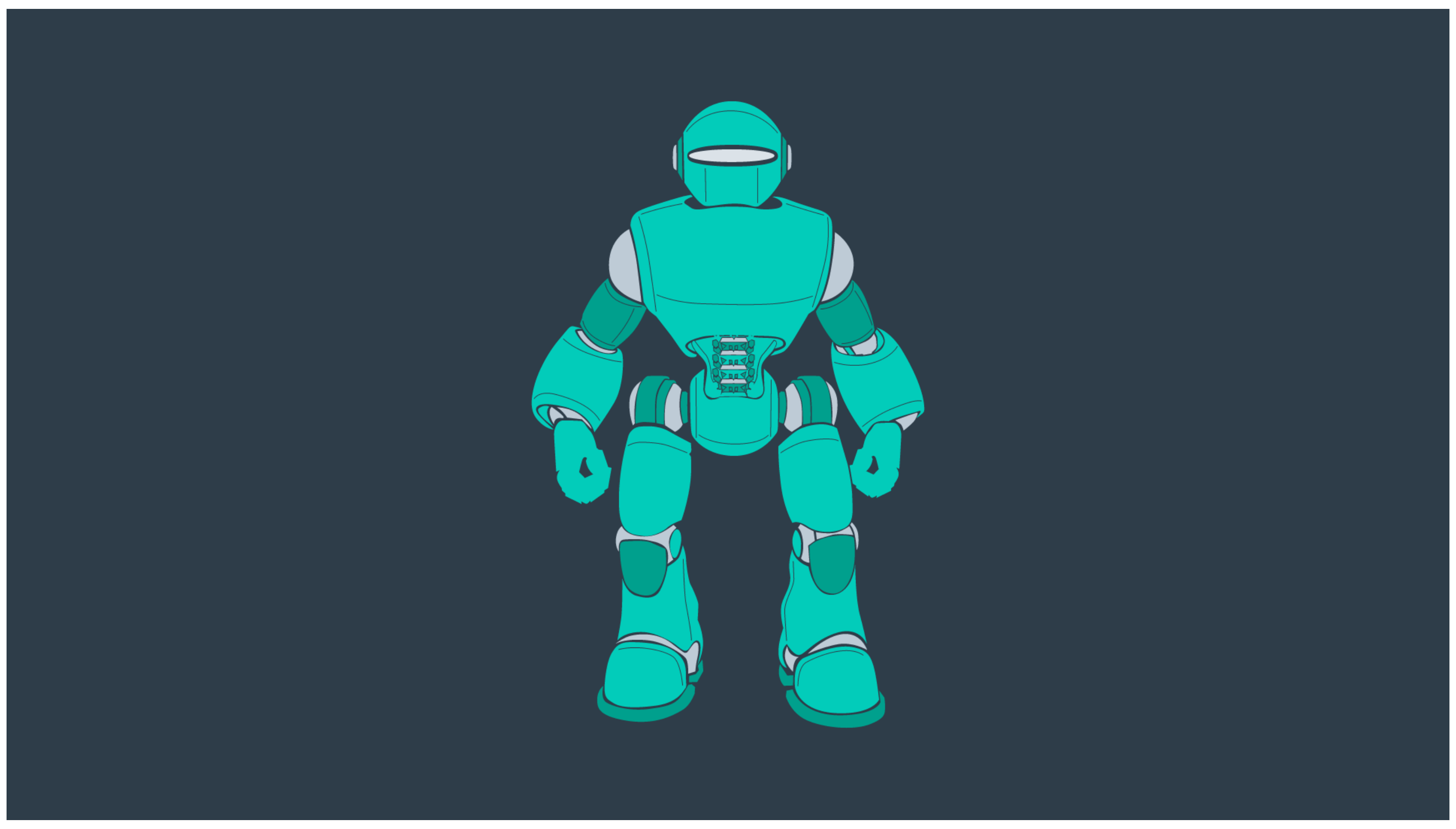
Why Sample-Based Planning?

So why exactly can't we use discrete planning for higher dimensional problems? Well, it's incredibly hard to discretize such a large space. The complexity of the path planning problem increases exponentially with the number of dimensions in the C-space.

Increased Dimensionality

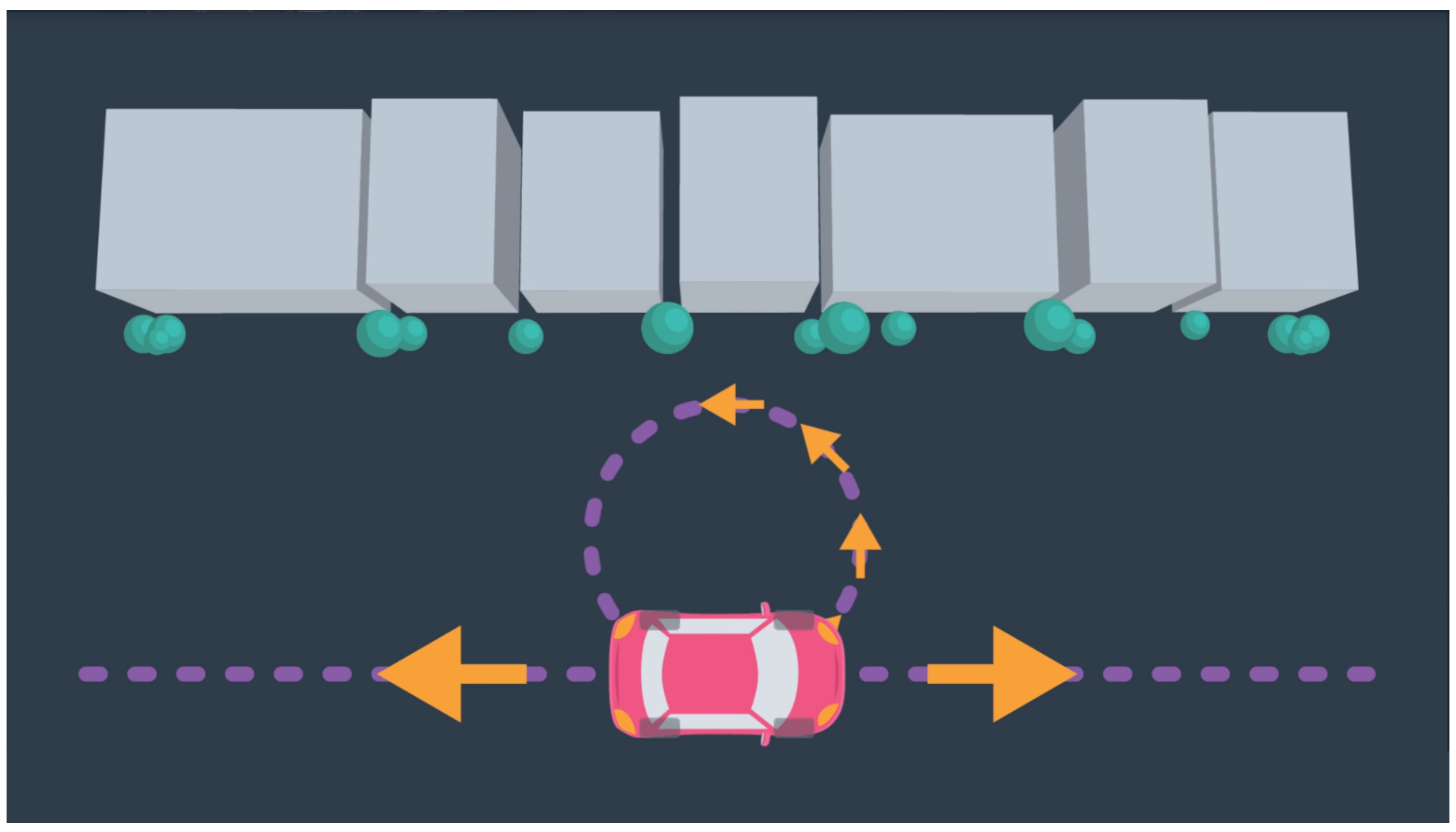
For a 2-dimensional 8-connected space, every node has 8 successors (8-connected means that from every cell you can move laterally or diagonally). Imagine a 3-dimensional 8-connected space, how many successors would every node have? 26. As the dimension of the C-space grows, the number of successors that every cell has increases substantially. In fact, for an n-dimensional space, it is equal to $3^n - 1$.

It is not uncommon for robots and robotic systems to have large numbers of dimensions. Recall the robotic arm that you worked with in the pick-and-place project - that was a 6-DOF arm. If multiple 6-DOF arms work in a common space, the computation required to perform path planning to avoid collisions increases substantially. Then, think about the complexity of planning for humanoid robots such as the one depicted below. Such problems may take intolerably long to solve using the combinatorial approach.



Constrained Dynamics

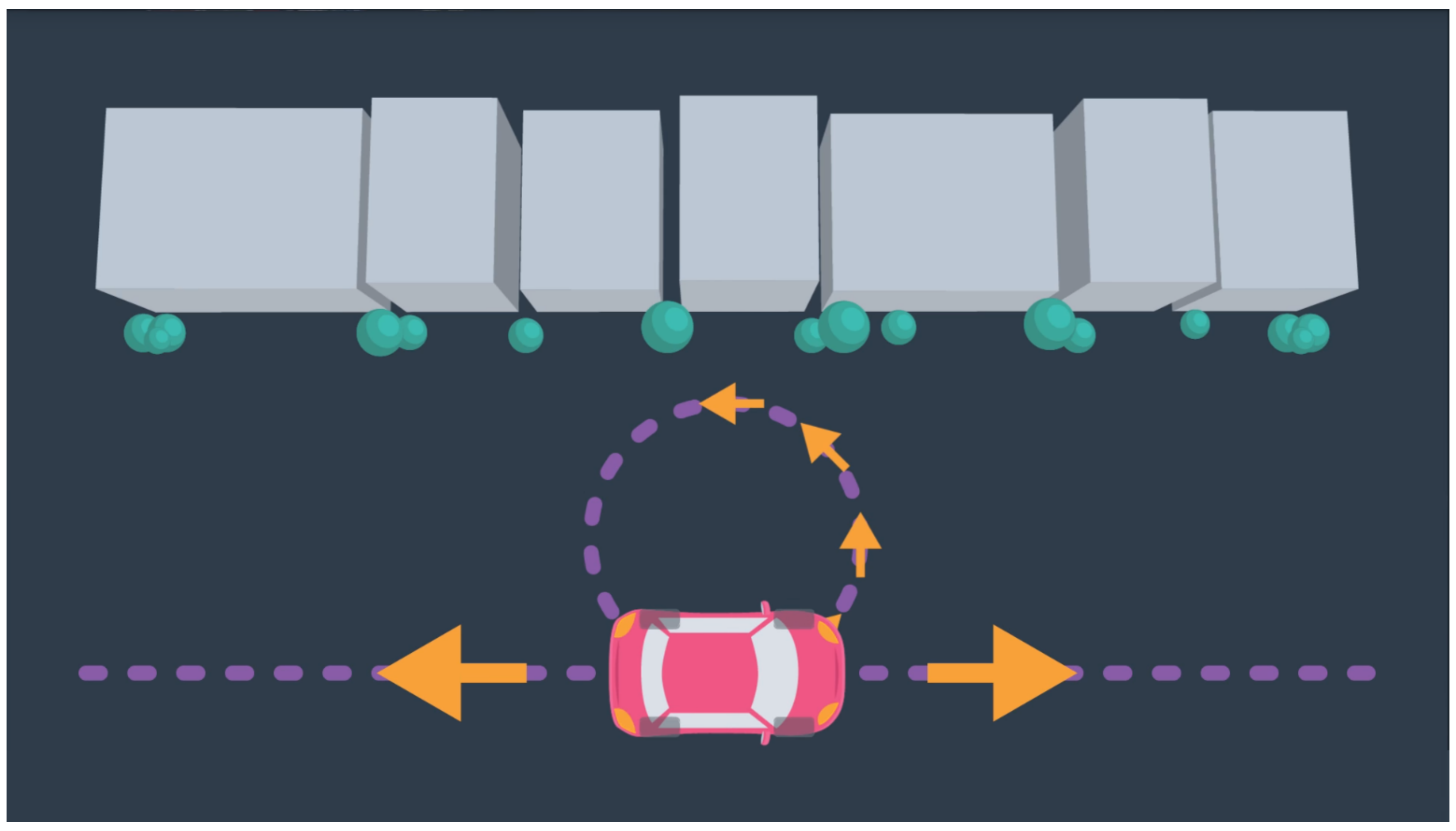
Aside from robots with many degrees of freedom and multi-robot systems, another computational difficulty involves working with robots that have constrained dynamics. For instance, a car is limited in its motion - it can move forward and backward, and it can turn with a limited turning radius - as you can see in the image below.



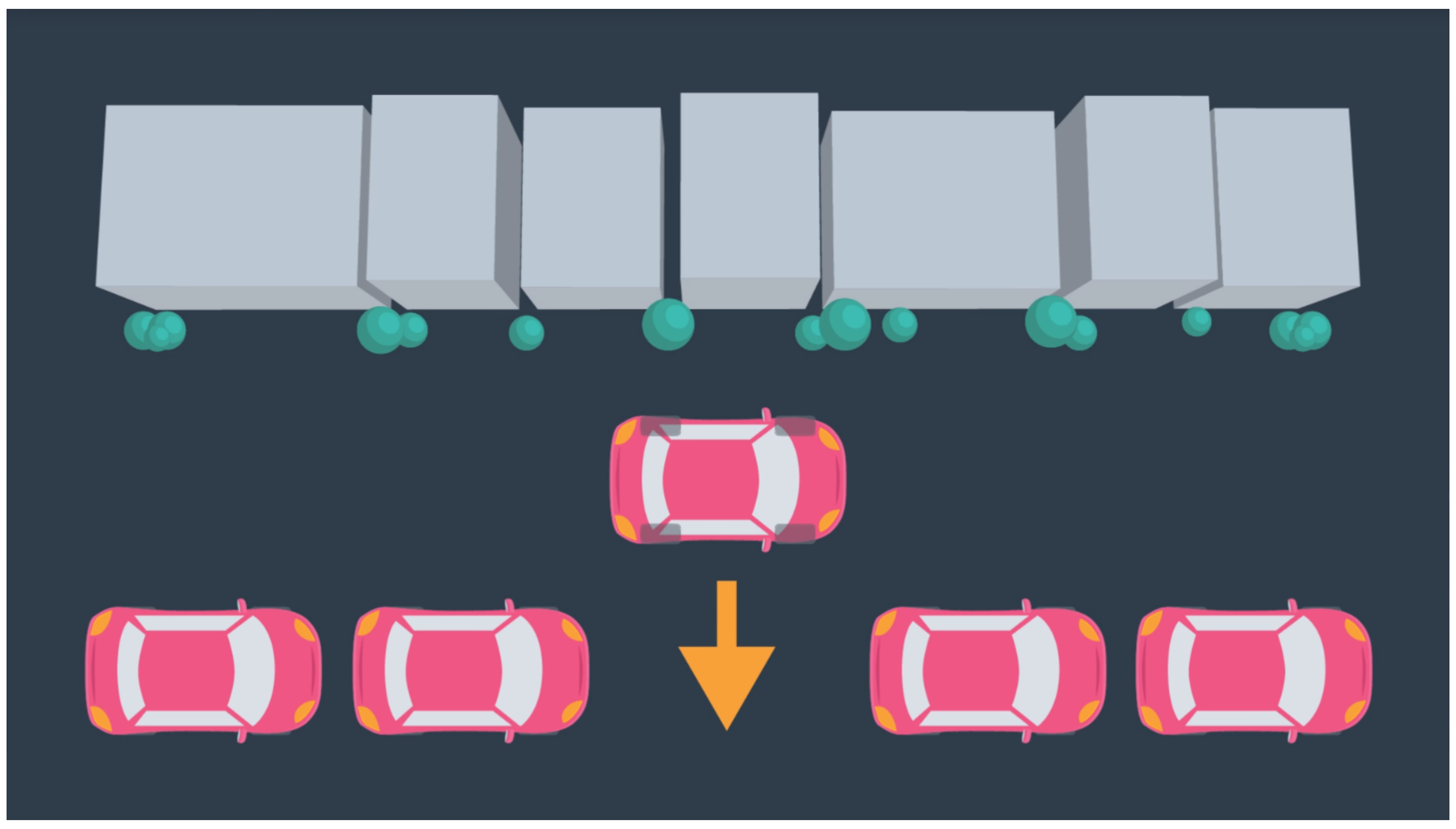
However, the car is *not* able to move laterally - as depicted in the following image. *(As unfortunate as it is for those of us that struggle to parallel park!)*

Constrained Dynamics

Aside from robots with many degrees of freedom and multi-robot systems, another computational difficulty involves working with robots that have constrained dynamics. For instance, a car is limited in its motion - it can move forward and backward, and it can turn with a limited turning radius - as you can see in the image below.



However, the car is *not* able to move laterally - as depicted in the following image. *(As unfortunate as it is for those of us that struggle to parallel park!)*



In the case of the car, more complex motion dynamics must be considered when path planning - including the derivatives of the state variables such as velocity. For example, a car's safe turning radius is dependent on it's velocity.

Robotic systems can be classified into two different categories - holonomic and non-holonomic. **Holonomic systems** can be defined as systems where every constraint depends exclusively on the current pose and time, and not on any derivatives with respect to time. **Nonholonomic systems**, on the other hand, are dependent on derivatives. Path planning for nonholonomic systems is more difficult due to the added constraints.

In this section, you will learn two different path planning algorithms, and understand how to tune their parameters for varying applications.

NEXT

Weakening Requirements

Combinatorial path planning algorithms are too inefficient to apply in high-dimensional environments, which means that some practical compromise is required to solve the problem! Instead of looking for a path planning algorithm that is both complete and optimal, what if the requirements of the algorithm were weakened?

Instead of aspiring to use an algorithm that is complete, the requirement can be weakened to use an algorithm that is probabilistically complete. A **probabilistically complete** algorithm is one who's probability of finding a path, if one exists, increases to 1 as time goes to infinity.

Similarly, the requirement of an optimal path can be weakened to that of a feasible path. A **feasible path** is one that obeys all environmental and robot constraints such as obstacles and motion constraints. For high-dimensional problems with long computational times, it may take unacceptably long to find the optimal path, whereas a feasible path can be found with relative ease. Finding a feasible path proves that a path from start to goal exists, and if needed, the path can be optimized locally to improve performance.

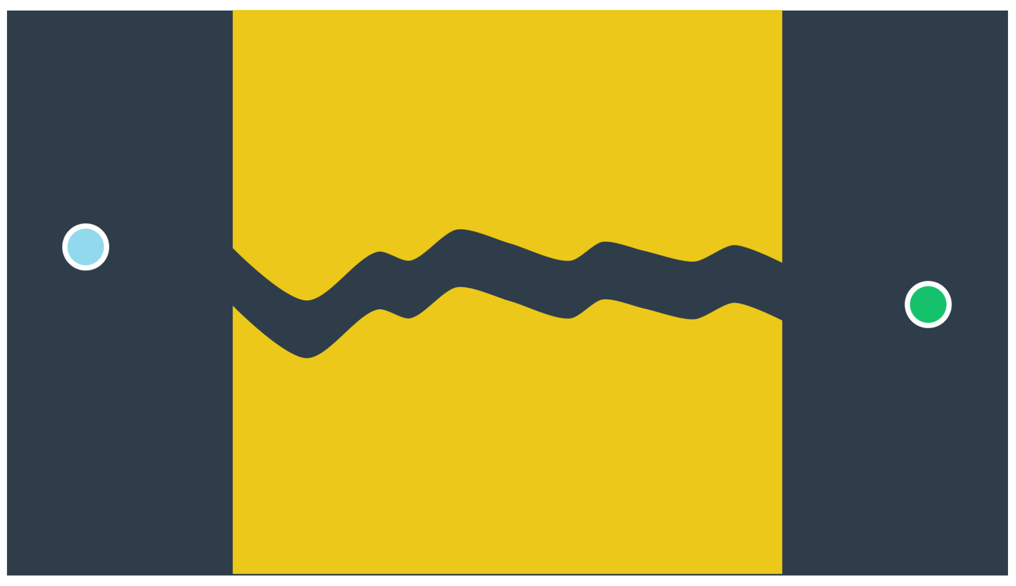
Sample-based planning is probabilistically complete and looks for a feasible path instead of the optimal path.

Sample-Based Path Planning

Sample-based path planning differs from combinatorial path planning in that it does not try to systematically discretize the entire configuration space. Instead, it samples the configuration space randomly (or semi-randomly) to build up a representation of the space. The resultant graph is not as precise as one created using combinatorial planning, but it is much quicker to construct because of the relatively small number of samples used.

Such a method is probabilistically complete because as time passes and the number of samples approaches infinity, the probability of finding a path, if one exists, approaches 1.

Such an approach is very effective in high-dimensional spaces, however it does have some downfalls. Sampling a space uniformly is not likely to reach small or narrow areas, such as the passage depicted in the image below. Since the passage is the only way to move from start to goal, it is critical that a sufficient number of samples occupy the passage, or the algorithm will return ‘no solution found’ to a problem that clearly has a solution.



Different sample-based planning approaches exist, each with their own benefits and downfalls. In the next few pages you will learn about,

- Probabilistic Roadmap Method
- Rapidly Exploring Random Tree Method

You will also learn about Path Smoothing - one improvement that can make resultant paths more efficient.

Algorithm

The pseudocode for the PRM learning phase is provided below.

```
Initialize an empty graph
For n iterations:

   Generate a random configuration.
   If the configuration is collision free:

      Add the configuration to the graph.
      Find the k-nearest neighbours of the configuration.
      For each of the k neighbours:

         Try to find a collision-free path between
            the neighbour and original configuration.
            If edge is collision-free:
                Add it to the graph.
```

After the learning phase, comes the query phase.

Setting Parameters

There are several parameters in the PRM algorithm that require tweaking to achieve success in a particular application. Firstly, the **number of iterations** can be adjusted - the parameter controls between how detailed the resultant graph is and how long the computation takes. For path planning problems in wide-open spaces, additional detail is unlikely to significantly improve the resultant path. However, the additional computation is required in complicated environments with narrow passages between obstacles. Beware, setting an insufficient number of iterations can result in a 'path not found' if the samples do not adequately represent the space.

Another decision that a robotics engineer would need to make is **how to find neighbors** for a randomly generated configuration. One option is to look for the k-nearest neighbors to a node. To do so efficiently, a k-d tree can be utilized - to break up the space into 'bins' with nodes, and then search the bins for the nearest nodes. Another option is to search for any nodes within a certain distance of the goal. Ultimately, knowledge of the environment and the solution requirements will drive this decision-making process.

The choice for what type of **local planner** to use is another decision that needs to be made by the robotics engineer. The local planner demonstrated in the video is an example of a very simple planner. For most scenarios, a simple planner is preferred, as the process of checking an edge for collisions is repeated many times (k*n times, to be exact) and efficiency is key. However, more powerful planners may be required in certain problems. In such a case, the local planner could even be another PRM.

Probabilistically Complete

As discussed before, sample-based path planning algorithms are probabilistically complete. Now that you have seen one such algorithm in action, you can see why this is the case. As the number of iterations approaches infinity, the graph approaches completeness and the optimal path through the graph approaches the optimal path in reality.

Variants

The algorithm that you learned here is the vanilla version of PRM, but many other variations to it exist. The following link discusses several alternative strategies for implementing a PRM that may produce a more optimal path in a more efficient manner.

- A Comparative Study of Probabilistic Roadmap Planners

PRM is a Multi-Query Planner

```
    Find the k-nearest neighbours of the configuration.
    For each of the k neighbours:

        Try to find a collision-free path between
            the neighbour and original configuration.
            If edge is collision-free:
                Add it to the graph.
```

After the learning phase, comes the query phase.

Setting Parameters

There are several parameters in the PRM algorithm that require tweaking to achieve success in a particular application. Firstly, the **number of iterations** can be adjusted - the parameter controls between how detailed the resultant graph is and how long the computation takes. For path planning problems in wide-open spaces, additional detail is unlikely to significantly improve the resultant path. However, the additional computation is required in complicated environments with narrow passages between obstacles. Beware, setting an insufficient number of iterations can result in a 'path not found' if the samples do not adequately represent the space.

Another decision that a robotics engineer would need to make is **how to find neighbors** for a randomly generated configuration. One option is to look for the k-nearest neighbors to a node. To do so efficiently, a k-d tree can be utilized - to break up the space into 'bins' with nodes, and then search the bins for the nearest nodes. Another option is to search for any nodes within a certain distance of the goal. Ultimately, knowledge of the environment and the solution requirements will drive this decision-making process.

The choice for what type of **local planner** to use is another decision that needs to be made by the robotics engineer. The local planner demonstrated in the video is an example of a very simple planner. For most scenarios, a simple planner is preferred, as the process of checking an edge for collisions is repeated many times (k*n times, to be exact) and efficiency is key. However, more powerful planners may be required in certain problems. In such a case, the local planner could even be another PRM.

Probabilistically Complete

As discussed before, sample-based path planning algorithms are probabilistically complete. Now that you have seen one such algorithm in action, you can see why this is the case. As the number of iterations approaches infinity, the graph approaches completeness and the optimal path through the graph approaches the optimal path in reality.

Variants

The algorithm that you learned here is the vanilla version of PRM, but many other variations to it exist. The following link discusses several alternative strategies for implementing a PRM that may produce a more optimal path in a more efficient manner.

- A Comparative Study of Probabilistic Roadmap Planners

PRM is a Multi-Query Planner

The Learning Phase takes significantly longer to implement than the Query Phase, which only has to connect the start and goal nodes, and then search for a path. However, the graph created by the Learning Phase can be reused for many subsequent queries. For this reason, PRM is called a **multi-query planner**.

This is very beneficial in static or mildly-changing environments. However, some environments change so quickly that PRM's multi-query property cannot be exploited. In such situations, PRM's additional detail and computational slow nature is not appreciated. A quicker algorithm would be preferred - one that doesn't spend time going in *all* directions without influence by the start and goal.

QUIZ QUESTION

Which of the following statements are true about Probabilistic Roadmaps?

- ☐ PRM is a multi-query method (ie. the resultant graph can be used for multiple queries).
- ☐ If an insufficient iterations of the PRM algorithm are run, there is a risk of finding an inefficient path or not finding a path at all.
- ☐ When A* is applied to the graph generated by PRM, the optimal path is found.

SUBMIT

NEXT

Algorithm

The pseudocode for the PRM learning phase is provided below.

```
Initialize an empty graph
For n iterations:

   Generate a random configuration.
   If the configuration is collision free:

      Add the configuration to the graph.
      Find the k-nearest neighbours of the configuration.
      For each of the k neighbours:

         Try to find a collision-free path between
            the neighbour and original configuration.
            If edge is collision-free:
                Add it to the graph.
```

After the learning phase, comes the query phase.

Setting Parameters

There are several parameters in the PRM algorithm that require tweaking to achieve success in a particular application. Firstly, the **number of iterations** can be adjusted - the parameter controls between how detailed the resultant graph is and how long the computation takes. For path planning problems in wide-open spaces, additional detail is unlikely to significantly improve the resultant path. However, the additional computation is required in complicated environments with narrow passages between obstacles. Beware, setting an insufficient number of iterations can result in a 'path not found' if the samples do not adequately represent the space.

Another decision that a robotics engineer would need to make is **how to find neighbors** for a randomly generated configuration. One option is to look for the k-nearest neighbors to a node. To do so efficiently, a k-d tree can be utilized - to break up the space into 'bins' with nodes, and then search the bins for the nearest nodes. Another option is to search for any nodes within a certain distance of the goal. Ultimately, knowledge of the environment and the solution requirements will drive this decision-making process.

The choice for what type of **local planner** to use is another decision that needs to be made by the robotics engineer. The local planner demonstrated in the video is an example of a very simple planner. For most scenarios, a simple planner is preferred, as the process of checking an edge for collisions is repeated many times (k*n times, to be exact) and efficiency is key. However, more powerful planners may be required in certain problems. In such a case, the local planner could even be another PRM.

Probabilistically Complete

As discussed before, sample-based path planning algorithms are probabilistically complete. Now that you have seen one such algorithm in action, you can see why this is the case. As the number of iterations approaches infinity, the graph approaches completeness and the optimal path through the graph approaches the optimal path in reality.

Variants

The algorithm that you learned here is the vanilla version of PRM, but many other variations to it exist. The following link discusses several alternative strategies for implementing a PRM that may produce a more optimal path in a more efficient manner.

- A Comparative Study of Probabilistic Roadmap Planners

PRM is a Multi-Query Planner

The Learning Phase takes significantly longer to implement than the Query Phase, which only has to connect the start and goal nodes, and then search for a path. However, the graph created by the Learning Phase can be reused for many subsequent queries. For this reason, PRM is called a **multi-query planner**.

This is very beneficial in static or mildly-changing environments. However, some environments change so quickly that PRM's multi-query property cannot be exploited. In such situations, PRM's additional detail and computational slow nature is not appreciated. A quicker algorithm would be preferred - one that doesn't spend time going in *all* directions without influence by the start and goal.

QUIZ QUESTION

Which of the following statements are true about Probabilistic Roadmaps?

☐ PRM is a multi-query method (ie. the resultant graph can be used for multiple queries).

Algorithm

The pseudocode for the RRT learning phase is provided below.

```
Initialize two empty trees.
Add start node to tree #1.
Add goal node to tree #2.
For n iterations, or until an edge connects trees #1 & #2:

    Generate a random configuration (alternating trees).
    If the configuration is collision free:
        Find the closest neighbour on the tree to the configuration
        If the configuration is less than a distance $\delta$ away from the neighbour:
            Try to connect the two with a local planner.
    Else:
        Replace the randomly generated configuration
            with a new configuration that falls along the same path,
            but a distance $\delta$ from the neighbour.
        Try to connect the two with a local planner.

    If node is added successfully:
        Try to connect the new node to the closest neighbour.
```

Setting Parameters

Just like with PRM, there are a few parameters that can be tuned to make RRT more efficient for a given application.

The first of these parameters is the **sampling method** (ie. how a random configuration is generated). As discussed in the video, you can sample uniformly - which would favour wide unexplored spaces, or you can sample with a bias - which would cause the search to advance greedily in the direction of the goal. Greediness can be beneficial in simple planning problems, however in some environments it can cause the robot to get stuck in a local minima. It is common to utilize a uniform sampling method with a *small* hint of bias.

The next parameter that can be tuned is δ. As RRT starts to generate random configurations, a large proportion of these configurations will lie further than a distance δ from the closest configuration in the graph. In such a situation, a randomly generated node will dictate the direction of growth, while δ is the growth rate.

Choosing a small δ will result in a large density of nodes and small growth rate. On the other hand, choosing a large δ may result in lost detail, as well as an increasing number of nodes being unable to connect to the graph due to the greater chance of collisions with obstacles. δ must be chosen carefully, with knowledge of the environment and requirements of the solution.

Single-Query Planner

Since the RRT method explores the graph starting with the start and goal nodes, the resultant graph cannot be applied to solve additional queries. RRT is a single-query planner.

RRT is, however, much quicker than PRM at solving a path planning problem. This is so *because* it takes into account the start and end nodes, and limits growth to the area surrounding the existing graph instead of reaching out into all distant corners, the way PRM does. RRT is more efficient than PRM at solving large path planning problems (ex. ones with hundreds of dimensions) in dynamic environments.

Generally speaking, RRT is able to solve problems with 7 dimensions in a matter of milliseconds, and may take several minutes to solve problems with over 20 dimensions. In comparison, such problems would be impossible to solve with the combinatorial path planning method.

RRT & Non-holonomic Systems

Knowledge
Search project Q&A

Student Hub
Chat with peers and mentors

```
    If the configuration is less than a distance δ away from the neighbour:
        Try to connect the two with a local planner.
Else:
    Replace the randomly generated configuration
        with a new configuration that falls along the same path,
        but a distance δ from the neighbour.
    Try to connect the two with a local planner.

If node is added successfully:
    Try to connect the new node to the closest neighbour.
```

Setting Parameters

Just like with PRM, there are a few parameters that can be tuned to make RRT more efficient for a given application.

The first of these parameters is the **sampling method** (ie. how a random configuration is generated). As discussed in the video, you can sample uniformly - which would favour wide unexplored spaces, or you can sample with a bias - which would cause the search to advance greedily in the direction of the goal. Greediness can be beneficial in simple planning problems, however in some environments it can cause the robot to get stuck in a local minima. It is common to utilize a uniform sampling method with a *small* hint of bias.

The next parameter that can be tuned is δ. As RRT starts to generate random configurations, a large proportion of these configurations will lie further than a distance δ from the closest configuration in the graph. In such a situation, a randomly generated node will dictate the direction of growth, while δ is the growth rate.

Choosing a small δ will result in a large density of nodes and small growth rate. On the other hand, choosing a large δ may result in lost detail, as well as an increasing number of nodes being unable to connect to the graph due to the greater chance of collisions with obstacles. δ must be chosen carefully, with knowledge of the environment and requirements of the solution.

Single-Query Planner

Since the RRT method explores the graph starting with the start and goal nodes, the resultant graph cannot be applied to solve additional queries. RRT is a single-query planner.

RRT is, however, much quicker than PRM at solving a path planning problem. This is so *because* it takes into account the start and end nodes, and limits growth to the area surrounding the existing graph instead of reaching out into all distant corners, the way PRM does. RRT is more efficient than PRM at solving large path planning problems (ex. ones with hundreds of dimensions) in dynamic environments.

Generally speaking, RRT is able to solve problems with 7 dimensions in a matter of milliseconds, and may take several minutes to solve problems with over 20 dimensions. In comparison, such problems would be impossible to solve with the combinatorial path planning method.

RRT & Non-holonomic Systems

While we will not go into significant detail on this topic, the RRT method supports planning for non-holonomic systems, while the PRM method does not. This is so because the RRT method can take into consideration the additional constraints (such as a car's turning radius at a particular speed) when adding nodes to a graph, the same way it already takes into consideration how far away a new node is from an existing tree.

Quiz

QUIZ QUESTION

Which of the following statements are true about the Rapidly Exploring Random Tree method?

- [ ] When a randomly generated configuration is greater than a distance, δ, from its closest neighbour, then the configuration is abandoned.
- RRT can be applied to path planning with non-holonomic systems.
- If δ is set to a large value, the algorithm's efficiency will drop, as the local planner is more likely to encounter collisions along a longer path.

SUBMIT