Realistic Path-planning for Autonomous Vehicles

Elben Shira
Collaborators: Patrick Beeson, Peter H. Stone
Department of Computer Sciences

Abstract

• Autonomous vehicles require fast and reliable path-planning to ensure safety, accuracy, and efficiency when traversing zones (e.g. parking lots)
• Problems in classical algorithms
  • we look at A* and Generalized Adaptive A* (GAA*)
  • ignore the turning restrictions of a vehicle
  • create paths that a vehicle cannot traverse
• We extend existing these algorithms
  • consider vehicle turning restrictions and dynamics
  • well-defined and drivable paths
  • performance issues
    • improved heuristics from vehicle restrictions
• This work is done to improve Marvin's (Figure 1) zone path-planning algorithm, which currently requires a separate algorithm for maneuvers that require accurate planning (e.g. parking)

Figure 1. "Marvin", Austin Robot Technology's Autonomous Vehicle

Definitions

• The **world** is a multi-dimensional array of states
• A **state** is a cell in the world that holds the position and orientation of that particular cell
  • The **successors of a state** are states that the vehicle can move to from its current state
• The **cost** of an action allows quantification of actions a vehicle can take such as moving from one state to another
  • The **g-cost** of a state s, **g(s)**, is the cost of traveling from the start state to s.
• A **heuristic** is known information that can be used to better estimate the cost of an action
  • The **Euclidean distance** is the distance between two points
  • The **h-cost** of a state s, **h(s)**,is the heuristics between that state that the goal state
• A **bin** is a way of discretizing orientation; a set of degrees fall inside a bin.
• **R*** is the family of real-space path-planning algorithms that is based on A*.

Algorithm Overview

A* Overview

A* determines the shortest path between two states via two factors: distance traveled and a heuristic (e.g. Euclidean distance to goal state). It is similar to Dijkstra'

A* finds the shortest path by adding the states surrounding the current state into an open list (a priority queue). Each state inserted into the open list contains a pointer that points to the previous state (the state the vehicle came from). It then pops the state with the lowest f(s) = h(s) + g(s) value and expands that state by adding its successors, the eight surrounding states, into the open list.

Once the goal state is popped, then we have found the shortest path. We backtrack from the goal state to the start state (via the back pointers) to retrieve our path.

The heuristics function of A* is consistent. That is, the h-cost must always be less than or equal to the actual cost to get to the goal state:

$$g(s) + h(s) \leq g(s_{goal})$$

GAA* Overview

GAA* [1], an extension of A*, remembers the costs of previous searches and uses this data to improve the heuristics of future searches.

1. Run A* to find the shortest path
2. Save the cost of that path
3. On future searches, if a previous search has visited a cell, use the previous cost to improve the heuristics of current search

If we are in state s, we update the h-value of s:

$$h(s) := g(s_{goal}) - g(s)$$

This updated h-value is guaranteed to be consistent. That is, it does not overestimate the cost. If an obstacle, however, disappears from the world on the next time step, we must run a consistency procedure to make sure that for all state s, h(s) is consistent.

Extending A* and GAA*

• Figure 2a shows a path that A* or GAA* might make. Notice that it is impossible for a vehicle (shown in green) to execute this path

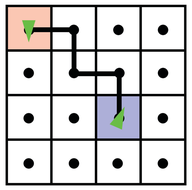

Figure 2a. Two-dimensional search.

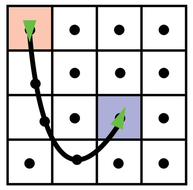

Figure 2b. R search.*

• All points that lands inside a particular state are discretized into that state's position. For example, if we are at (x, y) = (0.022, 0.544), we estimate this as state (0, 0), highlighted in red
  • This loss of information results in inaccurate planning
• To remedy these problems, we extend A* and GAA* into Real-space A* (RA*) and Real-space GAA* (RGAA*). This is similar to the work found in Stanford's autonomous vehicle [3][4].
• RA* and RGAA* improve path-planning by:
  • considering vehicle turning restrictions
  • keeping the vehicle's real-world position to prevent information loss [3]
  • creating drivable and efficient paths (Figure 2b)

Real-space A* and GAA*

We refer to the Real-space path-planning family of algorithms as **R***.

Continuous to Discrete

R* must convert the continuous world into a discrete world representation. R* define the vehicle's real-world position as the following triple:

$$(x_{real}, y_{real}, \theta_{real})$$

We keep track of the real-world position, but the search algorithms function in a discrete world (or else we would have an infinite number of states), so we convert the real-world values to:

$$(x, y, \theta) := (\lfloor x_{real} \rfloor, \lfloor y_{real} \rfloor, bin(\theta_{real}))$$

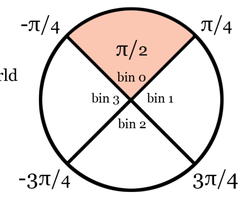

Figure 3. Four bins configuration.

Discretizing (x, y) is as simple as flooring (x, y) to the nearest integer. Discretizing orientation (theta), however, requires additional work.

Discretizing Orientation via Bins

A vehicle has, along with an (x, y) position, an orientation. Similar to (x, y), a vehicle can be in an infinite number of orientations, so discretization from the continuous world is needed.

We divide up 2π into cell-like objects we call "bins." In a configuration with 4 bins, as shown in Figure 3, each bin holds π/2 degrees. Bin 0 would contain degrees [-π/4, π/4), bin 1 would contain degrees [π/4, 3π/4), and so on. The equation is:

$$bin(\theta_{real}) = \left(round\left(\frac{\theta_{real}}{\beta}\right) + \alpha \right) \mod \alpha$$

$$\text{Where } \alpha = number\ of\ bins,\ \beta = \frac{2\pi}{\alpha} = radians\ per\ bin$$

Successor Function

R* finds its successors by "shooting out" arcs from the vehicles current position. Figure 4 shows five successors. The length of the arc is denoted as d, and the radius of curvature of the arc is denoted by r (Figure 5). Given a radius of curvature r and arc length d, we find theta by setting:

$$\theta := d/r$$

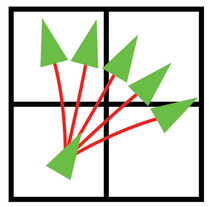

Figure 4. Five successors.

The successor of a state $(x_{real}, y_{real}, \theta_{real})$, given some r and t, is defined as:

$$\hat{x}_{real} := x_{real} - r \cdot \sin(\theta_{real}) + r \cdot \sin(\theta_{real} + \Delta\theta_{real})$$
$$\hat{y}_{real} := y_{real} - r \cdot \cos(\theta_{real} - r \cdot \cos(\theta_{real} + \Delta\theta_{real})$$
$$\hat{\theta}_{real} := \theta_{real} + \Delta\theta_{real}$$

Given these formulas, we can find several successors by passing in different radius of curvatures. The radius of curvature, not theta, is changed because the minimum radius of curvature the vehicle can make is known.

Other Considerations

Scaling. The scaling of the real-space might differ from that of the discrete-space. That is, a discrete state (0, 1, 0) might represent a real-space (300.3, 655.3, .002). Use real-space when in the finding successors. We then scale these values into discrete space.

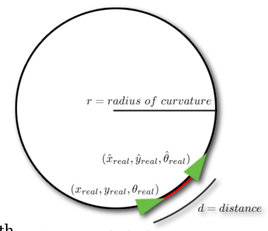

Figure 5. Vehicle dynamics.

Arc length. We find that sqrt(2) is the best arc length. This makes it possible, however, for a successor to jump over a touching state. This is legal, but it is possible that the successor jumps over the goal state, and thus we never find a plan. This is remedied by checking if our goal state is a touching state. If so, lower arc length and attempt to find a path that lands in the goal state.

Improved Heuristics

Extending a standard 2-d search algorithm like A* or GAA* into a 3-d search algorithm increases the number of states by a magnitude of b, where b is the number of bins in the system. This exponentially increases the number of possible paths and slows down path planning by an undesirable magnitude.

We remedy this problem by observing possible improvements in our heuristics. In fact, this is what GAA* observed about A*. Improvements in heuristics is possible due to the turning restrictions of a vehicle. Stanford suggested considering two heuristics and taking the best of the two [3][4].

The first heuristics, **h1(s)**, is the path cost from s to the goal state given that no obstacles exist. This can be calculated offline and implemented via a lookup table.

The second heuristics, **h2(s)**, is the path cost of a 2-d search of the world given the present obstacles. This is calculated real time. The number of possible states in a 2-d search (that is, do not consider turning restrictions) is much smaller than the number of possible states in a 3-d search (consider turning restrictions), so this approach is reasonable.

Given these two heuristics, we set:

$$h(s) := max(h1(s), h2(s))$$

Future Work

• Implement and analyze the improved heuristics against similar algorithms
• Implement R* into vehicle codebase and compare against current path-planning algorithm

Acknowledgements

This work is funded in part by the National Science Foundation and the Howard Hughes Medical Institute through the Freshman Research Initiative.

FRESHMAN RESEARCH INITIATIVE
COLLEGE OF NATURAL SCIENCES

References

[1] X. Sun, S. Koenig and W. Yeoh. Generalized Adaptive A* In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS),* 2008.

[2] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 2:100 -107, 1968.*

[3] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path Planning for Autonomous Driving in Unknown Environments. *In Proceedings of the Eleventh International Symposium on Experimental Robotics(ISER-08). July 2008.*

[4] M. Montemerlo, J. Becker, S. Thrun, et al. Junior: The Stanford Entry in the Urban Challenge. *In Journal of Field Robotics. 2008.*