Lab Assignment #5

TU Berlin Robotics WS 2018/2019

Abhiraj Bishnoi 406221
Kyra Kerz
Vito Mengers 371499
Justus Schmidt 405562

February 2019

A. Our approach

Our approach consists of four extensions that mostly aim to improve speed (and therefore also the number of queries) and decrease the number of nodes. These extensions are the avoiding of the Voronoi bias close to borders, an improved nearest neighbor search, avoiding exhausted nodes that failed very often and a different sampling strategy. In the end we also tried to increase the performance by some tuning of the new parameters we introduced in our extensions.

1. Extension: Avoiding the Voronoi bias

The Voronoi bias ensures that the tree explores rapidly into big unexplored regions and thus makes the expansion of nodes that are at the outer surface of the tree more likely. While this is in general the desired behaviour in open spaces, it leads to problems in spaces with many possible collisions. When there is a collision border in the C-space nodes that are close to it mostly have big Voronoi regions if the area behind the border is unexplored. Because they are close to the border an attempted extension of these nodes fails most of the times. To reduce the likelihood of an attempted extension at these points we reduce the size of their Voronoi region using the approach proposed in [1]. The extension aims to reduce the likelihood of extension of border nodes by

1. Detect nodes that are close to an obstacle.
2. Reduce the size of the Voronoi regions of these nodes.

The detection of nodes that are close to the border is very straight forward. When an extension of a node fails completely, there is a collision at most δ away and are thus close to a border, where δ is the minimum step-size of the connect steps.

The size of the Voronoi region of these nodes is then reduced to at maximum a sphere around it with the radius R. Because most nodes have other nodes in their neighborhood that will closer than $2R$ the effective Voronoi region of the nodes is often smaller in the direction of their neighbors (the Voronoi regions of their neighbors stay the same). If a samples nearest neighbor is now such a border node with maximum sphere around it, we only try to connect to it, if its also inside of the sphere around it, otherwise we will take a new sample, as the extension would most likely fail anyways. This way we reduced the likelihood of an attempted extension of a border node and we explore other regions more often.

This approach however leads to more nearest neighbor searches on average per expansion since it is searched for a neighbor for all samples, but there is no expansion to all samples. Therefore we try to reduce the time needed for the nearest neighbor search in the next extension.

2. Extension: Improved nearest neighbour search

The calculation of every distance to each node inside the tree is costly, therefore we decided to partition our configuration space into clusters. Each cluster contains a centroid and a vector of nearby vertices, which are representing a cluster. In the first iteration we're calculating the distances to each centroid to determine the cluster which is supposed to hold the nearest neighbour. After that we're looping through the resulting cluster to find the nearest neighbour.
Everytime a new vertex is constructed it will be associated to a cluster. That can happen in two ways. If the new vertex lies within a certain threshold distance to a centroid, it will be added to the vector of nearby vertices. If there are more centroids that are fulfilling the threshold requirement, the nearest one will be chosen. If no centroid lies in that threshold, the vertex will be assigned as centroid for a new cluster.
The threshold is set to $3 \cdot delta$ with $delta$ being the step size, that is already given in RrtConConBase. The threshold should not be too small, because it reduces the possibility that vertices wont be assigned to a cluster and create a new one, which is resulting in a lower number of clusters compared to the number of vertices, which increases the computation time for searching the nearest neighbour. On the other hand the threshold should not be too high, because this way the clusters may overlap. Rearranging the clusters could resolve overlapping, but it would increase the computation time.

3. Extension: Avoiding exhausted nodes

Each node/vertex was given an attribute $exhaustCount$ to count the number of times, when they failed to connect to their destination due to collision. If a vertex reaches a certain threshold, it will be less likely be chosen as the nearest neighbour. The cluster that is supposed to deliver the nearest neighbour will still be searched for the neighbour with the shortest distance, but the $exhaustCount$ will be checked against a certain threshold too. If the node with the shortest distance has a higher $exhaustCount$ than the threshold, it will be skipped and the vertex with the second nearest distance will be chosen and so on. If a cluster contains only exhausted nodes, the node with the lowest distance will be chosen. This ensures, that a node with a high $exhaustCount$ wont be discarded for ever, but is just less likely to be chosen. It will be assumed, that a vertex which has failed to connect a certain amount of times is nearby an obstacle. Therefore it is less likely to find a path through that node and another node will be chosen to speed up the search for a path. On the other hand the $exhaustCount$ will increase too, if there are narrow passages and not just dead ends. Therefore the threshold should not be too low to prevent the search from finding a path through narrow passages. To ensure generalization the threshold will change dynamically in regard to the highest $exhaustCount$ of all nodes and just the highest 5% will be penalized. Of course in the very beginning the nodes will have a low *exhaustCount* and therefore an offset of 10 was added.

4. Extension: Sampling

As our other extensions already increased the speed of the algorithm (especially with reducing the number of queries and speeding up the nearest neighbor search) we wanted to reduce the number of nodes as well. In order to do that we tried to use different sampling strategies.

Our first try was to get a sampling strategy that would generate more samples in the space between the two trees through a Gaussian distribution around the point between the trees, but this strategy was not able to improve the performance at all (it decreased it). We suppose that this is due to the fact that in C-Space the narrow passage looks different and is not strictly in the area where the trees come very close together (as it is in operational space).

Because of this we tried using the Bridge-Sampling strategy which in combination with our other strategies also did not work out. Therefore we decided on using the Gaussian-Sampling approach where we pick a random sample (uniformly distributed) and another random sample, which is positioned close to it(Gaussian around the first sample). We only use the first sample for our algorithm, if it is not colliding while the second is colliding. This means that we only take samples that are close to obstacles in C-Space. Because we hope that these points still are at least somewhat uniformly distributed in the C-Space, which should be the case given a large enough standard deviation σ for the Gaussian of the second sample (so that the points do not have to be to close to the obstacle). This reduces the number of nodes considerably in comparison to uniform sampling, while also sadly reducing the speed a bit (probably due to the fact that the Gaussian samples are not really uniformly distributed in the space).

5. Implementation

In *YourPlanner* we implemented most of our strategies by modifying the given versions of the base class. This way the nearest neighbor search, Voronoi bias avoidance and the exhausted nodes are implemented. In *YourSampler* our Gaussian-Sampler and Bridge-Sampler are implemented, while only the Gaussian one is used.

B. Results

	RRTConCon	RRTConCon (reversed)	Your Planner	Your Planner (reversed)
avgT	59752.9 ms	45511.8 ms	24894.9 ms	24263.3 ms
stdT	33418.54	28937.53	7745.78	9388.14
avgNodes	12638.1	10966.3	4823	4881.5
avgQueries	624246	550024	419441	432664

Our algorithm performs better in every aspect compared to RRTConCon. Especially the Nearest Neighbour Extention in combination with avoiding the Voronoi bias results in a significant reduction of computation time. As explained in the corresponding extension descriptions the Voronoi bias reduces the computation time by preventing extensions that would most likely fail and reduces the amount of average queries and the amount of nodes created by connect(). On the downside it leads to more nearest neighbor searches. This drawback is eliminated by improving the nearest neighbour search. Instead of calculating the distance to every node, the distance to each cluster will be calculated and after that only in the nearest cluster the nearest neighbour will be searched. This also results in decreased computation time. Choosing exhausted nodes less likely results in a reduction of average queries and nodes created by connect(), because similar to the Voronoi bias it is preventing extensions that would most likely fail. Both extensions also are reducing the standard deviation time, because it is less likely to get stuck in a blind alley, if you're unlucky. The Gaussian sampler also reduces the amount of nodes due to sampling along obstacles, which results in finding ways through narrow paths and not choosing nodes for the connecting operation in the free space.

Implementation Contribution

Student Name	Extension 1	Extension 2	Extension 3	Extension 4
Abhiraj Bishnoi			X	
Kyra Kerz		X	X	
Vito Mengers	X			X
Justus Schmidt			X	

References

[1] A. Yershova, L. Jaillet, T. Simeon and S. M. LaValle, "Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain," Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 2005, pp. 3856-3861.