A Mobile Robot Navigation Exploration Algorithm

Alexander Zelinsky
Department of Computer Science
University of Wollongong

Abstract

This paper will present an algorithm for path planning to a goal with a mobile robot in an unknown environment. The robot maps the environment only to the extent that is necessary to achieve the goal. Mapping is achieved using tactile sensing while the robot is executing a path to the specified goal. Paths are generated by treating unknown regions in the environment as free space. As obstacles are encountered en route to a goal, the model of the environment is updated and a new path to the goal is planned and executed. Initially the paths to the goal generated by this algorithm will be negotiable paths. However as the robot acquires more knowledge about the environment, the length of the planned paths will be optimised. The optimisation criteria can be modified to favour or avoid unexplored regions in the environment. The algorithm presented in this paper makes use of the quadtree data structure to model the environment and uses the distance transform methodology to generate paths for the robot to execute.

1. Introduction

The problem of finding optimal paths for robot manipulators and autonomous mobile robots through environments cluttered with obstacles has attracted much research interest. A great deal of this research has concentrated on situations where the environment in which the robot operates is completely known and supplied to the path planner [1] - [12]. For a robot navigating in a partially known or completely unknown environment these path planning techniques are not directly applicable or extendible. Difficulties arise in deciding how to treat the unexplored regions of the environment. A typical approach is to treat unexplored regions as obstacles, and only proceed into the unexplored regions if the goal lies there [14] - [16]. The path planner reported by Lumelsky *et. al.* [13] can function in unknown environments, without the need for a robot to build an environment map. However this method procludes finding the optimal paths to a goal.

One of the problems with mobile robots operating in partially known or completely unknown environments is the updating of the environment map with sensor data. Sensor data is prone to noise and inaccuracy. This introduces uncertainty into the environment map and makes the task of navigation more difficult. Similar difficulties are encountered when matching sensor data with a robot produced map of the area, when a location is revisited [17] - [19]. A way forward suggested by Brooks in [20] is either to engineer away sensor inaccuracies or develop robust navigation algorithms which handle uncertainty. To develop and implement a navigation system for a mobile robot with current technology sensors clearly requires sophisticated algorithms.

Research efforts into construction of a map of the environment with mobile robots have followed a variety of approaches [14] - [23]. These can be broadly classified into two groupings; network/graph models and grid-based models. Network/graph models are high level descriptions, and the structure of these models is dependent upon the nature and clutter of the environment. Typically such models represent the environment as a network of free space regions [15] [17] - [22] or as a graph of obstacle vertices [14]. The network/graph model environment mapping methods offer elegant solutions, however these methods require accurate sensor information, and are therefore difficult to implement in practise. Grid-based models impose a structure onto the environment without any regard to the nature and clutter of the environment [18] [23], thus the implementation of such environment mapping methods is easier than network/graph models.

The use of distance transforms to generate path planning solutions for mobile robots was first presented by Jarvis *et. al.* [24]. This approach considers the task of path planning, as finding paths from the goal location back to the start location. This path planning procedure covers the environment with a uniform grid, and then propagates distances through free space from the goal cell (cells are assumed to be 8 connected). The distance wave front flows around obstacles and eventually through all free space in the environment. For any starting point within the environment representing the initial position of the mobile robot, the shortest path to the goal is traced by walking down hill via the steepest descent path. If there is no downhill path, in other words if the start cell is on a plateau then it can be concluded that there is no path from the start cell to the goal cell (note: initially all the cells are initialised to high values). Refer to Figure 1 for an example of the distance transform.

Despite the high computational overhead, distance transform path planning offers several advantages. Namely, the shortest path to the goal is known from all free space grid cells, thus supporting multiple robots. The distance transform readily supports multiple goals, refer to Figure 2 for an example. In this case a robot heads towards its nearest goal, similar to a fire evacuation drill, people evacuate a building via the closest fire exit. The distance transform also provides the capability to favour or avoid known regions in the environment. This feature is discussed in Section 4.

The Jarvis distance transform has a a problem with the ambiguity of optimal paths caused by considering diagonal neighbours to have the same cost as vertical and horizontal neighbours. However this problem can be overcome by considering the diagonal path to have the correct euclidean distance of $\sqrt{2}$. Borgefors [25] determined that the euclidean distances for distance transforms can be approximated as 3 units for horizontal paths and 4 units for vertical paths, as shown in Figure 3.

A disadvantage of the distance transform approach is the inefficiency of a grid-based approach, which leads to the costly computation of the distance transform through large areas of free space. The reported path planning algorithm addresses this problem by using the quadtree data structure extensively studied by Samet [28], as the environment model. Quadtrees are a good compromise between network/graph models and grid-based models. Quadtrees are a hierarchical structure with a grid like structure. The grid like structure of the quadtree simplifies path planning and allows sensor data to be readily included into the environment model. However, the quadtree overcomes the deficiencies of the grid-based models because the quadtree is adaptive to the clutter of the environment. The quadtree fragments into the smallest possible grid cells only along the boundaries of obstacles. Refer to Figure 4 for an example of a quadtree environment model. Both Kambhampati *et. al.* [12] and Samet [28] comment on the savings of a quadtree representation compared to a grid based representation. Kambhampati *et. al.* [12] show that the number of leaf nodes in a quadtree is proportional to the sum of the perimeters of the obstacles in the environment. Samet [28] reports that if the resolution of a quadtree is doubled, then the number of nodes in the quadtree will double, while with grids, doubling the resolution quadruples the grid size. Thus applying the distance transform to quadtrees instead of grids, will significantly improve the storage efficiency and the execution time of the distance transform.

Path planning for mobile robots using quadtrees has been reported by Kambhampati *et. al.* [12]. This path planning algorithm used the A* search to find paths in a known environment. The work reported in this paper addresses the problem of finding paths in unknown and partially known environments.

Environment mapping by a mobile robot can be accomplished into one of two ways; either by operating in "mapping" or "learning" modes. When a robot is operating in "mapping" mode [18] [19], it traverses the entire environment in a systematic manner, while scanning with onboard sensors and updating a map. The map is then used for all subsequent path planning exercises. Difficulties arise with this method if the environment is allowed to alter after the mapping has been completed.

The other mode of learning is to sense the environment, while executing paths which have been generated by a path planner. As obstacles are encountered en route to a goal, the model of the environment is updated and a new path to the goal is planned to avoid the obstacles [16] [26] [27]. The length of the paths generated by this system will initially be locally optimal. However as knowledge of the environment increases the optimisation of the length of planned path improves, until the planned path is globally optimal. Environment learning methods of this type have difficulty in detecting that a goal is not reachable since they use heuristics when planning locally optimal paths.

This paper will present an environment exploration algorithm, which is based upon the "learning" mode of environment mapping. The reported algorithm since it uses the distance transform for path planning, can detect that a goal is not reachable. The algorithm is not restricted to the recognition of line of sight distances to obstacles, thus it can be used with current generation sensing technology. Although the algorithm is capable of handling other sensory systems, only a mobile robot equipped with tactile sensing is discussed. It is assumed that the robot knows its own current world coordinates, as well as the coordinates of the goal. The robot has no knowledge about the location and shape of obstacles. No constraints are imposed upon the shape or location of the obstacles. It is assumed that the tactile sensor does not detect the shape of the obstacles lying in the robot's path. The tactile sensor informs the path planning algorithm that an obstacle has been encountered somewhere on the robot's exterior surface which is orthogonal to the direction of the robot's motion. The algorithm assumes the worst case about the size of the obstacle i.e. that the encountered obstacle is at least the same size as the robot.

Most path planners shrink the robot to point and expand all the objects in the environment by the robot's radius. This strategy is useful in known environments. However in unexplored environments this represents unnecessary extra processing. In learning situations, the information required by a mobile robot is distances to obstacles, from which the robot can decide whether or not it can pass through gaps between the obstacles. The reported algorithm does not expand all the objects the environment by the robots radius. Therefore it is the responsibility of the path execution procedure to ensure the robot does not collide with any obstacles. This is achieved by making sure the robot stays entirely within the boundaries of the free space quadrant which the robot is traversing. Conceptually this is equivalent to shrinking the robot to a point, and shrinking the boundaries of the leaf quadrant representing free space by the robots radius.

Since navigation paths are generated through shrunken free space leaf nodes of a quadtree, paths can only be considered in the horizontal and vertical directions. Considering paths that pass over the shared corner of adjoining adjacent free space quadrants could cause a collision. Diagonal corner paths tend to clip obstacles [12]. Thus the consideration of horizontal and vertical paths will result in safer robot paths. However the penalty for this approach is that paths are no longer distance optimal [12].

Attention in this paper will be restricted to two dimensional path planning without rotation. The robot is approximated to be a cylinder. The lowest resolution quadrant leaves in the quadtree are sufficiently small to fully accommodate the mobile robot.

Each node of the quadtree has storage for the pointers to the nodes's parent and its four children. Storage is also provided for the node classification; free, obstacle or grey (neither free or obstacle). Space is provided within the node for the confidence the navigation system has in this node of belonging to a particular class e.g. 90% confidence that this node is a free node. The node confidence is updated by the navigation system every time the robot visits or observes the node. Storage within the leaf node is also provided for the distance transform value which is used in path planning. Refer to Figure 5 for a diagram of the node structure.

2. Navigation Task Composition

The navigation task is composed of a number of subtasks. Refer to Figure 6 for a diagram of the task decomposition of the Navigation task. The navigation task is composed of three main subtasks; Path Planning, Path Execution and Model Update.

The Path Planning task when supplied a goal location applies the distance transform to the quadtree model of the environment, and produces all the possible solution paths from any location in the environment to the goal.

The Path Execution task locates the start location in the quadtree, and then traces the least cost path through the quadtree leaves to the goal. This task uses the subtask Optimal to find the most efficient paths through the solution path quadrants. The Move subtask controls the motion of the robot, and the robots onboard sensors. The Update Free subtask stores in the quadtree, the level of confidence that the system has in a particular leaf quadrant being free space.

The Model Update task is invoked when the navigation system detects an obstacle, which is not present in the quadtree model of the environment. The divide subtask has the responsibility of recursively subdividing the quadtree until leaves are obtained which span regions that are entirely free space or obstacle filled. The subtasks Store Free and Store Occupancy calculate the confidences that the navigation system has in leaf quadrants being free space or obstacle filled, and store these confidences in the quadtree.

3. Navigation Learning Algorithm

The navigation learning algorithm requires two pieces of information upon initialisation; the notional size $m \times n$ of the environment to be learnt, and a position reference (x,y,q) of the cylindrical robot to some reference point. A quadtree Q of sufficient size is generated to cover the area in which the robot will operate. The smallest quadtree leaf resolution size is of diameter d, a size which allows the robot to pass through. The size of quadtree $Q = (2d)^i$, where i is an integer such that $(2d)^i > s$ and $s = max\ (m, n)$.

Given the Cartesian coordinates of the start and goal locations, the best navigation strategy in an unknown environment is to optimistically assume that the unknown regions of the environment are free space. The confidence in this assumption is low. The navigation system invokes a path planning process, which plans a straight line path to the goal and calculates the distance transform cost between the start and goal locations.

If the path cost returned by the path planner is finite, then a solution path exists between the start and goal locations, otherwise the goal is unreachable. If the goal is reachable, the path generated by the planning process is passed onto the path execution process, which ensures the planned path is executed by the robot. Since the environment is unknown, the robot proceeds cautiously toward the goal.

One of two conditions will occur; either the robot reaches the goal or it encounters an obstacle. If the goal is attained the path execution process reports a success to the navigation procedure and the free space confidence that the navigation system has in this path is updated. In the case of an obstacle blocking the robots path, the path execution process returns the position and orientation of the robot and the robots sensor readings at this location, to the navigation system.

The navigation system invokes the navigation update process, which given the robots location and sensor readings updates the environment model Q i.e. the quadtree structure, and the confidence model of the environment. Upon completion of the update of the environment and confidence models, the path planning process is invoked again.

Essentially the path planning process locates the leaves of the quadtree where the current location and goal are found and then applies the distance transform to generate a solution path, or deduces that no solution path exists. The revised plan is then attempted by the robot. This cycle of plan - execute - update continues until the robot successfully reaches the goal, or the navigation system deduces that the goal is unreachable.

The optimisation criterion used by this path planner is to minimise distance between the start and goal locations. However this optimisation criteria can be altered to induce exploratory or conservative behaviour by the robot. When the robot is operating in an exploratory mode, the path planner favours unknown paths to the goal. If the robot is in a conservative frame of mind, the path planner will favour known paths to the goal. The general navigation algorithm is given in the procedure NAVIGATION.

```
procedure NAVIGATION( Q, start, goal, behaviour )
      repeat
           cost = 0
           perform PATH_PLANNING( Q, goal, cost, behaviour )
           if ( goal reachable ) then
                perform PATH_EXECUTE( Q, start, goal, stop, sensors )
                if ( stop ≠ goal ) then
                     perform MODEL_UPDATE( Q, start, stop, sensors )
                     start = stop
                end  if
           end  if
      until ( stop = goal or goal not reachable )
end  procedure
```

4. Path Planning Algorithm using Distance Transform

Conceptually the distance transform can be regarded as dropping a goal "pebble" into an environment "pond" and watching the ripples radiate out. When the distance transform is applied to the quadtree structure, distances are propagated through the quadtree from the goal quadrant leaf to the neighbouring quadrant leaves, which in turn propagate the distance transform to their neighbouring leaves. This process is continued until the distance transform flows through the whole quadtree. The distance transform is measured in multiples of the minimum sized quadrant. Except for the case of the quadrant leaf containing the goal, the distance transform is calculated as the straight line distance from the goal location to the nearest edge of neighbouring leaves (note: the goal can be located anywhere within the goal quadrant). This measure identifies the neighbouring quadrants which are closest to the goal. Refer to Figure 7 for an example of the algorithm. Note: The distance transform values stored in all the quadrants are initialised to ∞ (in practise this is the largest available integer) before the path planner is invoked.

The general algorithm for distance transform is given in the procedure PATH_PLANNING. The procedure GET_NEIGHBOUR finds the next neighbour leaf to the current leaf in a particular direction. If more than one neighbour exists then they are accessed in EAST-WEST or NORTH-SOUTH order. The algorithm used in GET_NEIGHBOUR is a variant of Samet's neighbour finding algorithm [29].

The function FACTOR is used to give a weight to the distance transform depending on the leaf type; known, unknown or obstacle. Such an approach was used by Jarvis *et. al.* [24], where the largest possible integer was used for obstacles as the FACTOR value, thus ensuring that no obstacle cells were included in the solution. A FACTOR of 1 was used for known and unknown cells. However the FACTOR could be varied to induce different behaviours. If an inquisitive behaviour was required the FACTOR in known cells was doubled. If on the other hand a behaviour was sought where the robot avoided unknown cells, the FACTOR in unknown cells was doubled.

The path planner described in this paper uses a FACTOR function which it evaluates by the following rules. In all modes of navigation the value of the FACTOR function for obstacle leaf nodes is considered to be ∞ (in practise this is the largest available integer) . If the robot is in a learning mode, free spaces and unknown spaces are considered to belong to the same leaf class, since the best assumption that can be made of unknown space is that it is free space. Once the robot has learnt a portion of the environment, there could be a need to find the conservative path to a goal using the free spaces in which the navigation system possesses the greatest knowledge. Alternatively the robot may operate in an adventurous frame of mind and favour unknown spaces on route to a goal. The cost functions which are used in to generate the various behaviour distance transforms for a quadrant *q* are as follows:

$$\text{cost}(q) = \min(\forall \text{cost}(n_i)) + \text{size}(q) * \text{factor}(\text{behaviour}, q)$$

where

min($\forall$*cost(*n_i*))* returns from the set of quadrants which surround quadrant q the neighbouring quadrant n_i with the least cost.

size(q) returns an integer which represents the size of the quadrant q as a multiple of the smallest allowable quadrant in the quadtree representation.

factor(behaviour, q) returns a weight depending on the type of navigation behaviour that is sought, and the amount of information that is known about quadrant q.

if *behaviour = learning* then *factor = 1*

if *behaviour = conservative* then *factor = 1 + [1 - CONFIDENCE]*,

where *1 - CONFIDENCE* is a measure of the confidence the system has in the quadrant q NOT being free space.

if *behaviour = adventurous* then *factor = 1 + CONFIDENCE* ,

where *CONFIDENCE* is a measure of the confidence the system has in the quadrant q being free space.

The CONFIDENCE value of a free space quadrant is calculated by determining the area that has been swept by the robot inside a quadrant during the execution of a planned path, and dividing it by the area of the quadrant. The robot may navigate in a mapping mode, a mode which attempts to learn all the unknown areas, this can be accomplished by making the free space quadrants with the lowest confidences the goals, and propogating an inquisitive path transform. This will cause the robot to favour traveling through unknown areas enroute to the foreign goal.

```
procedure PATH_PLANNING( Q, leaf, cost, behaviour)
    minimum = EXTRACT( leaf )
    if ( cost < minimum ) then
        perform STORE( leaf, cost )
        for direction = EAST, WEST, NORTH and SOUTH do
            perform GET_NEIGHBOUR( Q, leaf, direction,neighbour )
            do while ( more neighbours exist )
                newcost = FACTOR( behaviour, leaf ) * SIZE( leaf ) + cost
                perform PATH_PLANNING( Q, neighbour, newcost, behaviour )
                perform GET_NEIGHBOUR( Q, leaf, direction,neighbour )
            end do
        end for
    end if
end procedure
```

5. Path Execution Algorithm

Upon the completion of path planning, the navigation system is ready to execute the planned path. The general algorithm for path execution is given in the procedure PATH_EXECUTE. The algorithm must first isolate the location of the robot into a leaf of the quadtree Q. The distance transform stored in this leaf is examined. If the distance transform = 0, this means that the start and goal points are located in the same leaf, therefore the robot can proceed directly to the goal. If the distance transform > 0, the vertical or horizontal neighbouring leaf with the minimum distance transform must be found. A subgoal point in the minimum valued distance transform quadrant is selected, and the robot proceeds to this subgoal. The robot will either reach the subgoal or it will encounter an obstacle. If the robot reaches the subgoal, the free space confidence between the start and subgoal is updated. Upon updating of free space confidence the next subgoal is found and the path to this subgoal is attempted. This process is applied repeatedly until the goal is attained or an obstacle blocks the robots path. If an obstacle is sensed the robot stops. The path execution is terminated and the robot location and the tactile sensor values are returned to the navigation system.

```
procedure PATH_EXECUTE( Q, start, goal, stop, sensors )
     obstacle = FALSE
     repeat
          leaf = LOCATE( Q, start )
          distance_transform = EXTRACT( leaf )
          if ( distance_transform = 0 ) then
               subgoal  = goal
          else
               subgoal = OPTIMAL( Q, leaf,goal )
          end  if
          perform MOVE( subgoal, stop, sensors )
          if ( stop = subgoal ) then
               perform UPDATE_FREE( Q, start, stop )
          else
               obstacle = TRUE
          end  if
          start = subgoal
     until ( start = goal or obstacle )
end  procedure
```

6. Optimal Path Algorithm

The OPTIMAL function searches the vertical and horizontal neighbours of the quadrant leaf the robot is located in, to find the leaf quadrant which yields the least cost path. Finding a neighbouring quadrant with the minimum distance transform does not guarantee a least cost path. The distance transform stored in a quadrant, represents the cost of traversing the quadrant from a boundary edge to the goal. To determine the true path cost to the goal, the cost of the path between the robot's location and the boundary edge of the neighbour quadrant, must be calculated and added to the quadrants distance transform. Once the least cost free space quadrant has been found, a path can be executed to this quadrant from the robot's current location.

During path execution a number of strategies can be used to generate a path through the free space quadrants, these are summarised in Figure 8. A reasonable strategy would be to steer through the middle of the intersection of entry and exit boundary edges of quadrants. Such a strategy generates relatively "safe" paths, and is reasonable during the exploration of an environment by the robot.

An optimal path could be found by constructing a visibility graph, between the robot's location and the goal. This is not a great computational burden since a course path of free space quadrants to the goal is known. However the work effort of this computation is likely to be wasted in an environment which is not well known, since an unexpected obstacle will result in path replanning. A solution to this problem is as follows. Instead of constructing the "full" visibility graph, look ahead *n* quadrants and construct a "reduced" visibility graph between the robot's location and the mid point of the exit edge of the *nth* quadrant (except in the case of the goal quadrant where the goal takes the place of the midpoint of the exit edge). The visibility graph is "extended" as the robot traverses the free space quadrants toward the goal. The "reduced" visibility graph approach will not always yield the optimal path, but it will give a path which is near optimal. Refer to Figure 9 for an example of optimal paths with "full" and "reduced" visibility graphs. Choosing the size of the look ahead can be related to the confidence in the knowledge of the environment. In a completely unknown environment one quadrant look ahead is sufficient. Once an environment is well known, the optimisation of the paths can be improved by looking ahead more than one quadrant. The navigation system could have a set of threshold environment confidence values with an appropriate look ahead prestored. Depending on the confidence in the knowledge of the environment, an appropriate look ahead is selected.

This strategy produces reasonable paths, however in the cases where the robot must steer through quadrants of the smallest size resolution, zig-zag paths could be generated. This is because paths between quadrants sharing a corner are not considered. This can be countered with the rule; if the quadrant leaf is of the smallest resolution, and the entry and exit edges are normal to one another, and the neighbouring quadrant is free space, then the path segment through this quadrant can be omitted. This causes the robot to steer diagonally between the free space quadrants. By combining the function OPTIMAL and the zig-zag smoothing rule reasonable paths can be produced.

7. Update Free Space Confidence Algorithm

To determine the confidence the navigation system has in a quadrant being free space, the system keeps track of what proportion of the quadrant's area has been swept by the robot. The free space confidence of a leaf is determined by calculating the area swept by the robot in the quadrant and dividing it by the area of the quadrant. This value represents the free space confidence, for the current visit by the robot to the quadrant, and it is added to the adjusted free space confidence generated on previous visits to the quadrant by the robot. The free space confidence currently stored in this quadrant, describes the free space confidence of the whole leaf, it must be adjusted to describe the free space confidence of the remaining portion of the quadrant not swept by the robot on this visit. This is done by multiplying the confidence value stored in the leaf, by the free space confidence of the area not swept by the robot on this visit.

The free space confidence value could also include the free space detected by sensors onboard the robot using the approach followed by Moravec *et. al.* [18], where an indoor environment was mapped using wide angle sonar sensors. This approach was motivated by the idea, that when an area to be mapped is observed from different positions in the environment, given sufficient observations, it can be concluded with high confidence that the area belongs to a particular class i.e. is free space or an obstacle. However due to the uncertainty of sensor readings, sensor free space confidence should be given less weight than free space confidences generated from the areas visited by the robot.

```
procedure UPDATE_FREE( Q, start, stop )
     leaf = LOCATE( Q, start )
     confidence = [DISTANCE( start, stop ) * ROBOT_AREA] / AREA( leaf )
     free_confidence = GET_CONFIDENCE( leaf ) * [1 - confidence] + confidence
     perform STORE_CONFIDENCE( leaf_start, free_confidence )
end  procedure
```

8. Model Update Algorithm

Once the robot has completed path execution, the navigation system is ready to update the environment and confidence models of the environment. The general algorithm is given in the procedure MODEL_UPDATE. Path execution terminates on one of two conditions; either the robot reached the goal or an obstacle was encountered. If the robot reached the goal, the environment model does not change, however the confidence that the environment is of the current model structure is increased. If an obstacle is encountered the environment model must be updated to reflect the presence of the freshly sensed obstacle and the confidence in the environment is also updated. To perform the update of the models, the algorithm must know the current location of the robot, the sensor readings at this location, and the location of the robot when this procedure was last invoked.

Once the algorithm updating the environment models is invoked, the current leaf location of the robot in the quadtree is found. This leaf is checked to make sure that none of the obstacle sensor readings occur in the leaf quadrant. If the sensor readings occur inside the leaf quadrant the robot is currently located, then the leaf quadrant must be divided into four quadrants. Leaf division is continued until the leaf is isolated from the sensor readings, or the leaf reaches the smallest size resolution allowable. Every time a leaf is divided, the free space confidences of the new leaves must be calculated. This is done by fitting a line between the current and previous robot locations, determining what area the robots path swept in the newly generated leaves and then dividing this area by the area of the new leaf. Upon the completion of isolating the current robot leaf location from the sensor readings, and the updating of the free space confidences, the sensor readings detected by the robot are each in turn isolated to a leaf of the smallest size resolution, this may require the division of leaf quadrants and the updating of free space confidences. Once a sensor reading has been isolated to a leaf, an estimate is made of what proportion of the leaf is occupied by the obstacle and this information is recorded.

By isolating the sensor reading to the smallest resolution leaf, this may be seen as unnecessarily fragmenting the leaves of the quadtree which span obstacles. A consolidation procedure is needed which can detect that neighbouring leaves of a quadtree are part of the same obstacle and thus the quadtree can be pruned back to the parent of the leaves. The consolidation procedure can work in conjunction with the navigation system when the robot is operating in a mapping mode. When the robot is navigating in a mapping mode, it places goals in the centre of the leaf quadrants which have the lowest free space confidences. If the path planning procedure deduces that a goal is not reachable from any direction, this implies that the goal is surrounded by obstacles. The consolidation procedure can then prune the quadtree, to reflect the knowledge that the region surrounding the goal is one obstacle. Refer to Figure 10 for an example of the consolidation algorithm.

```
procedure MODEL_UPDATE( Q, start, stop, sensors )
    repeat
        leaf = LOCATE( Q, stop )
        if ( leaf is smallest resolution )  then
            perform STORE_CONFIDENCE( leaf, 1.0 )
            colour = CLASSIFY( leaf, sensors )
            perform STORE_COLOUR( leaf, colour )
        else
            perform STORE_COLOUR( leaf, GREY )
            memory = GET_CONFIDENCE( leaf ) / 4
            perform DIVIDE( leaf, child )
            for i = 1 to 4 do
                perform STORE_CONFIDENCE( child[i], memory )
            end  for
    until ( leaf is smallest resolution )
    perform CONSOLIDATE( Q )
end  procedure
```

9. Partial Distance Transform Update Algorithm

Computing the full distance transform every time the robot encounters an obstacle is as an unacceptable computational burden. Considerable savings can be made if the distance transform is only partially updated when the environment model changes. The quadtree modelling of the environment can be elegantly used to limit the recomputation of the distance transform.

If the robot start and goal locations can be found in the same minimal subtree of the quadtree, the distance transform need only be calculated for the subtree. Considering a subset of the possible solution paths will yield either; locally optimal paths or no solution paths. Finding locally optimal paths is acceptable when the robot is learning an environment. To find globally optimal paths or if no solution paths can be found locally, simply move up one level in the quadtree structure and compute the distance transform for the fresh subtree.

Such a strategy is useful when the robot gets closer to the goal, however it does not limit the recomputation of the distance transform when the start and goal are separated by large distances. It is still possible to limit the recomputation to a partial update in such instances. When a change to the quadtree model of the environment occurs, isolate the change to a minimal subtree. The distance transform of this subtree either effects or is effected by its neighbours. Find the minimal subtree which isolates the subtree of change and its neighbours. If the goal is located in the minimal subtree then calculate the distance transform for this subtree only. Otherwise recompute the distance transform from all the neighbours of the subtree, but only in the direction of minimal subtree. The partial distance transform update mechanism can be incorporated into the navigation system. The modified navigation algorithm is given in the procedure NAVIGATION_2. Assume the procedure ISOLATE, finds the common subtree of two leaves.

```
procedure NAVIGATION_ 2( Q, start, goal )
     SQ  = ISOLATE ( Q, LOCATE ( Q, start ), LOCATE ( Q, goal ) )
     repeat
          perform PATH_PLANNING ( SQ, goal )
          if ( goal reachable ) then
               perform PATH_EXECUTE ( SQ, start, goal, location, sensors )
               if ( location ≠ goal ) then
                    perform MODEL_UPDATE ( SQ, start, location, sensors )
                    start = location
               end  if
               SQ = ISOLATE ( SQ, LOCATE( SQ, start ), LOCATE( SQ, goal ) )
          else
               SQ = PARENT ( Q, SQ )
               if ( SQ = nil ) then goal not reachable
          end  if
     until ( location = goal or goal not reachable )
end  procedure
```

10. Example of Exploration Algorithm

For an example of the exploration algorithm using the full distance transform refer to Figure 11. Once the robot has completed a journey in the unknown world to a selected goal, it has learnt a portion of the environment. Refer to Figure 12 Diagram A for an illustration of the confidence model of the environment after the completion of the journey to the goal. At this stage the robot can exhibit different path planning behaviours. The confidence model can now be used to plan paths which favour or avoid unexplored regions of the environment. Refer to Figure 12 for examples of this type of path planning. An example of the use of the partial distance transform is given in Figure 12.

Experimental results of this exploration algorithm showing execution timings for the new algorithm and a comparison against grid based path planners are presented in [32][33][34].

11. Discussion

Using quadtrees to represent polygonal shaped obstacles yields a quadtree with minimum sized leaf quadrants all along the edges of the polygon. The quadtree is large and the number of leaves in the tree is proportional to the polygon's perimeter. Solutions put forward by Samet *et. al.* using PM-quadtrees [30] and by Ayala *et. al.* [31] using extended-quadtrees, offer substantial reductions in the memory requirements to represent polygonal obstacles (refer to Figure 13 for an example). However these variants of the quadtree are not suitable for use with the path planner presented in this paper, since these representations only store the vertex points of a polygonal shape. The polygonal edge information which is important for path planning is not explicitly stored, thus making it difficult for the path planner to determine a path through a quadrant containing a polygonal edge without extensive computation.

An interesting concept would be to add a forget function to the navigation system. In this case the forget function would periodically reduce the robots knowledge of free space by a percentage factor. This percentage factor could be varied through the quadtree depending on where the robot is current located. Leaf quadrants which are in close proximity to the robot should be less easily forgotten than those further away. This would have the effect of forcing the robot to relearn the environment, particularly regions which had not been visited for some time. A forget function could help model environments that changed with time.

By considering the lowest size resolution of the quadtree to be the same size as the robot, this could exclude possible paths if a leaf is only partially occupied. Further investigation is warranted, possible ways forward would be to further apply the quadtree division of space, but have exceptions to the rule that a robot can only travel within a quadtree leaf. Another possibility is to used to "relax" the quadtree and move the quadrant leaf a sufficient amount until it is free space, and then attempt a path through the overlapping quadrants.

12. Conclusions

Quadtrees and distance transforms provide an effective mechanism for learning an environment with a mobile robot. The environment can be efficiently modelled with quadtrees, and distance transforms can be applied to explore paths in known and unknown portions of the environment. The problem of expensive computation of the distance transform is addressed by a concurrent processing implementation, and limiting the recomputation of the distance transform to a partial update. Mechanisms have been provided for the efficient execution of planned paths. It can seen that by varying the manner in which the distance transform is generated, different behaviours can be induced from the navigation system, such as exploratory, conservative and finding optimal path behaviours.

13. Acknowledgments

The author gratefully acknowledges Neil Smith for his suggestions and ideas concerning this research. The author extends his thanks to the referees, whose suggestions have improved the presentation of this paper.

14. Bibliography

[1] T. Lozano-Perez and M.A. Wesley, "An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles", Communications of the ACM, Vol. 22 No. 10, pp560-570, October 1979.

[2] L. Gouzenes, "Strategies for Solving Collision-free Trajectory Problems for Mobile and Manipulator Robots", International Journal of Robotics Research, Vol. 3 No. 4, pp51-65, 1984.

[3] R.A. Brooks, "Solving the Find-Path Problem by a Good Representation of Free Space", IEEE Trans. on Systems, Man and Cybernetics, SMC-13 No.3, pp190-197, March 1983.

[4] D.T. Kuan, J.C. Zamiska and R.A. Brooks, "Natural Decomposition of Free Space for Path Planning", Proceedings of IEEE International Conference on Robotics and Automation, pp168-173, March 1985.

[5] D.P. Miller, "Planning by Search Through Simulations", Phd dissertation, Yale University, Department of Computer Science, October 1985.

[6] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach", IEEE Trans. on Computers, C-32 No. 2, pp108-120, February 1983.

[7] R.A. Brooks and T. Lozano-Perez, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", IEEE Trans. on Systems, Man and Cybernetics, SMC-15 No.2, pp224-233, March/April 1985.

[8] J.T. Schwartz and M. Sharir, "On the Piano Movers Problem, the Case of a Two Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers",Communications on Pure and Applied Mathematics, Vol. 36, pp345-398, 1983.

[9] J.T. Schwartz and M. Sharir, "On the Piano Movers Problem, the Case of a Rod Moving in Three Dimensional Space Amidst Polyhedral Obstacles",Communications on Pure and Applied Mathematics, Vol. 37, pp815-848, 1984.

[10] S.M Udupa ,"Collision Detection and Avoidance in Computer Controlled Manipulators" Proc. of 5th International Joint Conference on Artificial Intelligence, pp737-748, August 1977.

[11] J.S. Singh and M.D. Waugh, "Robot Path Planning using Intersecting Convex Shapes: Analysis and Simulation", IEEE Journal of Robotics and Automation, Vol. RA-3 No. 2, pp101-108, April 1987.

[12] S. Kambhampati and L.S. Davis, "Multiresolution Path Planning for Mobile Robot", IEEE Journal of Robotics and Automation, Vol. RA-2 No. 3, pp135-145, September 1986.

[13] V.J. Lumelsky and A. A. Stepanov, "Path Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape", Algorithmica, Vol. 2 No. 4, pp403-430, 1987.

[14] A.M. Thompson, "The Navigation System of the JPL Robot", Proceedings of 5th International Joint Conference on Artificial Intelligence, pp749-757, August 1977.

[15] R. Chatila, "Path Planning and Environment Learning", European Conference on Artificial Intelligence, pp211-215, July 1982.

[16] H.P. Moravec, "Obstacle Avoidance and Navigation in the Real World by a Seeing Rover", Phd dissertation, Stanford University, September 1980.

[17] R. Chatila and J-P. Laumond, "Position Referencing and Consistent World Modelling for Mobile Robots", Proceedings of IEEE International Conference on Robotics and Automation, pp138-145, March 1985.

[18] H.P. Moravec and A. Elfes, "High Resolution Maps from Wide Angle Sonar", Proceedings of IEEE International Conference on Robotics and Automation, pp116-121, March 1985.

[19] J.L. Crowley, "Navigation for an Intelligent Mobile Robot", IEEE Journal of Robotics and Automation, Vol. RA-1 No. 1, pp31-41, March 1985.

[20] R.A. Brooks , "Aspects of Mobile Robot Visual Map Making", Proceedings of 2nd International Symposium of Robotics Research, pp287-293, August 1984.

[21] S.V.N. Rao, S.S. Iyengar, C.C. Jorgensen and C.R. Weisbin, "Robot Navigation in an Unexplored Terrain", Journal of Robotic Systems, Vol.3 No.4, pp389-407, 1986.

[22] S.S. Iyengar, C.C. Jorgensen, S.V.N. Rao and C.R. Weisbin,"Robot Navigation Algorithms using Learned Spatial Graphs", Robotica, Vol.4, pp93-100, 1986.

[23] C.E. Thorpe, "FIDO:Vision and Navigation for a Robot Rover", Phd dissertation, Carnegie-Mellon University, Department of Computer Science, December 1984.

[24] R.A. Jarvis and J.C. Byrne, "Robot Navigation: Touching, Seeing and Knowing", Proceedings of 1st Australian Conference on Artificial Intelligence, November 1986.

[25] G. Borgefors, "Distance Transform in Arbitary Dimensions", Computer Vision, Graphics and Image Processing, 27, pp321-345, 1984.

[26] A. Elfes, "Sonar-Based Real World Mapping and Navigation", IEEE Journal of Robotics and Automation, pp249-265, June 1987.

[27] G. Giralt, R. Chatila and M. Vaisset, "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots",The First International Symposium of Robotics Research, MIT Press, pp191-214, 1984.

[28] H. Samet, "An Overview of Quadtrees, Octrees and Related Hierarchical Data Structures", NATO ASI Series, Vol. F40, Theoretical Foundations of Computer Graphics, pp51-68, Springer-Verlag Berlin Heidelberg, 1988.

[29] H. Samet, "Neighbour Finding Techniques for Images Represented by Quadtrees", Computer Graphics and Image Processing, Vol. 18 No. 3, pp37-57, 1982.

[30] H. Samet and R.E. Webber, "Storing a Collection of Polygons Using Quadtrees", ACM Transactions on Graphics, Vol. 4 No. 3, pp182-222, July 1985.

[31] D. Ayala, P. Brunet, R. Juan and I. Navazo, "Object Representation by Means of Nonminimal Division Quadtrees and Octrees", ACM Transactions on Graphics, Vol. 4, No. 1, pp41-59, January 1985.

[32] A.Zelinsky, "Environment Exploration and Path Planning Algorithms for a Mobile Robot using Sonar", Phd dissertation, University of Wollongong, Department of Computer Science, October 1991.

[33] A. Zelinsky, "Environment Modelling with Autonomous Mobile Robots using Low Resolution Sonar". Second IARP Workshop on Sensor Fusion and Environment Modelling, September 1991, Oxford UK.

[34] A. Zelinsky, "A Comparison of Distance Transform based Mobile Robot Path Planners". *submitted* to Robotica.