Real-Time Robot Path Planning

Allan Willms

Department of Mathematics and Statistics

University of Guelph

Simon Yang

School of Engineering

University of Guelph

Light fare for a late afternoon

- Robot path planning in dynamic environments
- A distance-propagating dynamic system
  Paper to appear in IEEE Transactions on Systems, Man and Cybernetics, Series B, Vol 36, June 2006.
- Different updating schemes
- Capturing targets
- Shunning obstacles
- Multiple robots and targets
- Sweeping the floor

Robot Path Planning

- Easiest application to visualize is an actual robot moving on a flat surface attempting to reach a certain moving target, when there are barriers in the room that are also moving.
- An example of a more realistic application is a robot arm hinged at two points so that the position of the tip is determined by two angles, θ and ϕ.
- The illustrations are 2D but the algorithm can be easily extended to higher dimensional systems.
- The illustrations all use regular square lattices, but any grid will work. All that is required is that each grid point have a predefined set of neighbours with known distances.

Distance-Propagating Dynamic System

Space is discretized to a grid, each point with a defined set of neighbours at known distances. Initially all points are given a value D, some maximal distance, except target locations which are set to 0.

$B_i =$ Neighbour set of point i.

$d_{ij} =$ Euclidian distance between neighbours i and j

$$x_i(0) = \begin{cases} 0, & \text{if } i \text{ is a target} \\ D, & \text{otherwise} \end{cases}$$

Distance-Propagating Dynamic System

Each subsequent time step,

$$x_i(n+1) = \begin{cases} 0, & \text{if } i \text{ is a target} \\ D, & \text{if } i \text{ is a barrier} \\ \min\left(D, f(i,n)\right), & \text{otherwise} \end{cases}$$

where

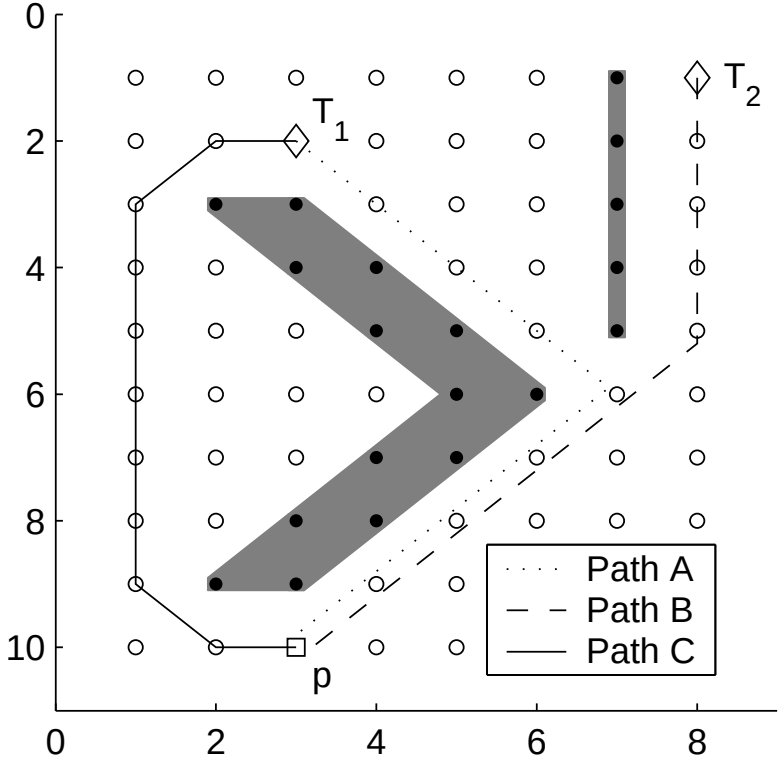
$$f(i,n) = \min_{j \in B_i} \left(d_{ij} + x_j(n)\right)$$

Targets or barriers moving to position i cause an immediate update of x_i to D or 0 respectively.

Distance vs. Neighbours

Unless all neighbour distances are equal across the whole grid, updating nearest neighbours means path information may not be optimal.

$$x_p(j) = D, \quad 0 \le j \le 7$$
$$x_p(8) = 8\sqrt{2}$$
$$x_p(9) = 4 + 5\sqrt{2}$$
$$x_p(j) = 8 + 2\sqrt{2}, \quad 10 \le j$$



Static Case

If barriers and targets do not move then the algorithm is a *dynamic programming* (DP)solution to the shortest path problem.

As written, the algorithm uses no global information when updating. Typical DP algorithms update the nodes in a specific order.

If we also keep track at each node the number of updates applied, then

$$\text{conv}_i = \left\lfloor \frac{x_i(n)}{d_{\min}} \right\rfloor - n$$

is the maximum number of time steps remaining until x_i converges to its minimum.

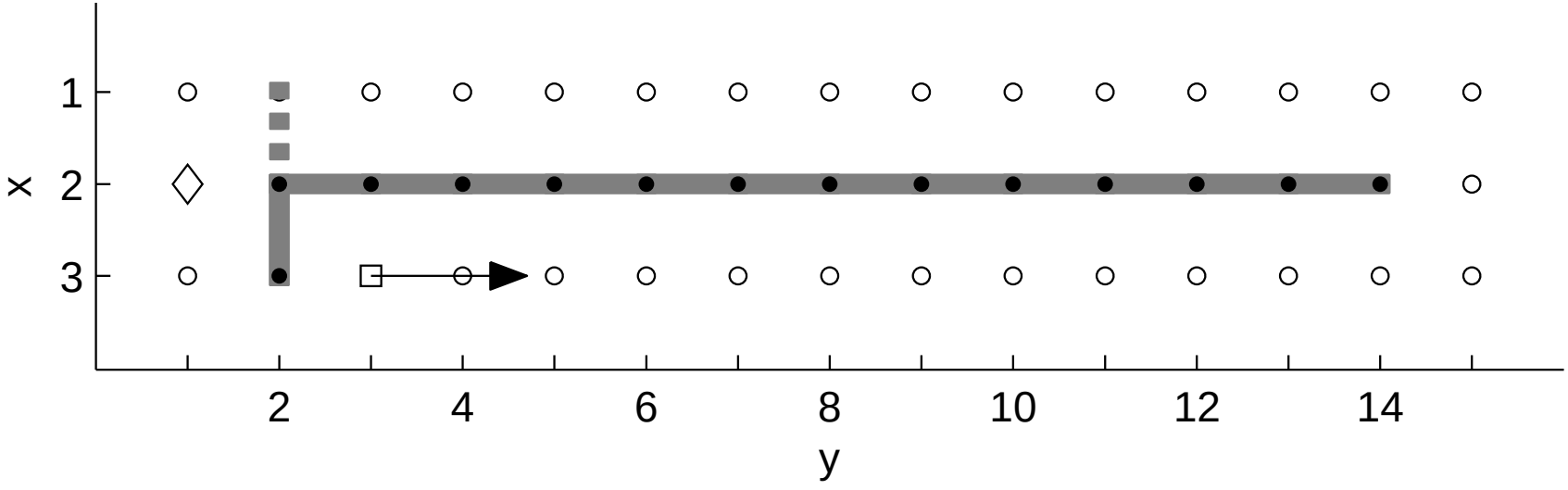
Alternative Implementations/Updating

Instead of computing $x_i(n+1)$ for every i from time step-n values:

- tell neighbours to update whenever I am updated
- pass distance to neighbours
- use “Gauss-Seidel” rather than “Jacobi”

No Guarantees when Barriers Move

If barriers are allowed to move, then, no matter how slow they move, how fast the robot moves, and how large the system update frequency is, it is always possible to construct environments where the target is never captured.



Sufficient Conditions for Capture

Suppose the robot arrives at i at some time t and $x_i(n) = f(i, n-1) < D$, where n is the largest time step for i prior to t. Then

$$x_j(n-1) = x_i(n) - d_{ij}, \qquad \text{for some neighbour } j \text{ of } i.$$

Let u be the system update frequency.

The maximum amount of time since i last received the value $f(i, n-1)$ from j is $\frac{1}{u}$, and the maximum amount of time prior to that since j received the value $f(j, n-2)$ from some neighbour k is $\frac{1}{u}$.

Sufficient Conditions for Capture

So the maximum amount of time during which x_j can have been updated from $f(j, n-2)$ when the robot arrives at j is

$$\frac{2}{u} + \frac{d_{ij}}{v_r}.$$

In this time the target can have travelled at most a distance

$$v_t \left(\frac{3}{u} + \frac{d_{ij}}{v_r} \right).$$

Even with infinitely fast propagation of that information to j, when the robot arrives at j, x_j will necessarily be smaller than $x_i(n)$ if d_{ij} is greater than this distance.

Sufficient Conditions for Capture

That is,

$$d_{ij} > v_t \left(\frac{3}{u} + \frac{d_{ij}}{v_r} \right).$$

Thus, for any point on the grid, if

$$u > \frac{3}{d_{\min} \left(\frac{1}{v_t} - \frac{1}{v_r} \right)}$$

then the robot will always capture a target.

Shunning Obstacles

An easy extension can be applied to not only prevent collision with obstacles, but also shun them.

- Define a penalty function, $P(d)$ which represents the extra distance you are willing the robot to travel to avoid being a distance d from a barrier at one point.
- Propagate two quantities:
  - distance to a barrier, and
  - the sum of the distance to a target and the cumulation of penalties at points along the path.

Multiple Robots and Targets

If there are multiple robots you also need a mechanism to avoid robots colliding with each other, and from uselessly chasing the same target.

- A local penalty function as for shunning barriers might work.
- A global coordination of robots ensuring they don't chase the same target. This would require the passing of a target identifier with the objective function.

Sweeping

If you wish the robot(s) to cover all points in the space, then simply defining them all to be targets provides one way to achieve this.

The way in which the algorithm chooses between equally optimal paths will impact on performance.

You might want to think about what better information other than simply distance to a target should be propagated to make for more efficient coverage.