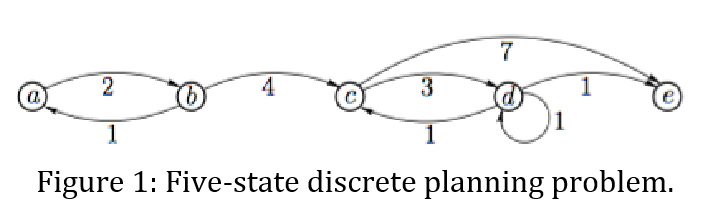
Andrew Smith

HW1 ROB-534 SDM



Backward value iteration:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | e |
| 1 | N | n | n | n | 0 |
| 2 | n | n | 7 | 1 | 0 |
| 3 | N | 11 | 4 | 1 | 0 |
| 4 | 12 | 8 | 4 | 1 | 0 |
| 5 | 10 | 8 | 4 | 1 | 0 |

Forward value iteration:

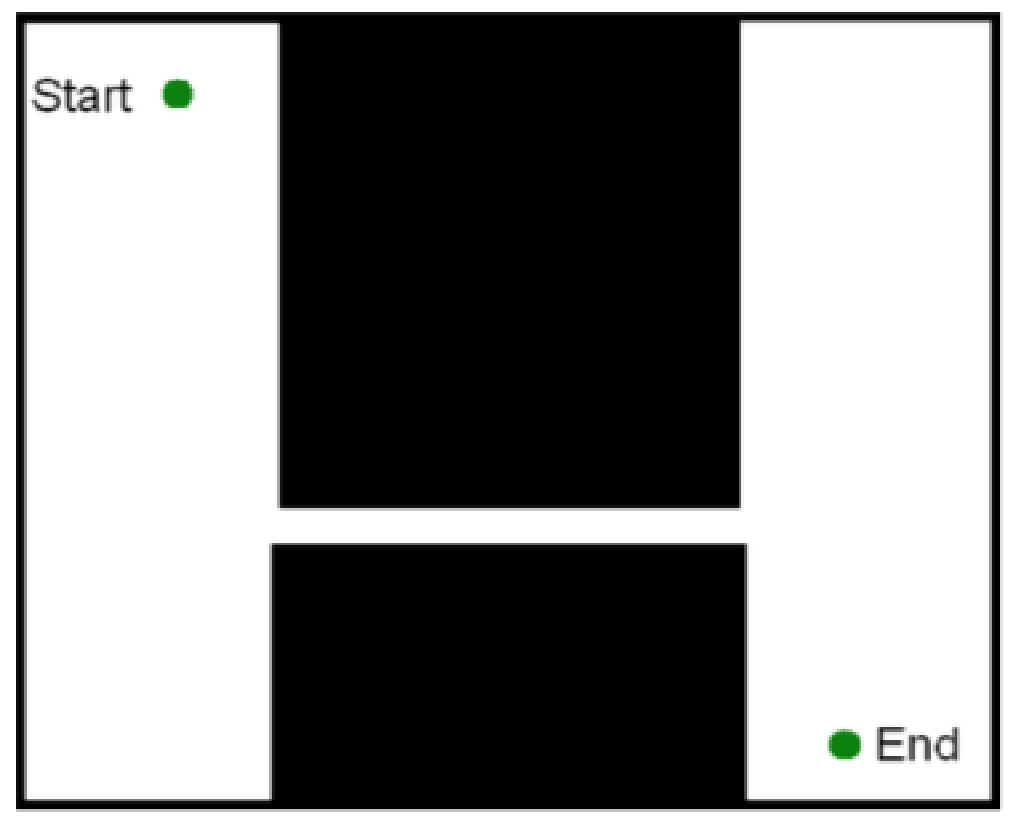
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | B | c | d | e |
| 1 | 0 | N | n | n | N |
| 2 | 0 | 2 | N | n | n |
| 3 | 0 | 2 | 6 | n | n |
| 4 | 0 | 2 | 6 | 9 | 13 |
| 5 | 0 | 2 | 6 | 9 | 10 |

B: 8-Puzzle Problem

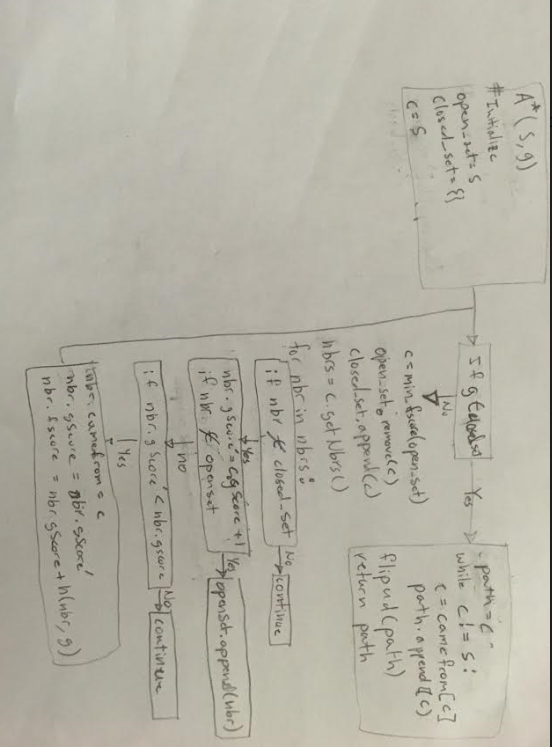
i,iii,,v are admissible – ii obviously over estimates in almost every case, iv overestimates when 1 tile is in the wrong place by a large amount

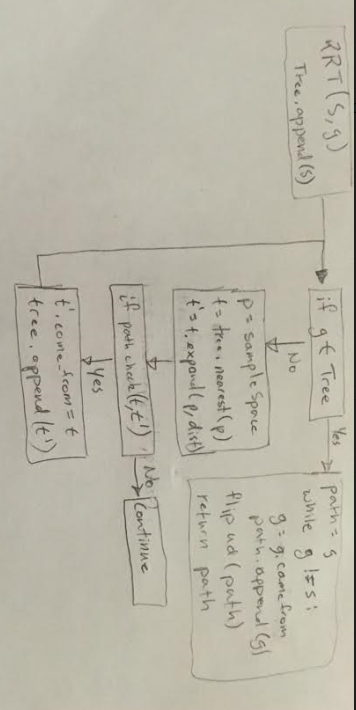
in order: iii,i,v

2-Configuration Space and RRT



The pictured environment may cause problems because it requires repeatedly sampling in the narrow passage to find a solution. As the passage is narrow and a small portion of the space, this will take a large number of samples (most of which do not contribute to the solution) to find a path that works. Using A\* would avoid this problem because it would widely search the open area and then upon finding the narrow passage would move across it quickly guided by the heuristic. Then it would widen the search in the open area while looking for the goal. The disadvantage of A\* would be the time wasted in the open area which RRT solves quickly. The advantage of A\* is how quickly it makes it through the narrow passage. As the image is small and 2D I would assume that A\* would be advantageous in memory an computation time as RRT would require a large number of samples to find a path.



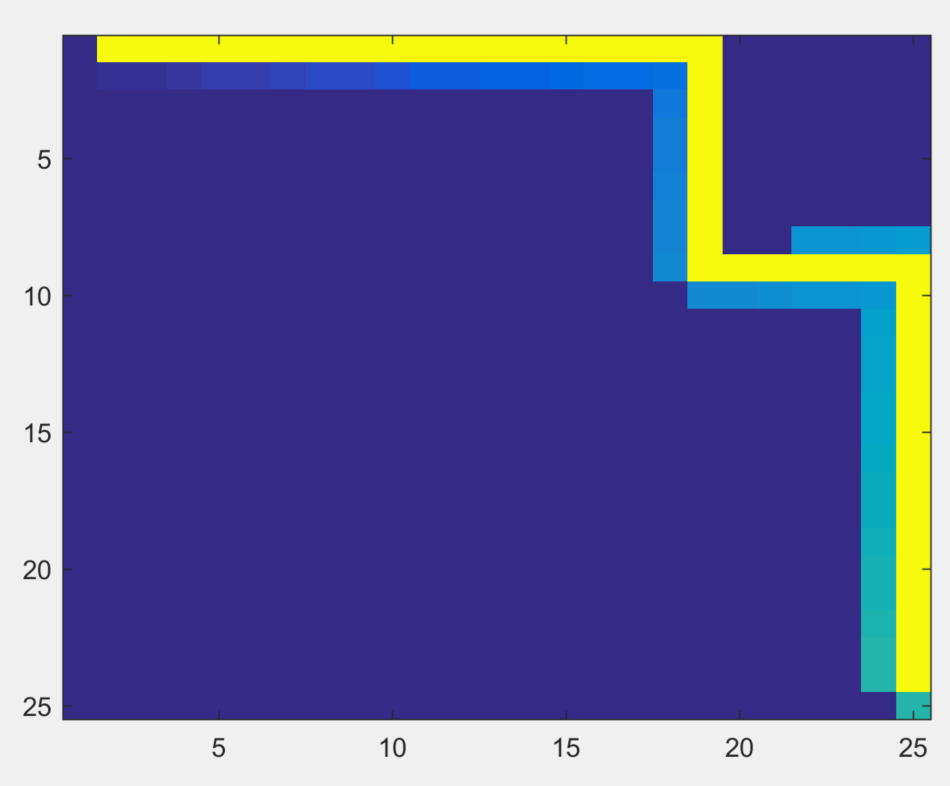


A\*

Heuristic = abs(sx-gx) + abs(sy-gy)

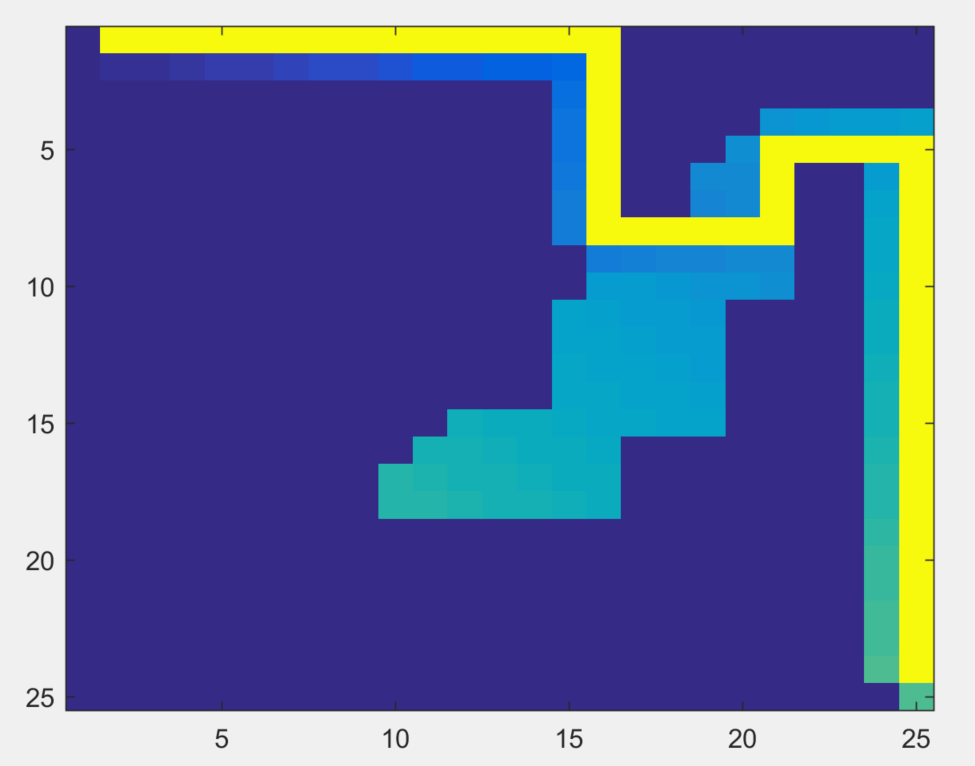
Maze1

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Final Epsilon | Length | Nodes |
| 1.0 | 1.001 | 48 | 49 |
| 0.25 | 1.001 | 48 | 49 |
| 0.05 | 3.25 | 48 | 49 |



Maze2

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Final Epsilon | Length | Nodes |
| 1.0 | 1.001 | 48 | 233 |
| 0.25 | 1.001 | 48 | 233 |
| 0.05 | 5.5 | 54 | 109 |

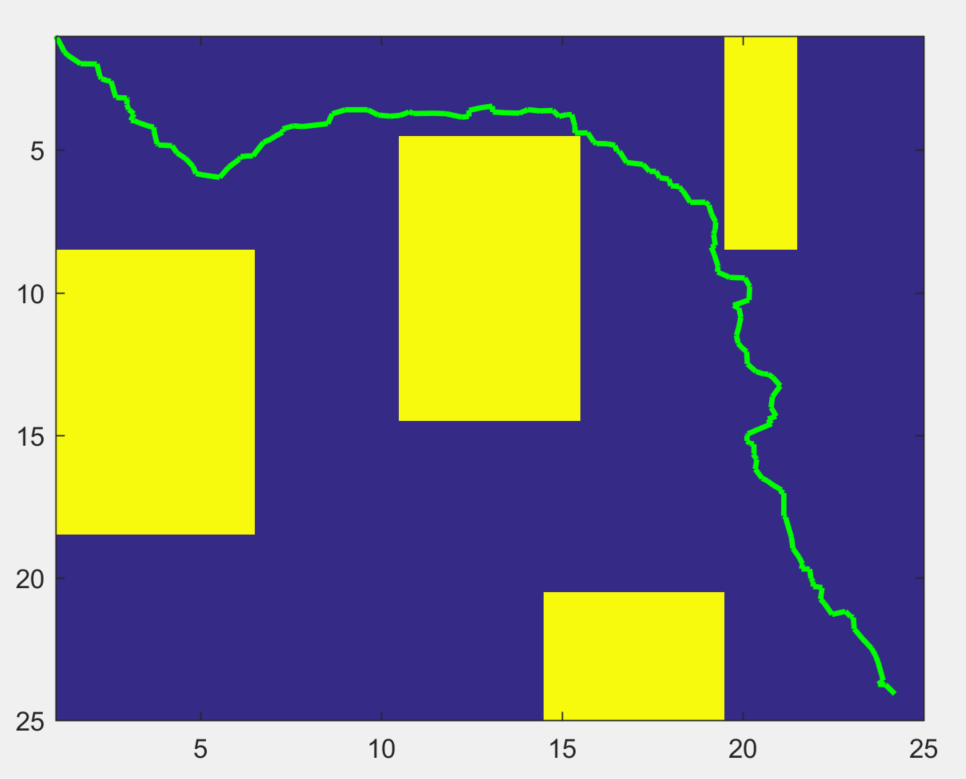


RRT map1

Length = 47.1534

Time = 0.8281

Nodes = 684

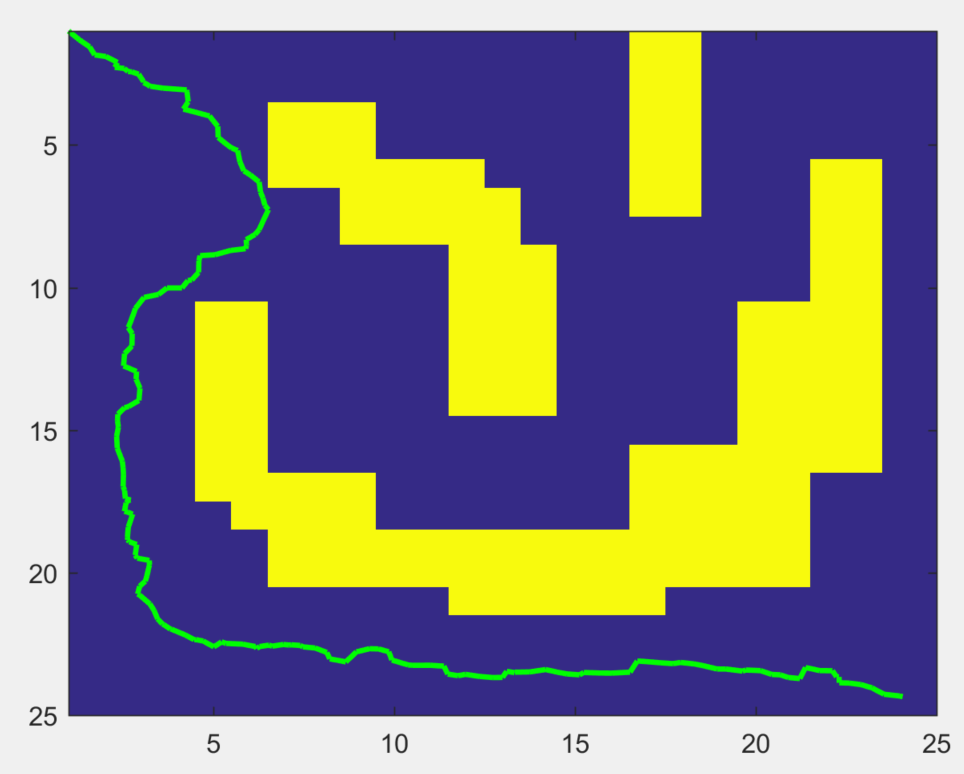


RRT map2

Length = 51.2658

Time = 5.370

Nodes = 2131

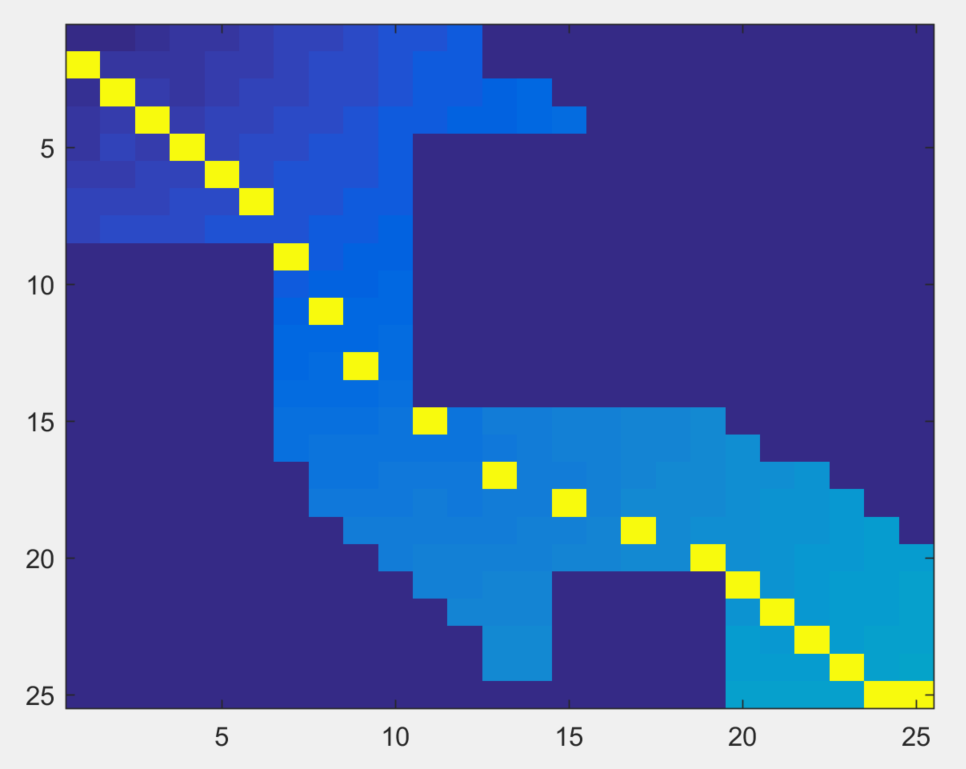


A\* 4D

Heuristic = sqrt((sx-gx)^2 + (sy-gy)^2 + (sdx-gdx)^2 + (sdy-gdy)^2)

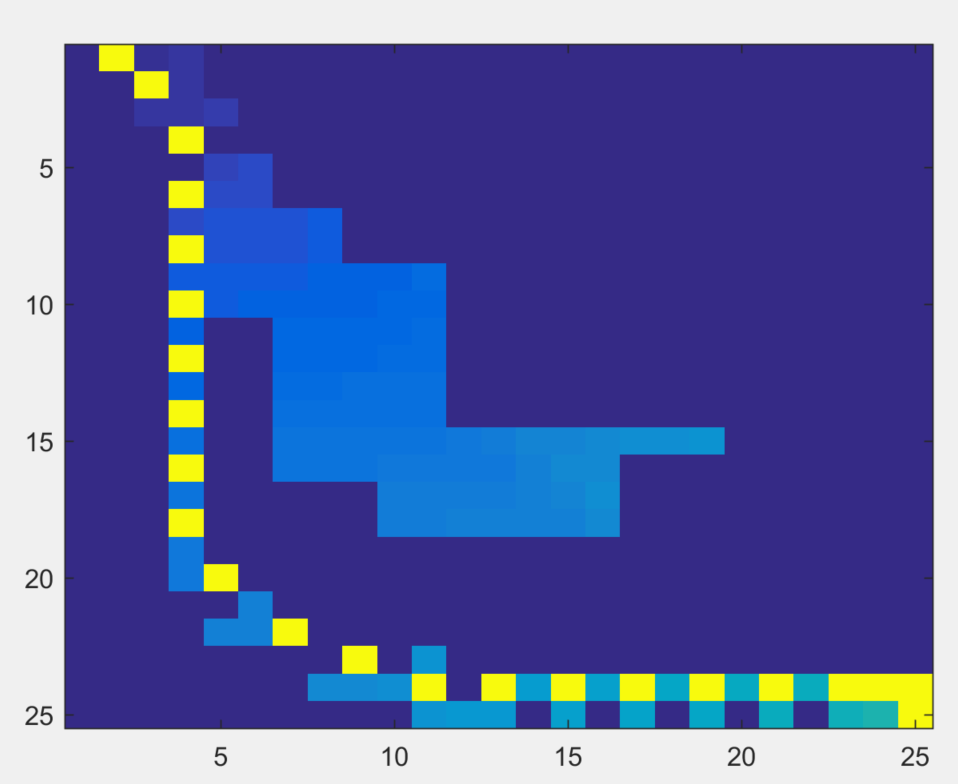
Maze1

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Final Epsilon | Length | Nodes |
| 1.0 | 1.001 | 21 | 1094 |
| 0.25 | 1.0352 | 21 | 910 |
| 0.05 | 10 | 21 | 30 |



Maze2

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Final Epsilon | Length | Nodes |
| 1.0 | 1.0176 | 27 | 1963 |
| 0.25 | 2.1250 | 24 | 528 |
| 0.05 | 10 | 25 | 525 |



Comparison

1 – As the heuristic decreases towards 1 in an environment with local minimums the number of nodes expanded increases and the path quality improves. This intuitively makes sense because you are less greedily searching for the goal and instead focusing on improving path quality. Improvement in path quality requires expanding more nodes, which is time intensive, while greedily heading towards the goal requires moving in the general direction of the goal, faster but lower path quality.

Selecting epsilon requires balancing the importance of path quality and computation time as well as prior knowledge of environment complexity. In a simple environment with low planning costs that requires only a path, not an optimal path, selecting a large epsilon will work well. As the environment becomes increasing complex and the importance of the path increases epsilon should be decreased. Where it becomes difficult is when epsilon should be decreased and computation is limited. In this case, I would recommend a system similar to the one implemented in this hw where epsilon starts large and then decays towards 1 in following iterations as time allows.

2 – Using RRT in the four-dimensional space does add a few problems not occurring in the 2d space. However, as RRT is intended to be implemented in the state space (4d) and not only the configuration space (2d) it is not complicated to overcome these problems as it has been done before. Adding the velocity into the problem affects which node is selected for expansion and the expansion process of the node.

In the 2d case, selecting the nearest node for expansion can be solved simply using Euclidian or city-block distance. In the 4d case this is no longer true as a node closer in configuration space to the sampled point may be moving away from the sampled node and therefor be further in the state space. This can be fixed through two ways depending upon computation resource. First, include the velocity in the distance calculation and weigh the velocity and position components. This will account for velocity but may not account for nearby nodes that could expand in the desired direction but are currently moving in a slightly different direction then the sampled position. Second, search the tree for candidates whose position distance is below some set threshold. Expand each of these nodes, temporarily, to determine how close their expanded state reaches. Add the expanded node closest to the sample to the tree. This would likely provide better matches but would significantly increase the computation time. Selecting between the two methods requires balancing the quality of expansions and the quantity of expansions, which is not a trivial decision. I would select the second option and perform some testing to set an adequate threshold if the environment is complex and the first otherwise.

To expand nodes is complicated somewhat by adding the velocity but not significantly. When expanding from a current node the action is chosen that moves the node towards the sampled point. Previously this was simple by moving a set distance in the direction of the sample. Now however the decision is more complicated as you have to select if you should speed up, maintain speed, or even decrease speed while moving towards the sampled state. This can be chosen randomly or by moving a random amount towards the sampled state. I would imagine that moving randomly would lead to the same problem of a naïve 2d tree that never expands. Alternatively moving randomly in the direction of the sampled state would probably lead to a better exploration of the space, for this reason I would select this approach.