Towards an Efficient Optimal Trajectory Planner for Multiple Mobile Robots

Jason Thomas, Alan Blair, Nick Barnes
Department of Computer Science and Software Engineering
University of Melbourne
Victoria, 3010, Australia
Email: {jrth, blair, nmb}@cs.mu.OZ.AU

***Abstract*— In this paper, we present a real-time algorithm that plans mostly optimal trajectories for multiple mobile robots in a dynamic environment. This approach combines the use of a Delaunay triangulation to discretise the environment, an efficient use of the A* search method, and a cubic spline representation for a trajectory that meets the kinematic and dynamic constraints of the robot. We show that for complex environments the shortest-distance path is not always the shortest-time path due to these constraints.**

I. INTRODUCTION

Path planning is fundamental to mobile robots and has been studied extensively over many years. Path planning is the process of calculating a path for a mobile robot to follow so that it can move from one pose (position and orientation) to another, while avoiding collisions with obstacles. Trajectory planning is the extension of path planning to consider velocity. A wide variety of methods have been developed for path planning and can be broadly classified as either local planners or global planners. Local planners derive a path based on local sensors on the robot and do not make use of a map of their environment. As such, local planners are reactive, based on immediate sensor data. Conversely, global planners utilise a map of the world, either as *a priori* knowledge, sensed by some global sensor (e.g. a camera over the environment), or built up through exploration. Beyond finding a path, some global planners attempt to find the optimal or near-optimal trajectory for a mobile robot and in some cases multiple mobile robots. This however, has been limited to static environments only. We see a need for an algorithm to find optimal trajectories for multiple robots, in a dynamic environment, in real-time. We have developed an algorithm that will find optimal trajectories for multiple robots in most cases, and present simulation and real robot experimental results.

A. Local Planners

There are a number of algorithms for planning from local sensor data. We discuss three popular methods: potential fields; the vector field histogram; and the dynamic window approach.

Potential fields[1][13], sometimes referred to as vector force fields (VFF) or velocity fields, are based on the concept of electric or magnetic fields. Various objects in the environment can impart an attractive or repulsive force on the robot. This method is computationally fast, however it does suffer from certain limitations. Often attractive and repulsive forces can negate each other resulting in the robot stopping, or moving slowly. Potential fields, as with all purely local planners, cannot handle the box-canyon problem (a local minima) as illustrated in figure 1.

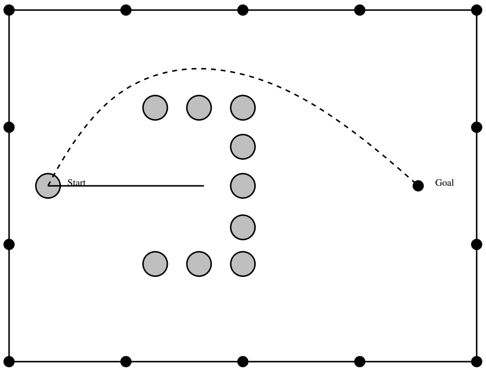

Fig. 1. Illustration of the Limitations of Local Planners. The dotted line is an optimal path, the solid line is a typical path for a local planner. Note that it does not find a path to the goal pose.

The Vector Field Histogram (VFH) algorithm [2] aims at handling the vector negation problem in potential fields by using a histogram. The VFH first creates a one-dimensional polar histogram then selects the most suitable sector for the steering direction. This allows the robot to maintain a higher average velocity.

The Dynamic Window Approach [3] goes further to search through the space of possible instantaneous velocities of the robot, taking into account the dynamics of the robot. This is achieved by reducing its search space to only circular trajectories which can be uniquely determined by pairs of translational and rotational velocities, a two-dimensional search space. Only velocities that are safe and will not collide with obstacles given the known *dynamic* constraints are allowed and only obstacles within a small *window* of space, reachable with a short amount of time, are considered. However, due to the lack of knowledge of the whole environment, all local planners are at best able to produce a globally optimal path in trivial environments.

B. Global Planners

Knowledge of the entire environment allows global planners to search for a complete path from the starting pose to the goal pose. This knowledge must be represented in a form that can be searched. This generally translates to discretising the world in some way. The typical approaches for this can be categorised into cell decomposition, or graph based methods. We examine the use of occupancy grids which is a form of cell decomposition, and Voronoi and visibility graphs, which are graphical methods.

Methods using Occupancy Grids [4] represent the environment as a grid. The resolution of the grid can vary depending on the fidelity of the available information, either *a priori* knowledge or sensed. Grid coordinates are classified as occupied by an obstacle, empty, or unknown. This grid can then be searched across for available paths either using an A* search [4] or the distance transform [11]. For a high resolution grid the search space can become large leading to high computational cost.

Graph based methods try to overcome this scaling problem by representing the knowledge as a graph. The size of the graph is dependent on the complexity of the environment (number of obstacles) rather than physical size. This abstraction permits applications in larger environments. Graph based path planning is often referred to as road-map[9] planning since the arcs on the graph can be considered roads the robot can traverse with the vertices as intersections. This graph must then be searched (e.g. using A* search) in some way to find a path.

Generalised Voronoi Diagrams (GVD) [5] are often used for this purpose. The GVD is defined as the locus of points which are equidistant from two or more obstacle boundaries including the workspace boundary [12]. A GVD can be generated in $O(NlogN)$[5]. GVDs result in a path that is as far away from all obstacles as possible. This is a good representation when the space between obstacles is slightly larger than the robot itself, however it is conservative if the relative size of the robot to the space between obstacles is small, since this results in a path that is sub-optimal due to the equidistant constraint on the graph.

Voronoi diagrams have also been used for structure extraction [6] to generate topological maps from other maps. A Voronoi diagram can be efficiently created from its dual, the Delaunay Triangulation, which is sometimes easier to extract directly than the Voronoi graph.

Additionally, visibility graphs are often used in robot motion planing [9]. Each obstacle is reduced to a vertex list. For simple objects this can often be one vertex representing a point source, however walls and polygons can be represented as a connected vertex list. A graph is produced such that each arc on the graph connects vertices that are visible to each other. When used for path planning the starting point and goal point are included in the graph, and the planner simply searches through the graph from the start vertex to the goal vertex. This can easily be implemented with an A* search using Euclidean distance as the heuristic. Algorithms for constructing and searching this graph exist with running time $O(N^2)$. While a Voronoi graph will force a path far away from an obstacle, a visibility graph has the opposite effect of taking the path close to the obstacle. This increases the risk of collisions with obstacles and can result in a sub-optimal path.

C. Optimal Planners

Optimal motion planners find a collision free path, with a the trajectory that is optimal according to some cost function, for example: the shortest distance or time to reach the final pose. This has previously been done using cubic splines to control a trajectory through an environment for robot arms working together [17]. However, without first reducing the environment to a small search space this problem is computationally expensive and must be performed offline. As a result, this method is intolerant to dynamic obstacles.

Cubic splines have also been used with kinematic visibility graphs to generate a path [7]. However, creating the visibility graph is time-expensive forcing the graph generation to be performed off-line. A computationally faster method for cubic splines uses randomly selected initial points and applies a genetic algorithm (GA) [16]. Both, again, are only used on static environments and are intolerant to dynamic obstacles.

Another method for optimal planning [8] has been developed for use in a dynamic environment, using a search through a five dimensional space. The dimensions are the pose (x, y, θ), and translational and rotational velocities (v, ω). This method first finds the shortest distance path across an occupancy grid using an A* search, then establishes how far it can look ahead for the five dimension search based on computational power available. For the space established each dimension is discretised by small amounts (e.g. 10cm for position, $\pi/16$ degrees for orientation) and the five dimensional space is searched, where each dimension is discretised by small amounts (e.g. 10cm for position, $\pi/16$ degrees for orientation). This algorithm finds the shortest distance path globally, and the shortest time trajectory locally. Finding the shortest time trajectory globally would be preferable.

To find the shortest time trajectory globally, we must search across the dynamic space for the full path. For closed, controlled environments, such as a factory floor, offices, homes, shopping malls, or robotic soccer fields, this is made possible by mounting global sensors around the environment. The difference in shortest distance and shortest time trajectories produced is illustrated in figure 5.

II. Algorithm

The algorithm we have developed is based on a graphical representation of the environment (e.g. Fig 3) in the form of a Delaunay triangulation (Fig 4). This graph contains useful information for the purposes of path planning which is not present in a Voronoi or visibility graph. Each arc on the graph represents a line joining two obstacles that are closest to each other. It is exactly this information that is important to a path planner plotting paths through the gaps between obstacles. In order to optimise the trajectory, we consider these arcs as constrained variable way-points through which the trajectory must pass (Fig 2). If we are treating the robot and obstacles as point sources we must reduce the line segment

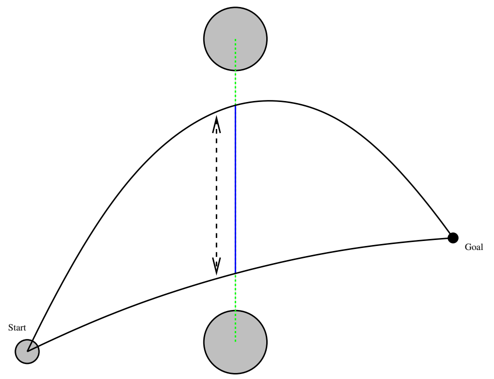

Fig. 2. **Illustration of way-point tuning**. The way-point can be moved along the line to get a curve with a faster velocity profile.

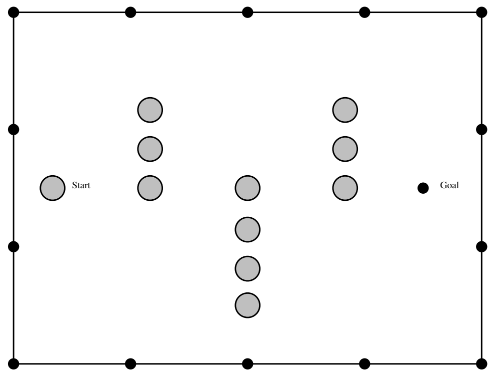

Fig. 3. **Illustration of an environment**. There are a number of obstacles blocking a direct path to the destination pose. The boundary walls have points highlighted. The separation between points is relative to the robots size.

representing the constrained variable way-point along the arc to the robot's configuration space (the physical space the robot is able to occupy). The ability to tune the way-points gives us an advantage over Voronoi and visibility graphs.

Before searching we can prune any arcs that the robot cannot traverse. These arcs occur along walls and when obstacles are too close to each other for the robot to pass between them. This has the effect of making obstacles in close proximity to each other appear topologically as one obstacle. This pruning, and the fact that the graph is a triangulation, results in a low branching factor. If we also prevent the search from entering the same triangle twice we have a branching factor of 0, 1, or 2 at any point. This reduces the search space considerably.

We search the graph using an A* search. This searches based on a cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of reaching a point in the graph, and $h(n)$ is an admissible heuristic (never overestimates) estimate of the cost to the goal. A* expands nodes in order of smallest $f(n)$ first. We can continue expanding nodes after we have our first solution to get a sorted list of all paths.

Ideally we would use the cubic spline time calculated below as the cost function $g(n)$ for the search, however this would be too computationally expensive to run at frame-rate. For efficiency we use straight-line distances (SLD) connecting the way-point on one arc with the way-point on the next. We use the same cost function for $h(n)$ connecting the current way-point to the goal point.

We can calculate a lower bound on the traversal time for a path by dividing the total SLD of the line segments, by the maximum speed at which the robot can travel. This time ($time_{SLD}$) is a lower bound because it is the shortest distance path, and we are ignoring the kinematic and dynamic constraints of the robot which prevents it from travelling at maximum speed all the time.

Once a path of straight-line segments reaches the goal node we convert it to a cubic spline trajectory. This is discussed in II-B. As shown in section II-A, we can calculate an accurate estimate of the time ($time_{cubic}$) to traverse this path. Since this trajectory is constrained by physics it will always take a longer time to traverse then the straight-line estimate ($time_{SLD}$) above.

We continue converting straight-line segment paths, which we get in sorted order from the A* search, to cubic spline trajectories until we get a path length that results in a $time_{SLD}$ estimate that is longer than that of our best cubic spline trajectory ($time_{SLD} >$ best $time_{cubic}$). No further searching is required at this point as it is impossible for that path to result in a trajectory that will be better than the best one already converted. This algorithm is shown as pseudo-code:

---

for each frame
  Update map from sensor data (Fig 3)
  Build Delaunay graph from new map (Fig 4)
  for each robot, calculate a trajectory
    Perform A* Search on graph (using Euclidean distance)
      Calculate *straight line time* ignoring dynamics:
      $time_{SLD} = \frac{distance}{V_{max}}$
      if goal point is reached **then**
        if $time_{SLD} >$ best $time_{cubic}$ **then**
          stop searching
        else
          convert new path to cubic spline trajectory (Fig 5)
          Calculate *cubic spline time* considering
          dynamics (Sec II-A): $time_{cubic}$
          if $time_{cubic} <$ best $time_{cubic}$ **then**
            best $time_{cubic} = time_{cubic}$
            keep new cubic spline trajectory as best
          continue searching
    return best cubic spline trajectory

A. Parametric Cubic (PC) Curves

We use Parametric Cubic (PC) curves as they can represent a space-time curve, or trajectory, algebraically. A unique PC curve can be generated from the following information: initial position (P_{x_0}, P_{y_0}) and velocity (V_{x_0}, V_{y_0}); final position (P_{x_s}, P_{y_s}) and velocity (V_{x_s}, V_{y_s}); and maximum speed (V) and acceleration (A). We can solve a set of equations to find the time (s) it takes to complete this curve.

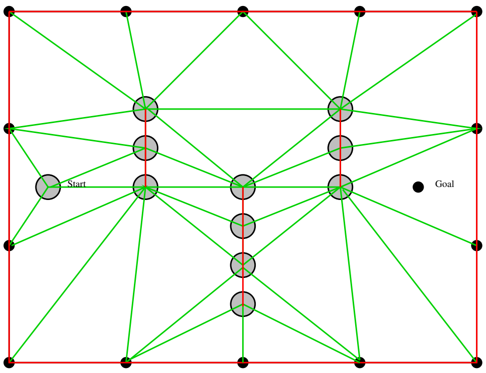

Fig. 4. **Illustration of a Delaunay Graph**. Arcs that cannot be traversed are pruned. The search is not permitted to cross the same triangle twice for any given path. This reduces the branching factor to either one or two, but no more.

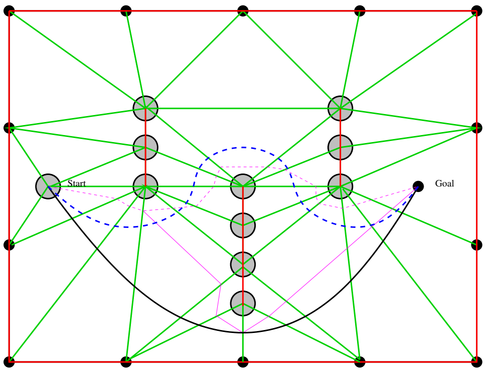

Fig. 5. **Illustration of Differences between distance and time cost functions**. The dotted spline is the first path found. The bold solid spline is a longer distance path, that has a shorter time. The straight-line segment paths are those generated by the A* search for each of the trajectories.

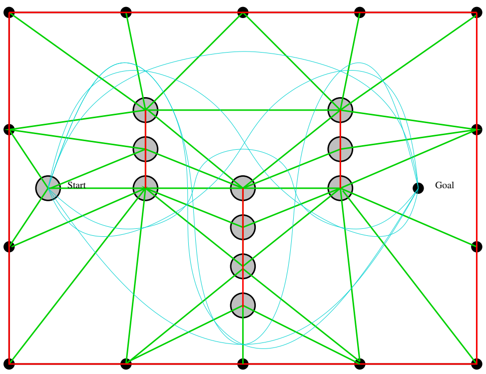

Fig. 6. **Illustration of Path Permutations**. Shown in light grey are splines for all the topologically different paths possible for this environment.

Consider a curve of the form:

$$P(t) = \begin{pmatrix} P_x(t) \\ P_y(t) \end{pmatrix} = \begin{pmatrix} a_x \\ a_y \end{pmatrix} t^3 + \begin{pmatrix} b_x \\ b_y \end{pmatrix} t^2 + \begin{pmatrix} c_x \\ c_y \end{pmatrix} t + \begin{pmatrix} d_x \\ d_y \end{pmatrix} \quad (1)$$

which we write in shorthand form as:

$$\mathbf{P}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d} \quad (2)$$

We can calculate the equations for velocity and acceleration at any point through differentiation:

$$\mathbf{P}'(t) = 3\mathbf{a}t^2 + 2\mathbf{b}t + \mathbf{c} \quad (3)$$

$$\mathbf{P}''(t) = 6\mathbf{a}t + 2\mathbf{b} \quad (4)$$

We assume the following initial and final conditions:

$$\begin{aligned} \mathbf{P}(0) &= \begin{pmatrix} P_{x_0} \\ P_{y_0} \end{pmatrix} \\ \mathbf{P}(s) &= \begin{pmatrix} P_{x_s} \\ P_{y_s} \end{pmatrix} \\ \mathbf{P}'(0) &= \begin{pmatrix} V_{x_0} \\ V_{y_0} \end{pmatrix} \\ \mathbf{P}'(s) &= \begin{pmatrix} V_{x_s} \\ V_{y_s} \end{pmatrix} \end{aligned} \quad (5)$$

Solving for $t = 0$:

$$\mathbf{P}(0) = \mathbf{d} \quad (6)$$

$$\mathbf{P}'(0) = \mathbf{c} \quad (7)$$

Solving for $t = s$:

$$\mathbf{P}(s) = \mathbf{a}s^3 + \mathbf{b}s^2 + \mathbf{c}s + \mathbf{d} \quad (8)$$

$$\mathbf{P}'(s) = 3\mathbf{a}s^2 + 2\mathbf{b}s + \mathbf{c} \quad (9)$$

These equations can be combined in matrix form as follows:

$$\begin{pmatrix} \mathbf{P}(0)\, \mathbf{P}(s)\, \mathbf{P}'(0)\, \mathbf{P}'(s) \end{pmatrix} = \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{pmatrix} \begin{pmatrix} 0 & s^3 & 0 & 3s^3 \\ 0 & s^2 & 0 & 2s \\ 0 & s & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad (10)$$

Equation (2) then becomes:

$$\begin{aligned} \mathbf{P}(t) &= \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix} && (11) \\ &= \begin{pmatrix} \mathbf{P}(0)\, \mathbf{P}(s)\, \mathbf{P}'(0)\, \mathbf{P}'(s) \end{pmatrix} \begin{pmatrix} 0 & s^3 & 0 & 3s^3 \\ 0 & s^2 & 0 & 2s \\ 0 & s & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix} && (12) \\ &= \begin{pmatrix} \mathbf{P}(0)\, \mathbf{P}(s)\, \mathbf{P}'(0)\, \mathbf{P}'(s) \end{pmatrix} \begin{pmatrix} \frac{2}{s^3} & \frac{-3}{s^2} & 0 & 1 \\ \frac{-2}{s^3} & \frac{3}{s^2} & 0 & 0 \\ \frac{1}{s^2} & \frac{-2}{s} & 1 & 0 \\ \frac{1}{s^2} & \frac{-1}{s} & 0 & 0 \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix} && (13) \end{aligned}$$

These equations uniquely determine the trajectory, once the time s for traversing the entire path segment has been chosen. For curve fitting applications, it is common to arbitrarily choose $s = 1$ as it simplifies the equation. However, for our trajectory planning task, we would like to choose the minimal s satisfying the following dynamic constraint (which is a unit circle in velocity and acceleration space):

$$|\frac{\mathbf{P}''(t)}{A} + \frac{\mathbf{P}'(t)}{V}|^2 \leq 1, \qquad \text{for } 0 \leq t \leq s, \tag{14}$$

where A and V are the maximal acceleration and velocity of the robot. From Equations (3) and (4) this becomes

$$1 \leq \left| \frac{1}{AV} \left(\mathbf{P}(0)\, \mathbf{P}(s)\, \mathbf{P}'(0)\, \mathbf{P}'(s) \right) \frac{1}{s^3} \times \begin{pmatrix} 2 & -3s & 0 & s^3 \\ -2 & 3s & 0 & 0 \\ s & -2s^2 & s^3 & 0 \\ s & -s^2 & 0 & 0 \end{pmatrix} \left[V \begin{pmatrix} 6t \\ 2 \\ 0 \\ 0 \end{pmatrix} + A \begin{pmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{pmatrix} \right] \right|^2 \tag{15}$$

Ideally, we would like to find the smallest (positive) value of s such that the vector on the RHS of Equation (15) does not exceed unit length for any t between 0 and s. Unfortunately, since the (vector-valued) function is quadratic in t and quartic in s, this problem is, in general, analytically intractable. However, provided that the lengths of the individual segments are sufficiently small relative to the smoothness of the curve, we can get a reasonable approximation by imposing the dynamic constraint only at the endpoints. (In particular, if we were to ignore friction by taking the limit as $V \to \infty$, the vector-valued expression in Equation (4) would become a linear function of t, ensuring that it must achieve its maximum length at one of the endpoints.) At the initial point, Equation (15) becomes

$$\begin{aligned} 1 \leq & \; |\tfrac{P''(0)}{A} + \tfrac{P'(0)}{V}|^2 \\ = & \left| \tfrac{2}{AVs^2} \begin{pmatrix} P_{x_0} & P_{x_s} & V_{x_0} & V_{x_s} \\ P_{y_0} & P_{y_s} & V_{y_0} & V_{y_s} \end{pmatrix} \begin{pmatrix} -3V \\ 3V \\ -2Vs + \frac{A}{2}s^2 \\ -Vs \end{pmatrix} \right|^2 \\ = & \; \tfrac{4}{A^2V^2s^4} \left| \tfrac{A}{2} \begin{pmatrix} V_{x_0} \\ V_{y_0} \end{pmatrix} s^2 \times \right. \\ & \left. -V \begin{pmatrix} 2V_{x_0} + V_{x_s} \\ 2V_{y_0} + V_{y_s} \end{pmatrix} s + 3V \begin{pmatrix} P_{x_s} - P_{x_0} \\ P_{y_s} - P_{y_0} \end{pmatrix} \right|^2 \end{aligned} \tag{16}$$

Thus:

$$\begin{aligned} \frac{A^2V^2s^4}{4} \leq & \; \tfrac{A^2}{4} \begin{pmatrix} V_{x_0} \\ V_{y_0} \end{pmatrix}^2 s^4 - AV \begin{pmatrix} V_{x_0} \\ V_{y_0} \end{pmatrix} \cdot \begin{pmatrix} 2V_{x_0} + V_{x_s} \\ 2V_{y_0} + V_{y_s} \end{pmatrix} s^3 \\ & + 3AV \begin{pmatrix} V_{x_0} \\ V_{y_0} \end{pmatrix} \cdot \begin{pmatrix} P_{x_s} - P_{x_0} \\ P_{y_s} - P_{y_0} \end{pmatrix} s^2 \\ & + V^2 \begin{pmatrix} 2V_{x_0} + V_{x_s} \\ 2V_{y_0} + V_{y_s} \end{pmatrix}^2 s^2 \\ & - 6V^2 \begin{pmatrix} 2V_{x_0} + V_{x_s} \\ 2V_{y_0} + V_{y_s} \end{pmatrix} \cdot \begin{pmatrix} P_{x_s} - P_{x_0} \\ P_{y_s} - P_{y_0} \end{pmatrix} \\ & + 9V^2 \begin{pmatrix} P_{x_s} - P_{x_0} \\ P_{y_s} - P_{y_0} \end{pmatrix}^2 \end{aligned} \tag{17}$$

The kinematic constraint at the final point leads to an inequality of the same form, but with the roles of (x_0, y_0) and (x_s, y_s) reversed.

The appropriate value of s will therefore occur at a root of one of these two quartic equations. It is easy to see that each of these equations must have at least one positive root. Thus the desired value for s can be obtained by checking the positive roots in increasing order, until one is found to satisfy both constraints.

This value for s will be the minimum time the robot can traverse the curve given the constraint in Equation (14). when s is substituted back into Equation (13) we have a unique representation for this trajectory. Additionally, by using Equations (2), (3), and (4), we can calculate the position, velocity, and acceleration at any time t along the curve, where $0 \leq t \leq s$.

B. Cubic Spline Trajectory

A PC spline is a chain of PC curves where the final position and tangent vector of one curve (say k), is the initial position and tangent vector for the next $(k+1)$. The textbook Hermite spline is immediately solvable since all control points (way-points) are constrained to an exact location [10]. For our purposes this overly constrains the trajectory since we have variable way-points that can be tuned to optimize the trajectory. Smoothing the trajectory allows the robot to maintain higher velocities for longer periods of time. For this reason we developed a new algorithm for smoothing the spline, shown below. It is essentially a gradient descent method across the four dimensional space of position and velocity at each way-point.

```
Initialise all variable way-points (k) to the position used
in the A* search and a velocity of the maximum speed
(V) the robot can travel at, in the direction from
point k − 1 to point k + 1.
do
  for each way-point ‘k’ (not including the end points)
    Generate a Parametric Cubic (PC) curve from
    points k − 1 and k + 1. (Sec. II-A)
    Find the intersection of this curve with the
    variable way-point line at k
    Set the way-point at k to the position and velocity
    values generated at the intersection.
    if either position or velocity violate a constraint
    then
      bring it back within the constraints
while changes are larger than error threshold or until
time-out
```

Although the trajectories produced are not guaranteed to be optimal, as they could fall into a local minima, they are guaranteed to be faster, and a more accurate representation, then a path modelled using straight line segments. An example of when this is sub-optimal is when the maximal velocity and acceleration is sufficiently high that instantaneous changes in direction (kinks in the path) are possible. In this case a globally smooth curve is not optimal for time.

III. Implementation Details

The following assumptions and simplifications were made for implementation:

1) All dynamic objects are treated as circles. The centre of the circles are the vertices of the graph. The radii affect the configuration space of the robot.
2) Walls are represented as point sources at regular intervals along the wall. The separation between points is approximately three times the size of the robot. Delaunay arcs connecting points on the same wall are pruned from the graph prior to searching.
3) The implementation of the graph allows four degree polygons to exist if the conversion to two triangles would result a very thin triangle. This is required since very thin triangles cannot be traversed in all directions.
4) The curves were further constrained to remove loops. This was achieved by over-riding the magnitude of the final velocity so that it does not need a "run up".

IV. Experimental Results

The algorithm presented was written in C and run under Linux. It was developed for use in the robotic soccer competition: RoboCup Small-Sized Robot league. The environment used is approximately three by four meters. It is surrounded by walls and contains ten robots of 0.18 meters diameter moving at speeds up to $2.5ms^{-1}$ and accelerations of approximately $2.0ms^{-2}$. A single camera is mounted $3m$ above the field with an unobstructed view of the environment [15].

Experiments were performed in simulation using a Pentium III 866MHz computer, and on real robots using a dual processor 1800 Athlon MP computer. The faster computer was required for the real experiments due to the increase in computation required for full frame-rate image processing. The robots are custom built small, omni-directional drive robots equipped with radio transceivers to communicate with the central computer [14]. Their maximum velocity was $1.5ms^{-1}$ and maximum acceleration was $2.0ms^{-2}$.

We compare the algorithm with our path planner previously used in RoboCup competitions, an A* search using straight line distance cost.

The cases we examined were finding a path/trajectory in:

1) trivial environments where a straight line path is possible
2) an environment with the zig-zag layout shown in Fig. 3
3) cluttered environments where an indirect path is required

The results shown in the graph (Fig. 8) are from real-robot experiments and show the average time taken to complete a path. Once the robot's destination point was reached, the original staring point was set as a new destination point and the process was repeated. The robot continued until 20 laps were completed and the time taken was averaged. For the easier environments the SLD algorithm results in the same path and trajectory as the cubic spline algorithm. However, for the randomly cluttered environment, the SLD choice varied greatly, sometimes taking almost twice as long as the cubic spline trajectory method. In some instances it took less time, but collided with some obstacles.

In the experiments the average computation time for calculating trajectories was low. Building the Delaunay graph took approx. $2.9ms$ on the 866MHz PC and $2.4ms$ on the dual

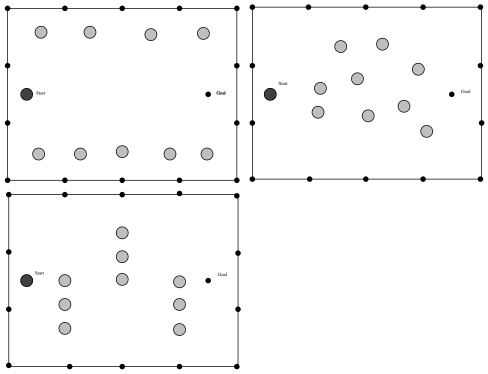

Fig. 7. **Illustration of experimental layouts**

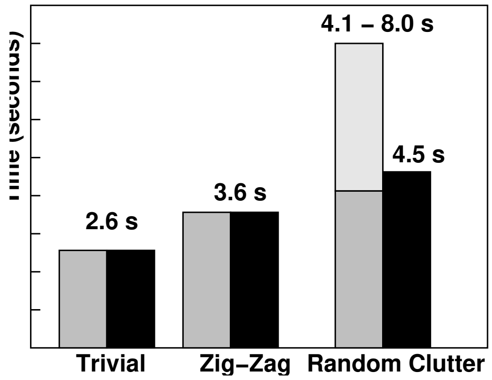

Fig. 8. **Experimental Results**

1800MHz PC. Plotting a trajectory based on the graph took approx. $1.4ms$ on the 866 MHz PC and $0.4ms$ on the dual 1800MHz PC. We consider the graph building independent of the trajectory planning as it is done once, and used by all robots.

These results show that this algorithm is efficient enough to be run at frame-rate. This allows us to plan trajectories in a dynamic environment.

V. Future Work

We intend to extend this algorithm to enhance coordination between multiple robots. There are three methods in particular:

1) Simultaneously search for all robots to ensure multiple robots are not planning to occupy the same space at the same time.
2) predict the movement of dynamic obstacles beyond our control, and plan around predicted positions at critical times.
3) Extend the set of equations to consider orientation and rotational velocity of the robot. (This is currently ignored.)
4) Trial the algorithm on a much larger scale.

VI. Conclusion

In this paper we presented an algorithm to find the time-optimal dynamic trajectory for mobile robots. Our system represents the environment as a Delaunay triangulation. We use an A* search to find all possible shortest distance paths. We then convert each complete path to a more accurate representation of an actual path using a PC spline consisting of PC curve segments. From this we can calculate an accurate estimate of the time required for each path. We continue to convert shortest distance paths to shortest time paths until we reach a path length that cannot possibly be better then our shortest time path given the know velocity and acceleration limits of the robot.

Due to the efficiency of this algorithm, it can be run at the rate sensor data is collected and was tested at a frame rate of 25Hz. This allows it to handle dynamic obstacles.

Experimental results show that in most cases this algorithm can plot time-optimal trajectories for multiple robots in real time. When the velocity and acceleration constraints permitted "kinked" paths, it occasionally produced sub-optimal paths.

ACKNOWLEDGMENT

The authors would like to thank the AECM for providing funding associated with this project.

REFERENCES

[1] Ronald C. Arkin, *Motor Schema Based Mobile Robot Navigation*. In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), 1987, pp 264-271.

[2] Johann Borenstein and Yoran Koren, *The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots*. IEEE Transactions on Robotics and Automation, 1991.

[3] Dieter Fox et al., *The Dynamic Window Approach to Collision Avoidance*. IEEE Robotics and Automation Magazine, 1997.

[4] Alberto Elfes, *Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception*. Proceedings of the Sixth Conference on Uncertainty in AI, July, 1990.

[5] Osamu Takahashi and R. J. Schilling, *Motion Planning in a Plane Using Generalized Voronoi Diagrams*. In Journal of the IEEE Transactions on Robotics and Automation, Vol. 5 No 2., April 1989.

[6] Vachirasuk Setalaphruk, Takashi Uneno, Yasuyuki Kono, Masatsugu Kidode, *Topological Map Generation from Simplified Map for Mobile Robot Navigation*. In Proc. of The 16th Annual Conference of Japanese Society for Artificial Intelligence, 2002.

[7] V. F. Munoz and A. Ollero, *Smooth Trajectory Planning Method for Mobile Robots*. Special issue on Intelligent Autonomous Vehicles of the Journal of Integrated Computed-Aided Engineering, Vol 6 No. 4, IOS Press, Netherlands, 1999.

[8] Cyrill Stachniss and Wolfram Burgard, *An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments*. In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2002.

[9] J.-C Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[10] Bruce R. Dewey, *Computer Graphics for Engineers*. Harper and Row, Publishers, 1988.

[11] A. Zelinsky, *A Mobile Robot Exploration Algorithm*, In Journal of the IEEE Transactions on Robotics and Automation, Vol 8. No. 6, pp 707-717, December 1992.

[12] D. T. Lee and R. L. Drysdale III, *Generalized Voronoi diagrams in the plain*, SIAM J. Comput., Vol 10. No. 1, pp 73-87, Feb. 1981.

[13] O. Khatib, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*, The International Journal of Robotics Research, Vol. 5 No. 1, pp 90-99, 1986.

[14] J. Thomas, A. Peel: Roobots Team Description In: RoboCup-2002: Robot Soccer World Cup VI Lecture Notes in Computer Science, Springer, in press

[15] http://www.itee.uq.edu.au/w̃yeth/F180 Rules/index.htm

[16] Kai-Ming Tse; Chi-Hsu Wang, *Evolutionary optimization of cubic polynomial joint trajectories for industrial robots*, IEEE International Conference on Systems, Man, and Cybernetics, Volume: 4, pp 3272 -3276, Oct 1998

[17] Cao, B.; Dodds, G.I.; Irwin, G.W., *Constrained time-efficient and smooth cubic spline trajectory generation for industrial robots*, IEE Proceedings Control Theory and Applications, Volume: 144 Issue: 5, pp 467 -475 , Sep 1997