

Hierarchical Multi-Agent Motion Planning in a Static Environment¹

Edgar Johnson, II

Freshman, Dept. of Computer Science and NASA ACE
North Carolina A&T State University, Greensboro, NC 27411
ej980406@ncat.edu
Dr. Albert Esterline, advisor

Abstract

We address the problem of motion planning for groups of mobile agents in an environment of static obstacles. We assume that the obstacles' start-to-destination paths are predetermined for the individual agents. We also make several simplifying assumptions about the priority of each agent and groups of agents and about the coherence of each group as it moves. This allows each group to be replaced by a bounding rectangle when paths are modified to avoid collisions. This simplification greatly reduces the complexity of multi-agent path planning.

1. Introduction

This paper reports on undergraduate research being conducted as part of the NASA-supported project "Motion Planning in a Society of Intelligent Mobile Agents." We think of agents as mobile robots. Our problem is to determine safe (no collision) paths [Fu91] of travel of groups of agents under several simplifying yet reasonable assumptions. We currently address static environments (all objects are stationary) but intend to move on to dynamic environments. For the background in computational geometry, see [PS85, Se92].

Our assumptions ensure that each group of agents can be enclosed within a rectangular "envelope" that moves in a straight line at constant speed. Perhaps the biggest challenge here is determining such envelopes. This approach greatly reduces the complexity of multi-agent motion planning since each group of agents is represented by a single rectangle.

2. Setting, Assumptions, and Approach

We assume that agents have priorities within a group and that groups themselves are ranked by priority. We also assume that the agents start and end relatively close to one another, that they move at approximately the same speed, with constant velocity, and, to prevent deadlock, that our environment is sparse. With our assumptions, a group of agents moves along a rectangular area, which we call a *channel*. We define our channel as having seven parts (see Figure 2.1): major axis (MA), goal side (GS), start side (SS), long side 1 (LS1), long side 2 (LS2), upper extreme (UE) and lower extreme (LE). MA is the greatest distance from any two points in the visibility graph. GS is a line that intersects with an endpoint of MA, parallel with UE and LE and of a length equal to the sum of the lengths of LE and UE. SS is similar to GS except SS intersects with the agent furthest from any destination point. GS and SS are parallel and perpendicular to LS1 and LS2. UE and LE are lines perpendicular to MA and their endpoints are the vertices furthest from MA (above and below). GS is relatively close to the destinations while SS is relatively close to the start.

¹ This research was supported by a grant NAG 2-1150, "Motion Planning in a Society of Intelligent Mobile Agents," from NASA Ames Research Center.

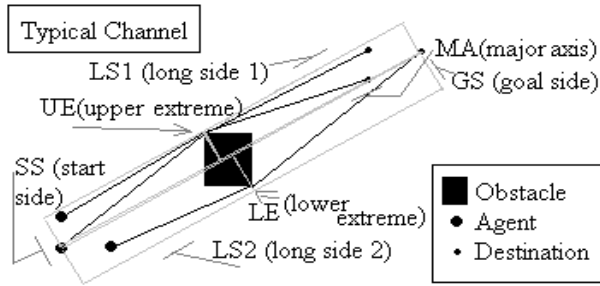


Figure 2.1

to avoid collisions between agents of different groups is drastically reduced.

Our assumptions ensure that the agents in a group are always bounded by a rectangle that lies within its channel. We call such a rectangle an *envelope*. The goal is to come up with an envelope for each group that moves with constant speed within that groups channel and whose width is as small as possible. With each group represented by its envelope, the complexity of planning

3. Avoiding Collisions within a Group

We generally delay a lower priority agent (before it moves) enough to prevent possible collisions with higher prioritized agents. We first assume that the shortest path for each agent from its start to its destination point is known. In our algorithm, the shortest path of each agent is recorded one at a time from highest priority to the lowest priority. If both agents arrive at the same point at the same time, then this point is a possible collision point between two agents. Now, to avoid collision, the highest priority agent is given a delay of zero, and each agent ranked in priority below her is delayed just long enough for each agent ranked in priority above the agent in question to pass the point of collision.

4. Finding the Channel for a Group

The endpoints of MA are the two path vertices that are the greatest distance apart. We find the points on both sides furthest from the MA. The perpendicular segments from these vertices to MA are known as the upper extreme (UE) and lower extreme (LE). We find UE and LE by drawing segments perpendicular from MA to each point above and below MA, respectively; the greatest distance once again is found and recorded. Next, we take the slope and length of MA and create the long sides (LS1 and LS2) of the channel with LS1 intersecting the endpoint of LE and LS2 intersecting the endpoint of UE. Connecting the corresponding ends of the two new lines creates the start side (SS) and goal side (GS) of the rectangular channel. This is a tight encapsulation of the mobile agents. The process is referred to as MAPTE (multi-agent process for tight encapsulation) and may apply to an arbitrary set of points.

The correctness of MAPTE is due to the fact that all points are between the endpoints of MA, creating a bound on one dimension of the points. Likewise UE and LE provide a bound in the second dimension for points above and below this line (to the right and left if the MA is perpendicular to the horizontal axis).

To see that this is correct, suppose that there is a path vertex lying beyond one of the short sides of the rectangle. Without loss of generality, assume that this vertex lies beyond the side GS. In Figure 4.1, the endpoints of MA are labeled E_1 (at SS) and E_2 (GS); the point beyond GS is labeled B. Now, the segment $\overline{E_1B}$ intersects GS at point C. Now, the distance E_1B equals $E_1C + CB$. Since E_1CE_2 is a right triangle with hypotenuse E_1C , E_1C is greater than E_1E_2 . Therefore, since CB is greater than 0, E_1B is greater than

E_1E_2 . Thus, E_1B is the greatest distance between any pair of vertices, contradicting the fact that E_2 and E_1 are the vertices the greatest distance apart.

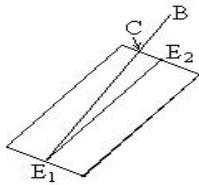


Figure 4.1

Figure 4.2 gives pseudo code for finding MA for a set of arbitrary points and for finding the upper and lower extreme points. We discovered that the algorithm is clearer if at this stage we rotate and translate the points so that MA lies on the x-axis of a Cartesian

plane and the endpoint to the left lies at the origin. Our equations for rotation are:

$$x' = x \cos\phi + y \sin\phi$$

$$y' = -x \sin\phi + y \cos\phi$$

ϕ is equal to $\tan^{-1}(-1/\text{slope of the MA})$ and x and y are the given points while x' and y' are the rotated points. By doing this we rule out the possibility that the MA is vertical hence we can always refer to points in relationship to above and below the MA.

Figure 4.2 is sample pseudo code for finding MA, UE and LE. This code has two variations in calculating distance. The maximum distance is found first by using the distance formula on all points in the set. The `ldistance` function returns the perpendicular distance between a point and a line segment. (It is assumed that the point is on a line perpendicular to the line segment.) `end1_MA` and `end2_MA` are the two endpoints of MA. V is the set of path vertices.

Figure 4.2 Pseudo code for finding a major axis (MA) and the relative upper and lower extreme points.

Major Axis (MA)

```
max_distance := 0.0;
for each pair  $v_1, v_2$  of vertices in  $V$  do
  if distance( $v_1, v_2$ ) > max_distance then
    begin
      max_distance := distance( $v_1, v_2$ );
      (end1_MA, end2_MA) := ( $v_1, v_2$ )
    end
end
```

Upper and Lower Extreme Points

```
max_upper_dist := 0;
max_lower_dist := 0;
for each vertex  $v$  in  $V$  do
  if  $v$  is above MA then
    begin
      if ldistance( $v, MA$ ) > max_upper_dist then
        begin
          max_upper_dist := ldistance( $v, MA$ );
          upper_extreme :=  $v$ 
        end
      if  $v$  is below MA then
        begin
          max_lower_dist := ldistance( $v, MA$ );
          lower_extreme :=  $v$ 
        end
    end
end
```

```

        end
    end
else if ldistance(v,MA) > max_lower_distance then
    begin
        max_lower_distance := ldistance(v,MA);
        lower_extreme := v
    end
end

```

5. Finding the Envelope for a Group

The procedure to find the envelope involves two steps. In the first step, we determine a constant speed for the front edge so that it stays ahead – but not too far ahead – of all the agents. In the second step, given this constant speed of the envelope, we determine the width of the envelope, that is, how far behind the front edge the back edge must be so that it trails all of the agents but not by too much. For simplicity, we assume that each path vertex is on the path of exactly one agent and is reached by that agent exactly once. (Violations of this assumption can be handled by duplicating vertices.) Then, with each path vertex, we associate the unique time when it is reached by its agent. We define a *leading vertex* as a path vertex v such that, when v is reached by its agent, its agent is closer to GS than any other agent.

To find the speed of the front edge (the first step), we need consider only leading vertices. The front edge initially intersects the start point of the agent that starts closest to GS. Let x_0 be the abscissa of this point and let t_0 be the start time. The final position of the front edge is coincident with GS. Let x_f be this final abscissa and let t_f be the time when the last agent reaches its destination. The first estimate of the speed Sp of the front edge is $(x_f - x_0)/(t_f - t_0)$. With the leading vertices sorted by nonincreasing abscissa values, we check each in turn. Let x_i be the abscissa of the i^{th} leading vertex and let t_i be the time it is reached. If $(x_i - x_0)/(t_i - t_0) > Sp$, then we update Sp to $(x_i - x_0)/(t_i - t_0)$. If this would result in a change in Sp of more than 20%, then x_0 (the initial position of the front edge) is advanced by small increments until the change in Sp is no more than 20%. This is so that a large component of the velocity of an agent parallel to MA early on does not force Sp to be large throughout.

Given the speed of the front edge, that is, of the entire envelope, we next find the width of the envelope, that is, the (constant) distance d by which the back edge trails the front edge. Initially, the back edge coincides with SS, and d is $x_0 - x_{SS}$, where x_{SS} is the coordinate of SS. We then go through the leading vertices in nondecreasing order of abscissa values. We know the time t_i when the i^{th} leading vertex v_i is reached by its agent. We find the maximum difference d_i between the abscissa of v_i and the abscissae of the locations of all other agents at t_i . If $d_i > d$, then we update d to d_i .

Our pseudo code for finding the set of leading vertices is shown in Figure 5.1. We assume now that V is an array of all path vertices and that n agents are identified with the natural numbers $1, \dots, n$. NumV is an array such that, where ag is an agent number, $\text{NumV}[ag]$ is the number of vertices in ag 's path. ReachTime is a two-dimensional array whose row indices are agent numbers. The row for any agent contains a record for

each vertex in its path. The `vindex` field of such a record contains the index of the vertex in the `V` array. The `time` field contains the time the vertex is reached. The `find_prev` function, given `ReachTime`, an agent number `ag1`, and a time `t`, returns the index in the `ReachTime` row for `ag1` of the last vertex `ag1` reaches before time `t`. Fuction `x`, given a vertex index, returns the abscissa of that vertex.

Figure 5.1 *leading edge development towards a concise envelope*

```

for ag := 0 to Number of Agents do
  for ord := 0 to NumV[Ag]
    if leading(ag, x(ReachTime[ag][ord].time, ReachTime)
      then Add V[ReachTime[ag][ord].vindex] to leading
        vertices
  leading - takes an agent index, ag, an abscissa, x_ag and the Reach_Time array
for ag1 := 0 to Number of Agents do
  if ag1 ≠ ag then
  begin
    if ReachTime[ag1][NumV[ag1]].time ≤ t then
    begin
      if x_ag < x(ReachTime[ag1][NumV[ag1]].vindex) then
        return false
    end
    else
    begin
      prev_ord := find_prev(ReachTime, ag1, t);
      if ReachTime[ag1][prev_ord].time = t then
      begin
        if x_ag < x(ReachTime[ag1][prev_ord].vindex) then
          return false
        end
      else
      begin
        x1 := x(ReachTime[ag1][prev_ord].vindex);
        x2 := x(ReachTime[ag1][prev_ord + 1].vindex);
        t1 := ReachTime[ag1][prev_ord].time;
        t2 := ReachTime[ag1][prev_ord + 1].time;
        x_ag1 := x1 + (x2 - x1)(t-t1)/(t2-t1);
        if x_ag < x_ag1 then
          return false
        end
      end;
    return true
  end

```

6. Multiple Groups

When looking at problems involving multiple groups, we examine possible collisions between groups represented by their envelopes. We first consider the angles at which two envelopes collide. The edges of the envelopes that can touch each other are looked at for

further simplification of the collision points that need to be examined. In the case of two envelopes possibly colliding with one another, given that they are not perpendicular, there are two critical edges. If two envelopes are perpendicular to each other at the point of collision, then there is only one critical side for each envelope. These edges can be found by examining the angle and position of the two envelopes. A delay is prescribed to the lower prioritized group. The length of the delay is calculated from the length of the one critical side divided by the velocity of the respective envelope plus the length of the critical side of the other envelope divided by the same velocity. Also, the length of time for the smallest delay is when the angles of the two envelopes are perpendicular. This delay time is necessary for the non-delayed envelope to cross the collision area. Refer to Figure 6.1.

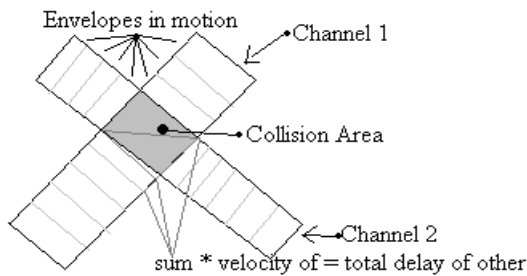


Figure 6.1: Delay times for channels of lower priority

When, on the other hand, the angle between the colliding edges is not perpendicular, then the lengths of these edges as well as the angle they form help to determine the size of the collision area. The distance of the point furthest from the entry point of the collision area added to the distance to the respective exit point is known as the furthest vertex sum (FVS) with respect to the collision area. All three points that make up the

FVS form a straight line parallel to an edge of the envelope also known as the line of sight. The amount of time for the higher prioritized envelope to cross the collision area completely is found by taking the FVS and dividing it by the velocity of the envelope.

7. Conclusion

We now have a clear method for encapsulating an arbitrary set of points. This allows a hierarchical approach to multi-agent motion planning that can greatly reduce the complexity of this task when the appropriate assumptions hold. Although we have assumed that our polygons are convex, we have proved that MAPTE works even in the absence of this assumption. Under our assumptions, our heuristics have proven sound.

There are several additional assumptions that we have found must be made to ensure the correctness of our approach. For example, no endpoint of a channel can lie in another channel; if so, deadlock may occur. We intend to study the requirements further to articulate all needed assumptions. Also, we intend to look for additional optimization possibilities. For example, rather than delaying an agent so that it avoids collision with a higher-priority agent, we could modify its path to go around the other agent.

References

- [Fu91] Fujimura, K., *Motion Planning in Dynamic Environments*. Tokyo: Springer-Verlag, 1991.
- [PS85] Preparata, F. and Shamos, M., *Computational Geometry: An Introduction*. New York: Springer-Verlag, 1985.
- [Se92] Sedgewick, Robert, *Algorithms in C*. Reading, MA: Addison-Wesley, 1992.