects the time to collision. For a growth rate of 1 per cycle, starting with a size of 1, the average number of collision potential hits was 8-10 per simulation run. The theoretical best distance is 500*sqrt(2) or about 700 steps. Due to line fitting, this number is increased slightly. This suggests that the average hit distance is about 70-90 steps before the check is done and the updated grid is loaded.

Conclusion and Recommendations

The traditional path planning is considered using a configuration space calculation using Dijkstra's algorithm and a graph based on Voronoi and triangulation. This approach can find an optimal path from the start to the target. However, as obstacles move, the robot can collide with obstacles as they intersect the original path. In order to prevent mid-journey collisions, we propose a method called "check and reset" to update the best path when a potential collision based on intersection is determined. The algorithm for making the check decision is based on heuristics that take into account the probability that any obstacle can intersect the path. The assumption about how the obstacle can move is choice based – any number of different policies can be considered. For this project, we assumed a random movement that is updated at every step, so the potential for intersection is an expanding circle whose radius grows with each step.

When a check point is determined, the location of obstacles is updated, and a new path is calculated. The choice of how many obstacles are updated is a parameter (based on practical considerations, for example how far the robot can actually see or sense the locations of nearby obstacles). The resulting updated path can continue on with no collisions.

Compared to earlier methods, this approach can completely avoid a collision, at the cost of the check point update and the effort to re-calculate the new path.

Future Work

The dynamic aspect of obstacles and path planning is a useful area of study. There are many robot related applications which benefit from both accuracy and speed of path planning and execution. This work focused on the basic idea of improving path planning under controlled obstacle movement. However, there are many parameters that should be considered, including:

- Altering the speed of the robot. If it can speed up and slow down, the probability of intersection gets more complicated, but the robot may be able to avoid a check point (expensive).
- Obstacles that move at varying rate, and with non-point size.
- Limit the range of check and reset – only local obstacles can be "updated". This is a realistic condition if a waypoint check is made and the only obstacles that are visible are in line of sight or within reach of a robot sensor.
- Increase the penalty of a collision and cost for reset. This has the effect of increasing the effort to initially plan the path to include potential bypass or alternate detours in the event of expected collision.

There are many other possible areas for study, but the basic idea of improving path performance is leveraged from the approach of path intersection prediction or anticipation.

Bibliography

1. Wang, H., et al; **"Application of Dijkstra algorithm in Robot Path-Planning"**, IEEE Second International Conference on Mechanical and Control Engineering, 2011.
2. Gong, F. and Wang, X.; "**Robot Path-planning based on Triangulation Tracing**", IEEE International Symposium on Intelligent Information Technology Applications, 2008.
3. Sariff, N. and Buniyamin, N.; "**An Overview of Autonomous Mobile Robot Path Planning Algorithms**", IEEE 4th Student Conference on Research and Development, 2006.
4. Tang, S., et al; "**A Review on Robot Motion Planning Approaches**", Pertanika J. Science and Technology, 2012.
5. Willms, A. and Yang, S.; "**An Efficient Dynamic System for Real-Time Robot-Path Planning**", IEEE Transactions on Systems, Man and Cybernetics, 2006.
6. Dong, H. and Li, W.; "**The Path Planning for Mobile Robot Based on Voronoi Diagram**", 2010 Third International Conference on Intelligent Networks and Intelligent Systems, 2010.
7. Khatib, O.; "**Real-time obstacles avoidance for manipulators and mobile robot**", The International Journal Robotics Research, 1986.