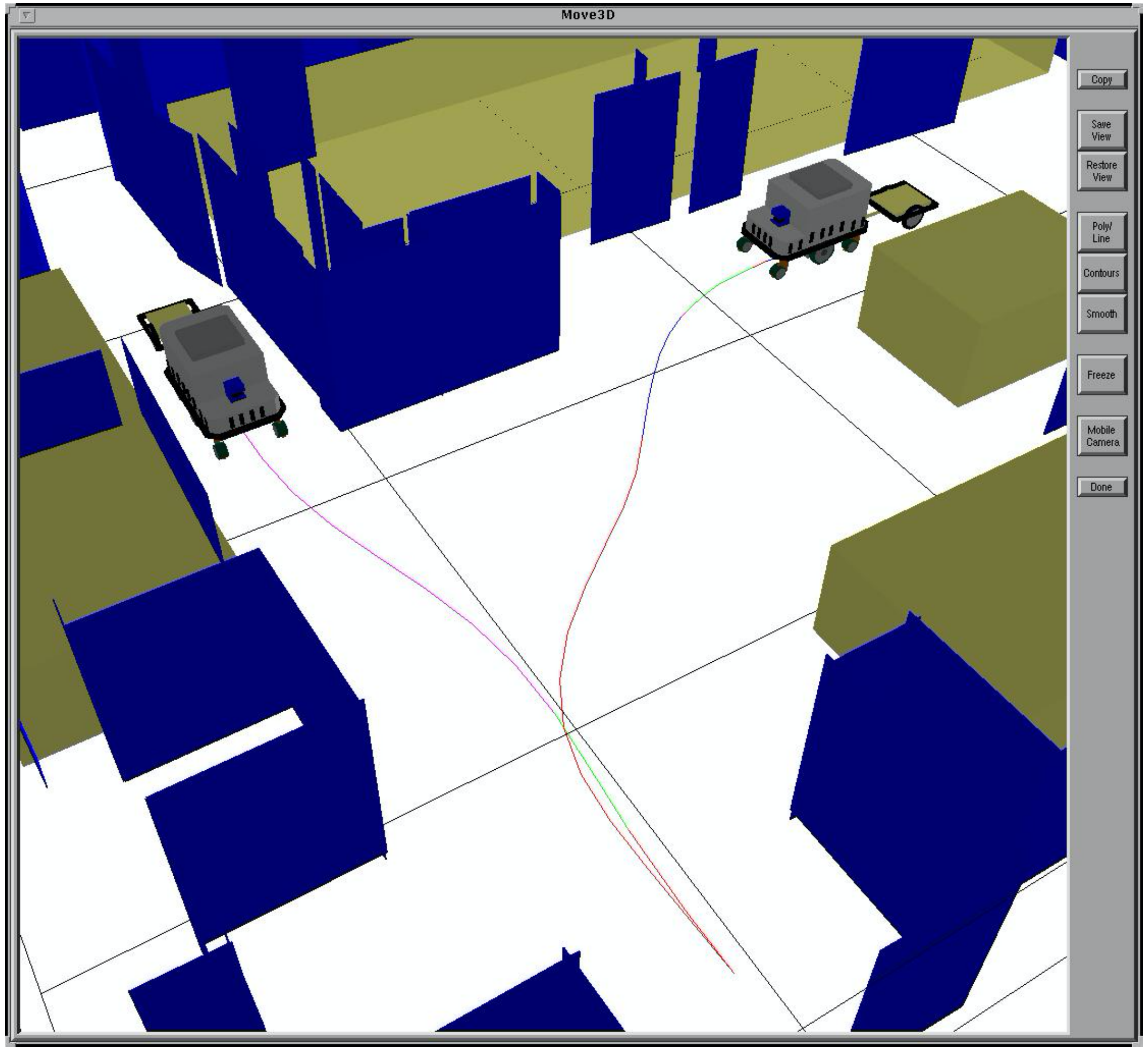
CS 354
Autonomous Robotics

Planning

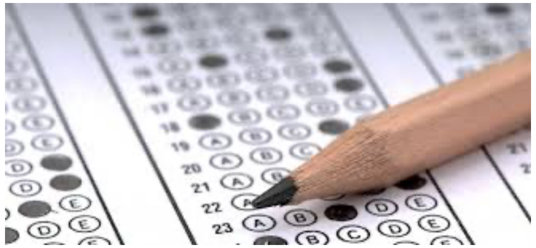
Instructors: Dr. Kevin Molloy and Dr. Nathan Sprague



Logistics

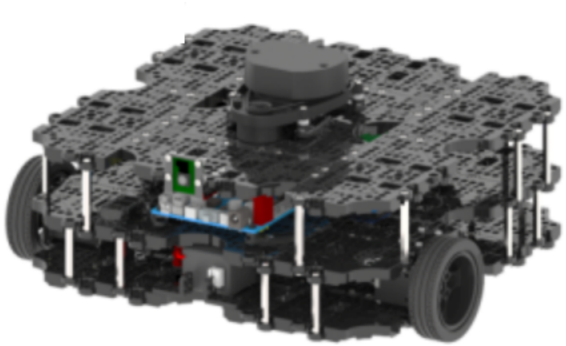
Class Calendar has been updated!

First exam



- Next Tuesday Oct 6 @ 4:00 pm
- Due Thursday Oct 8 @ 2:30 pm

Meet the JMU TurtleBots

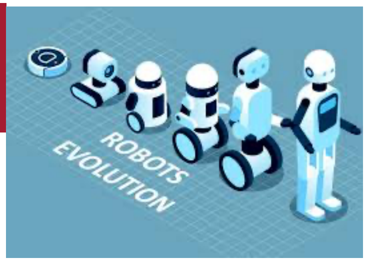


Opportunity to come to the robotics lab on October 8th during class.

We will be running your ROS programs (like wanderer) on the robots.

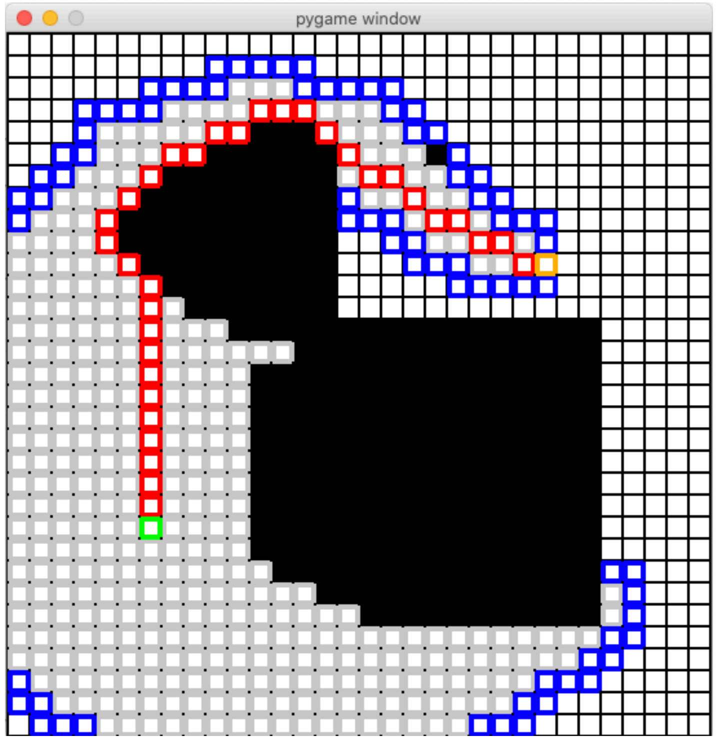
Attendance in person is optional but attendance is mandatory.

Robotics Research Review and Presentation



Dr. Sprague will explain.

Review from Last Time



Optimal Path Planning with A*

- Construct a grid to discretize **C** (the configuration space)

Issues with A*

- Construct a grid to discretize **C** (the configuration space)
- Grid size is exponential to cover C. Bad news for A*

Probabilistic Planner

Rapidly-Exploring Random Trees (LaValle 1998)

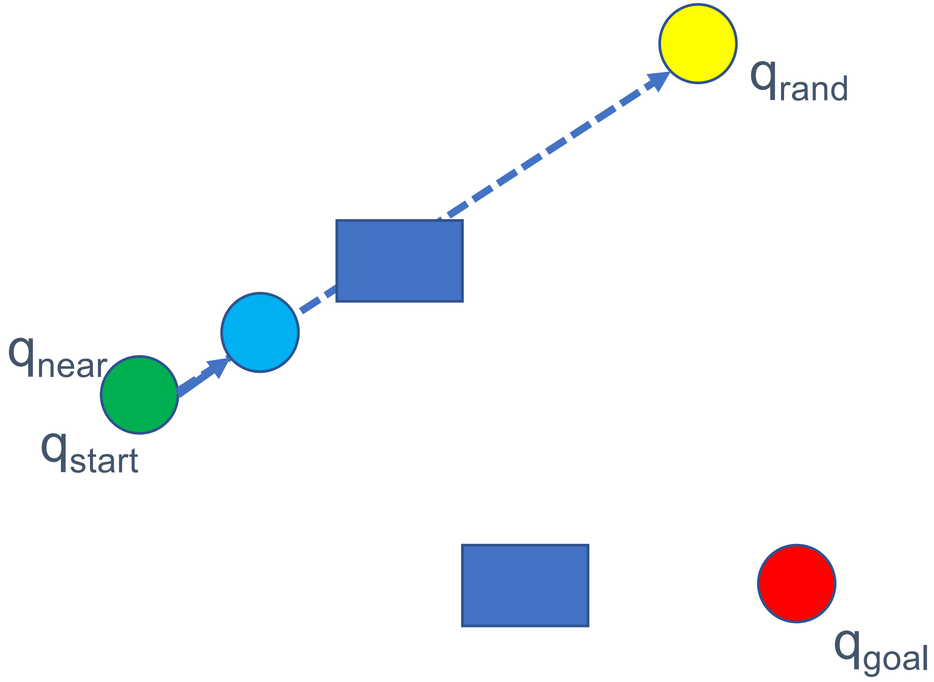
Idea: Grow a search tree in the configuration space that expands the frontier in random directions.

RRT Exploring

Voronoi Diagram

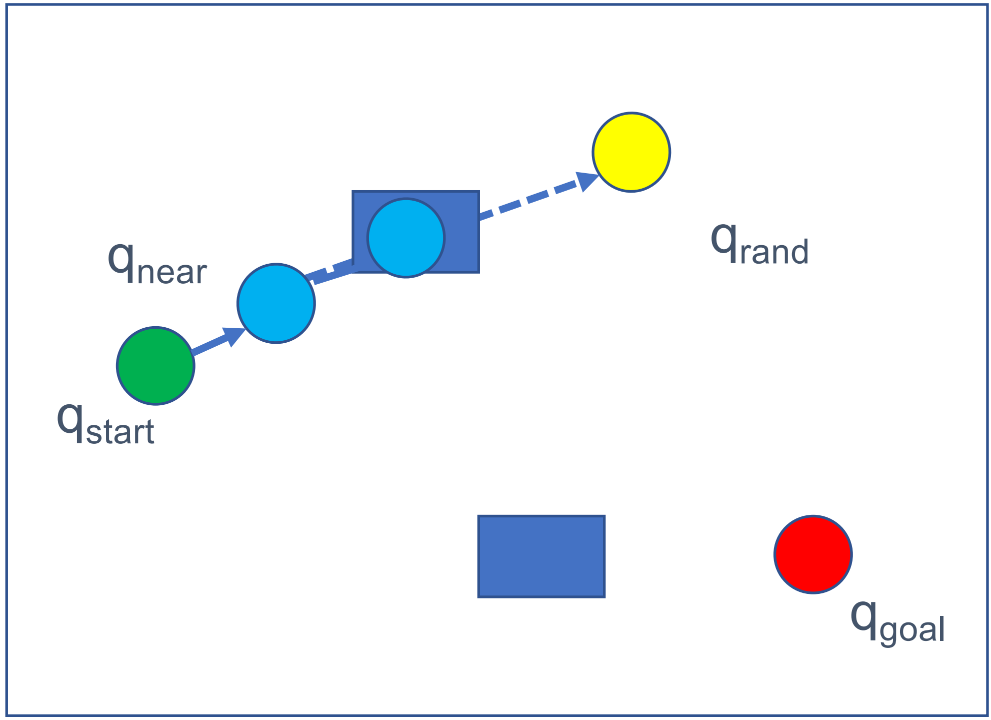
RRT Algorithm

```
RRT (q_start, q_goal, goalTolerance, maxTreeSize)
    tree = new Tree()
    while treeSize < maxTreeSize
        q_rand = SampleRandomConfig()
        q_near  = Tree.findClosest(q_near)
        q_new  = Expand(q_near, q_rand , stepsize)
        if q_new /* path is collision free */
            tree.addNode(q_new)
            tree.addEdge(q_near, q_new)
            if goalCheck(q_new, q_goal, goalTolerance)
                return Tree
    return null
```

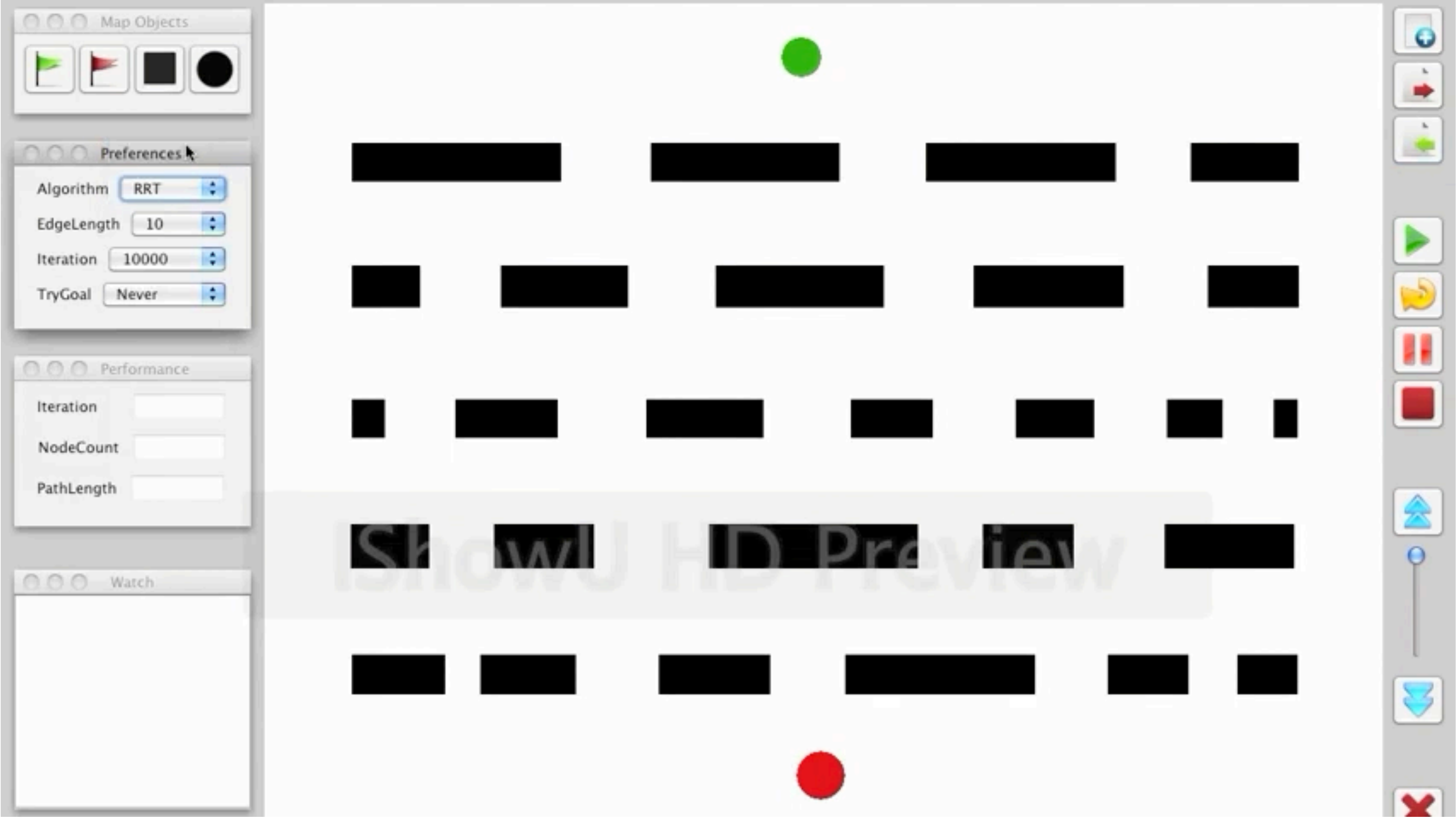


RRT Algorithm

```
RRT (q_start, q_goal, goalTolerance, maxTreeSize)
   tree = new Tree()
   while treeSize < maxTreeSize
      q_rand = SampleRandomConfig()
      q_near  = Tree.findClosest(q_near)
      q_new  = Expand(q_near, q_rand , stepsize)
      if q_new /* path is collision free */
            tree.addNode(q_new)
            tree.addEdge(q_near, q_new)
            if goalCheck(q_new, q_goal, goalTolerance)
               return Tree
   return null
```



RRT Algorithm In Action



HW 2 – Implement Basic RRT

Download the last section of HW2 and complete the code to implement an RRT algorithm that explores a 2d configuration space.

RRT Algorithm Analysis

Is RRT Optimal?

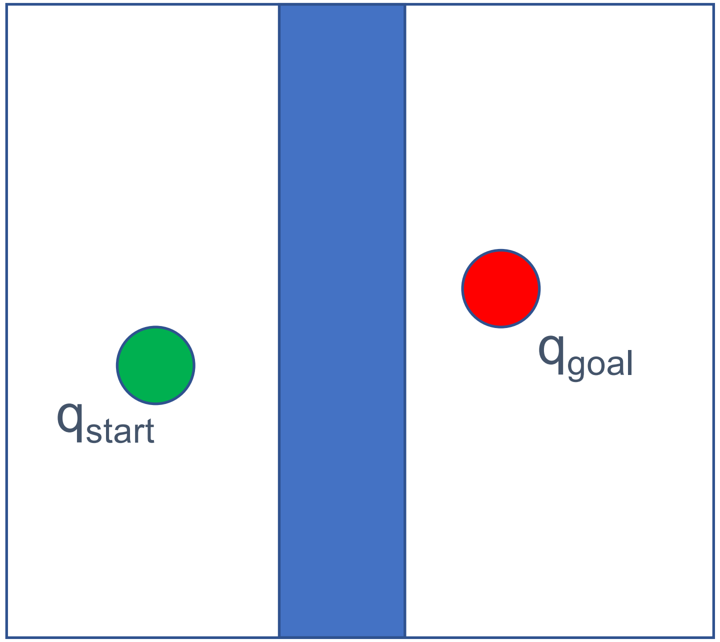
No. Why not?

- The tree is expanded in random directions.
- Nodes are connected to the nearest node (q_{near}) with no concern of any *cost*.

Complete Planning?

If a solution **does not** exist, can A* theoretically tell you that?

Yes, since there is a finite number of grid cells, that algorithm can explore all of them in some amount of time. If all nodes explored, then it can report no solution.



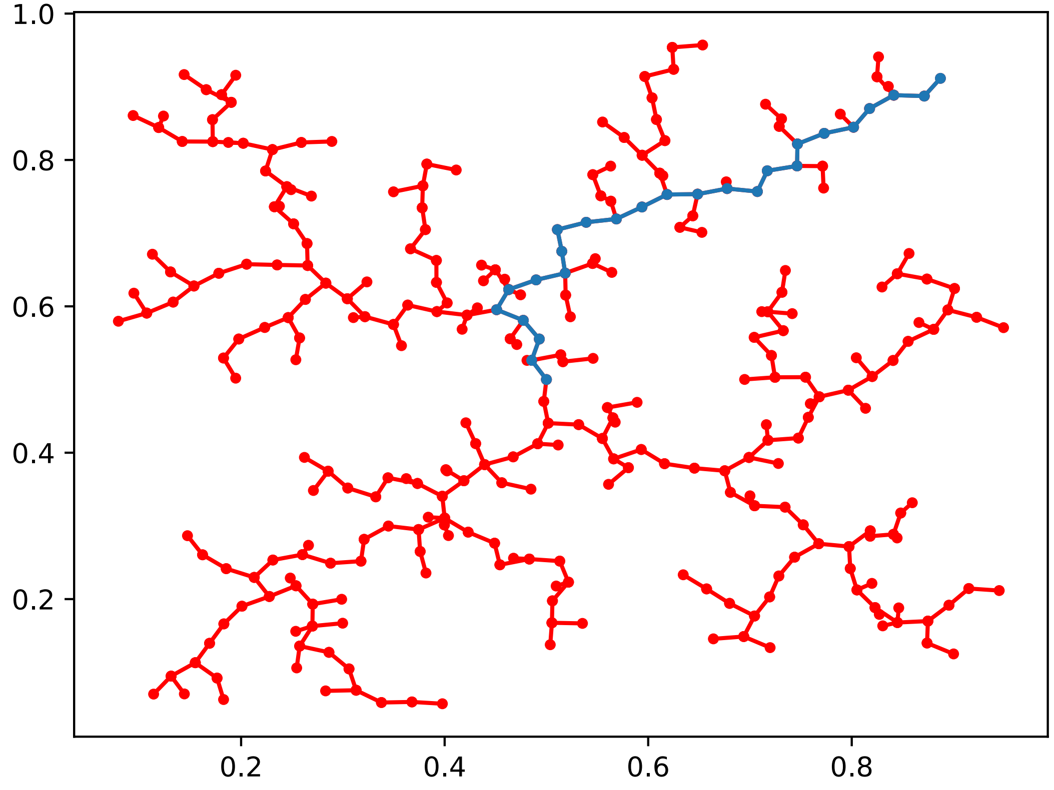
Can RRT inform you that no solution exists?

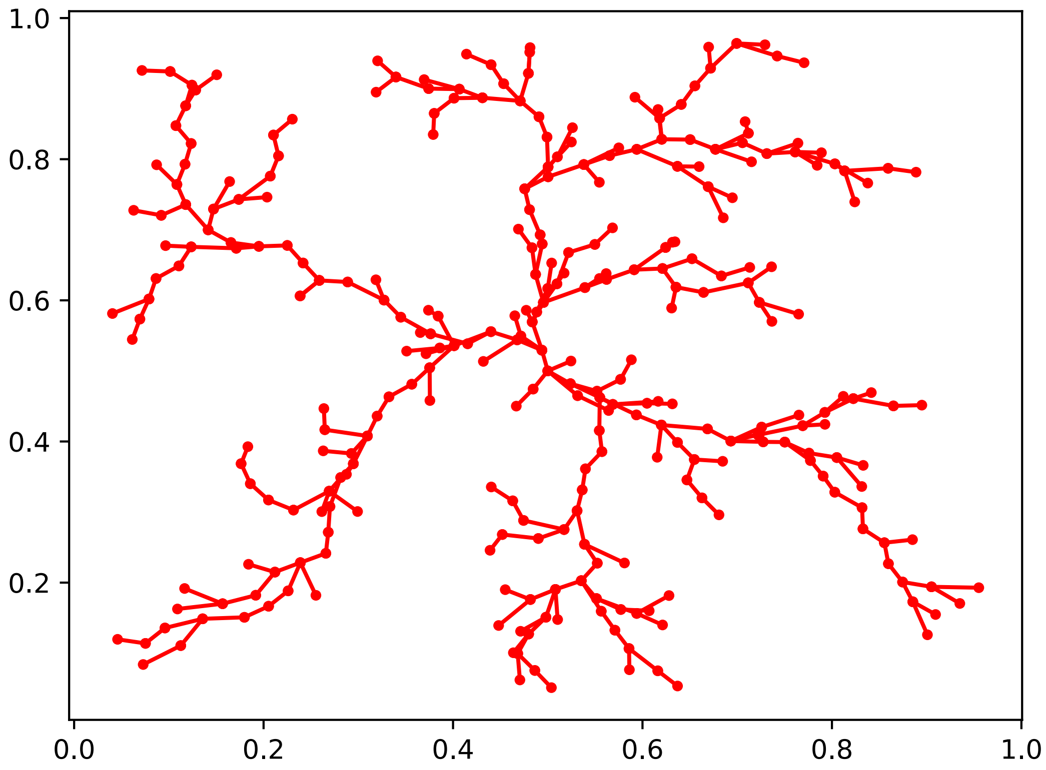
No. Why not?

The search space is not discretized. Thus, it can continue sampling and expanding the tree forever.

Optimality

If a solution **does not** exist, can A* theoretically tell you that?





Steps To Improve the Quality of RRT Solutions

```
RRT (q_start, q_goal, goalTolerance, maxTreeSize)
    tree = new Tree()
    while treeSize < maxTreeSize
        q_rand = SampleRandomConfig()
        q_near = Tree.findClosest(q_near)
        q_new = Expand(q_near, q_rand , stepsize)
        if q_new /* path is collision free */
            tree.addNode(q_new)
            tree.addEdge(q_near, q_new)
            if goalCheck(q_new, q_goal, goalTolerance)
                return Tree
    return null
```

Goal State q_{rand}

Every now and then (maybe 5%), make q_{rand} the goal state. If you extend a node in the tree towards the goal, probably a good thing.

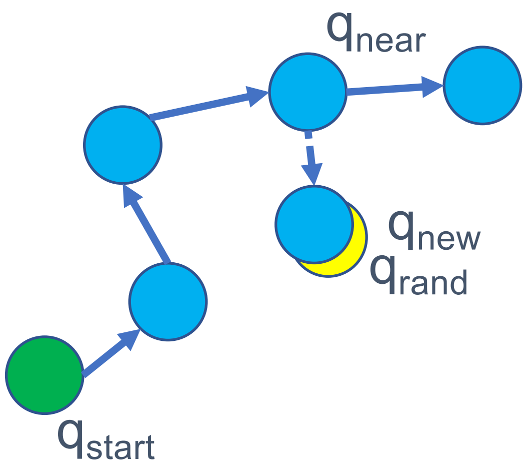
Keep Extending

- Keep expanding and creating new nodes at the step size interval in the direction of q_{rand}. until you reach q_{rand} or you encounter an obstacle.

RRT* -- A Path to Optimality

Idea: Improve the path costs during each iteration

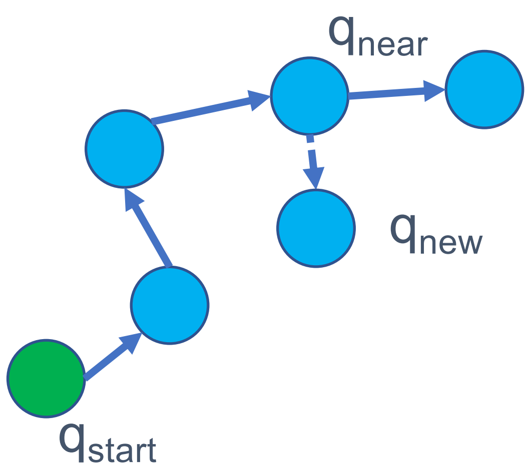
1. It records the distance each vertex has traveled from q_{start}
2. q_{new} is proposed to be wired into the tree as before. Before wiring q_{new} to q_{near}, an additional check is made. A neighborhood of points are examined to see if connecting q_{new} to any of them will result in a lower cost (that is, a shortest distance traveled from the root node).

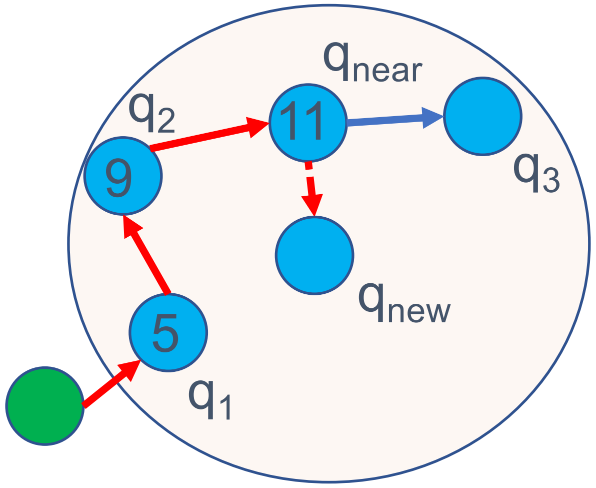


RRT* -- A Path to Optimality

Idea: Improve the path costs during each iteration

1. It records the distance each vertex has traveled from q_{start}
2. q_{new} is proposed to be wired into the tree as before. Before wiring q_{new} to q_{near}, an additional check is made. A neighborhood of points are examined to see if connecting q_{new} to any of them will result in a lower cost (that is, a shortest distance traveled from the root node).

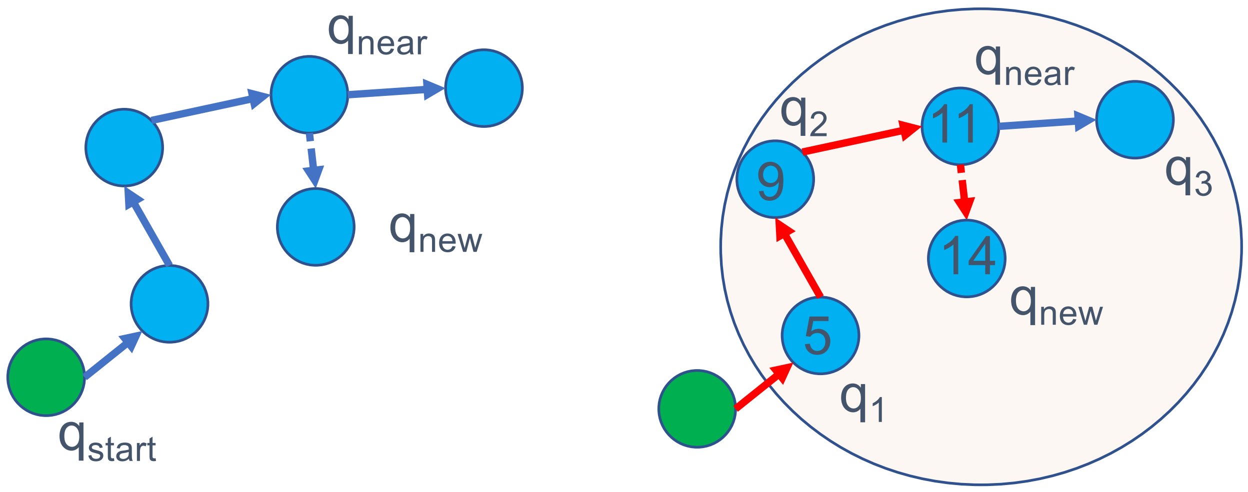




RRT* -- A Path to Optimality

Idea: Improve the path costs during each iteration

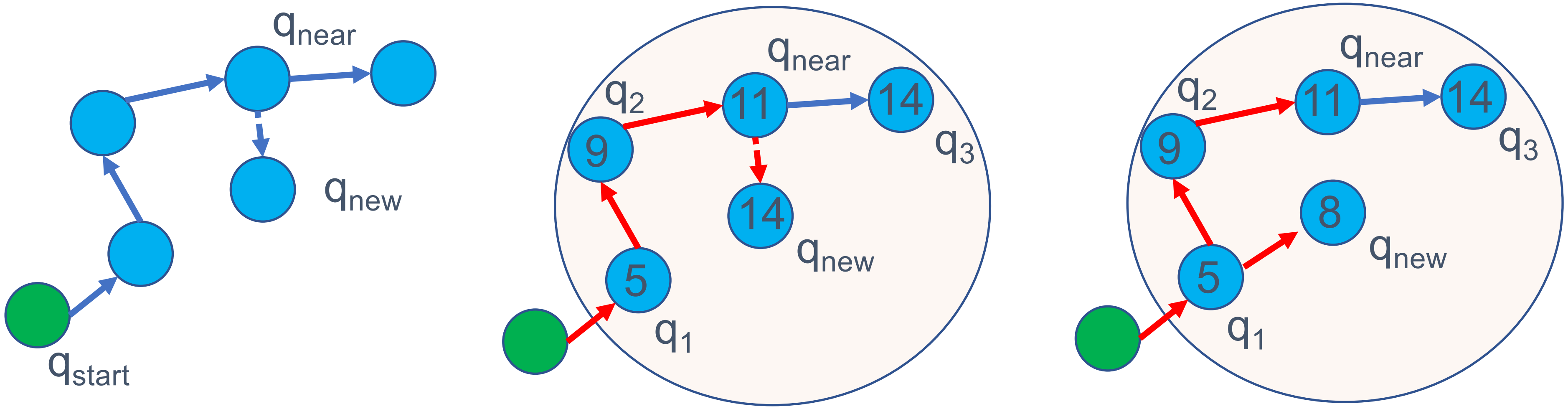
1. It records the distance each vertex has traveled from q_{start}
2. q_{new} is proposed to be wired into the tree as before. Before wiring q_{new} to q_{near}, an additional check is made. A neighborhood of points are examined to see if connecting q_{new} to any of them will result in a lower cost (that is, a shortest distance traveled from the root node).



RRT* -- A Path to Optimality

Idea: Improve the path costs during each iteration

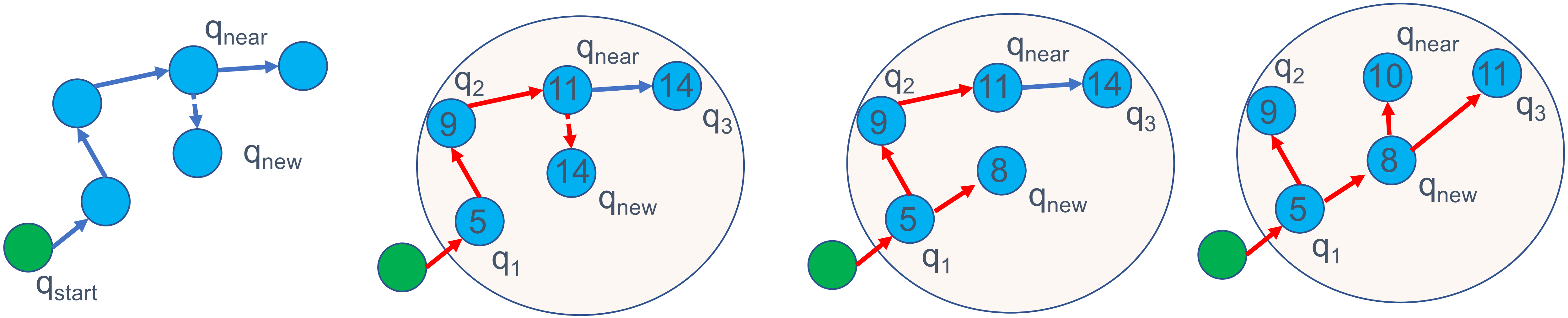
1. It records the distance each vertex has traveled from q_{start}
2. q_{new} is proposed to be wired into the tree as before. Before wiring q_{new} to q_{near}, an additional check is made. A neighborhood of points are examined to see if connecting q_{new} to any of them will result in a lower cost (that is, a shortest distance traveled from the root node).



RRT* -- A Path to Optimality

Idea: Improve the path costs during each iteration

1. It records the distance each vertex has traveled from q_{start}
2. q_{new} is proposed to be wired into the tree as before. Before wiring q_{new} to q_{near}, an additional check is made. A neighborhood of points are examined to see if connecting q_{new} to any of them will result in a lower cost (that is, a shortest distance traveled from the root node).
3. Check all nodes in the neighborhood. If their distance can be lowered by connecting through q_{new} instead of their existing parent, "rewire" the tree.



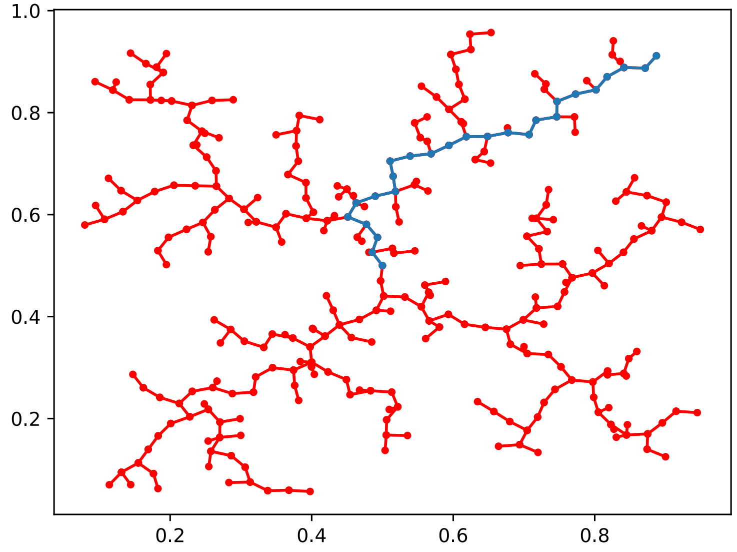
RRT* Optimality

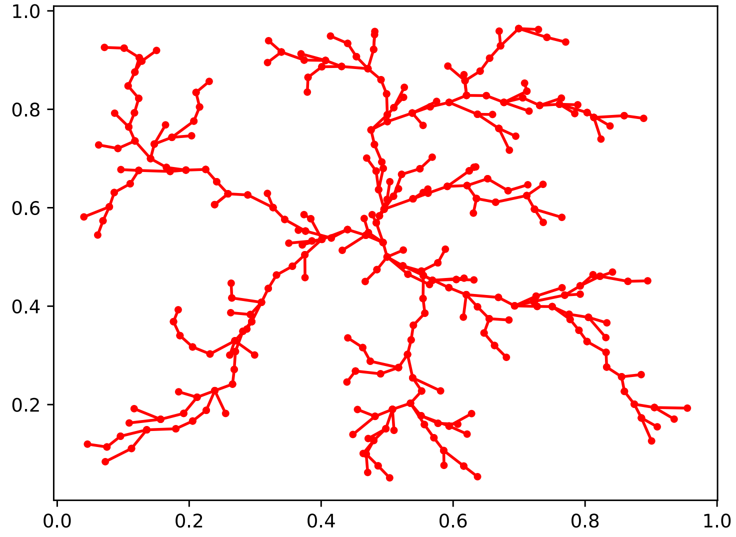
Does RRT converge to the optimal solution?

No, running RRT for a longer duration has no guarantees about convergence.

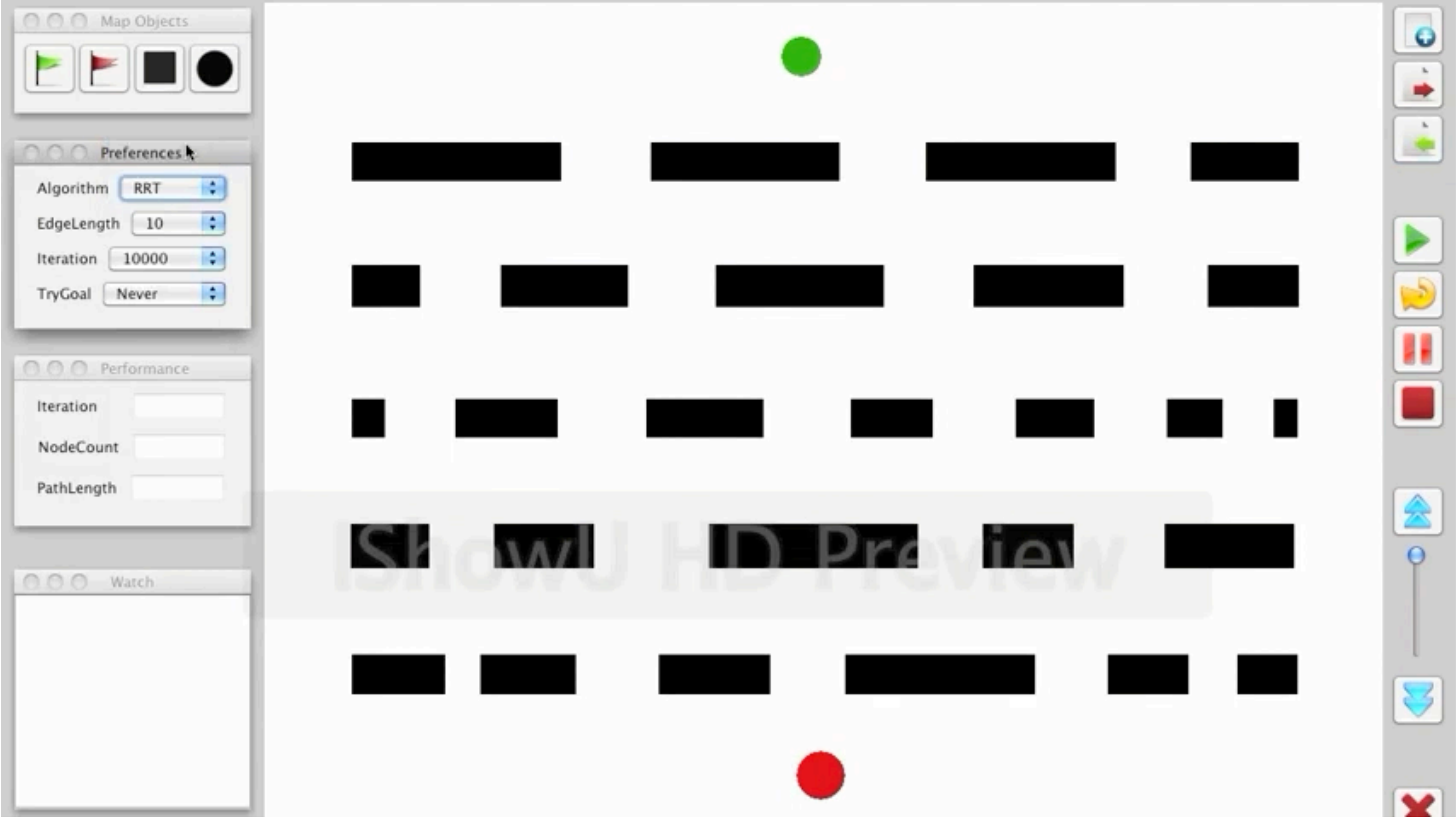
Does RRT* converge to the optimal solution?

Running RRT* will converge to the optimal solution.

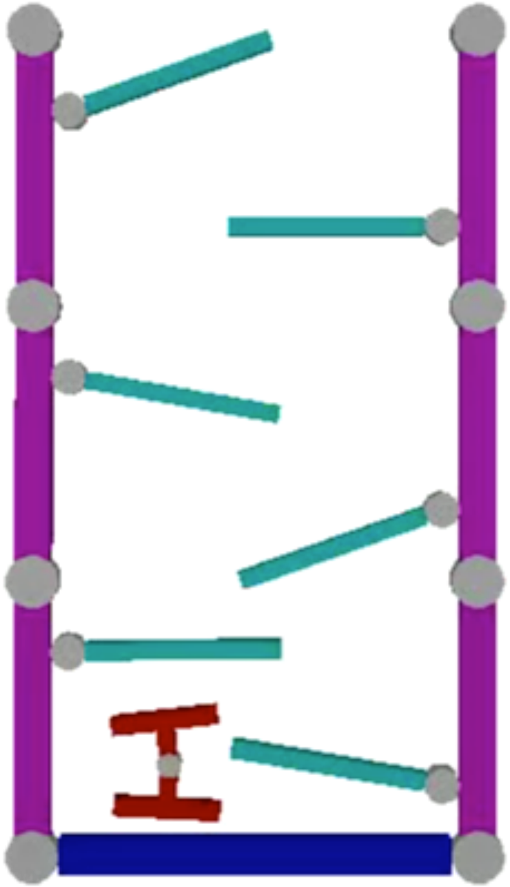


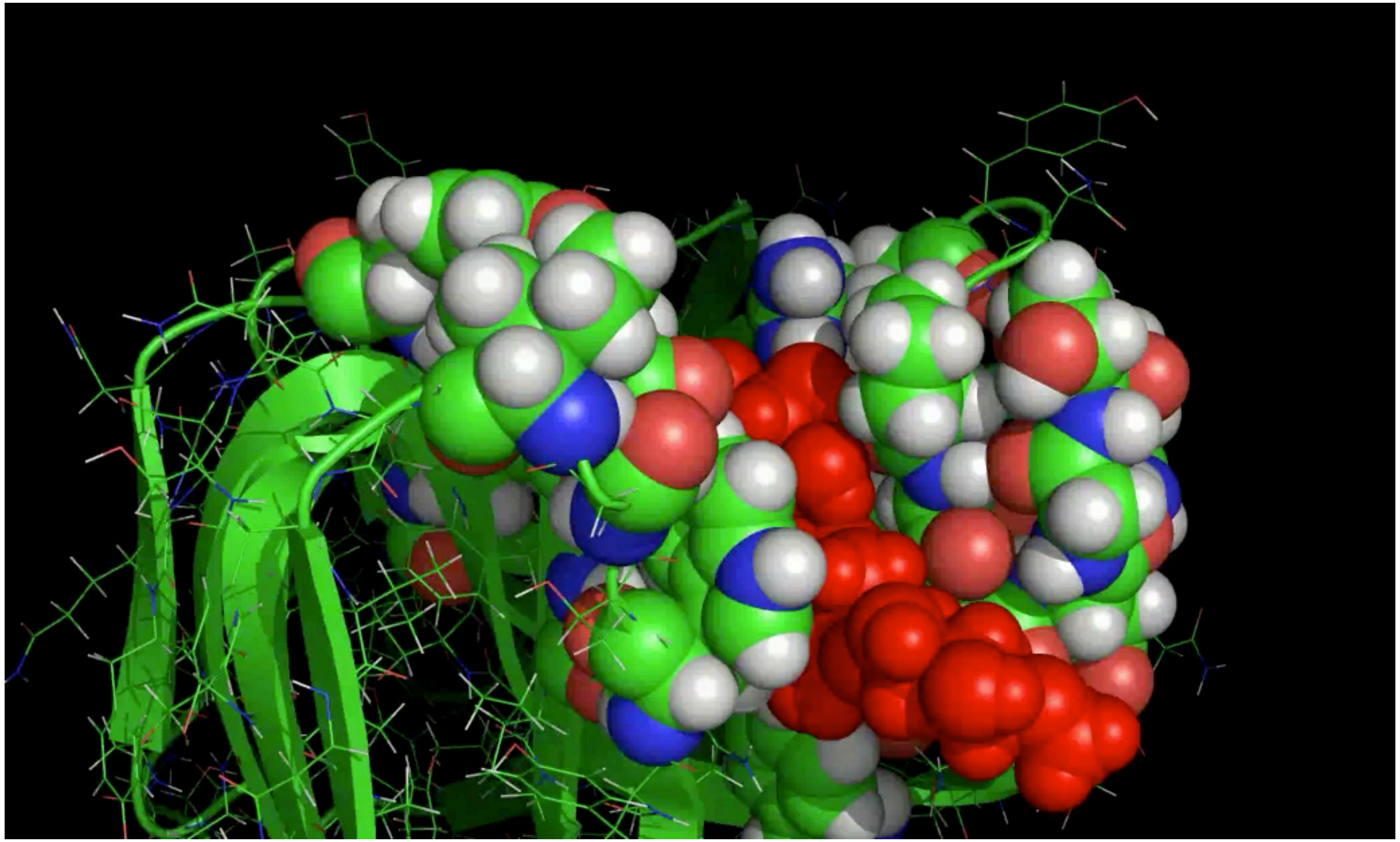


RRT Algorithm In Action



Varients of RRT (some examples)





Probabilistic Roadmap (PRM)

Idea: Build a graph/roadmap through the configuration space

While (g.get_node_count() < n)

q_{rand} = problem.random_state()

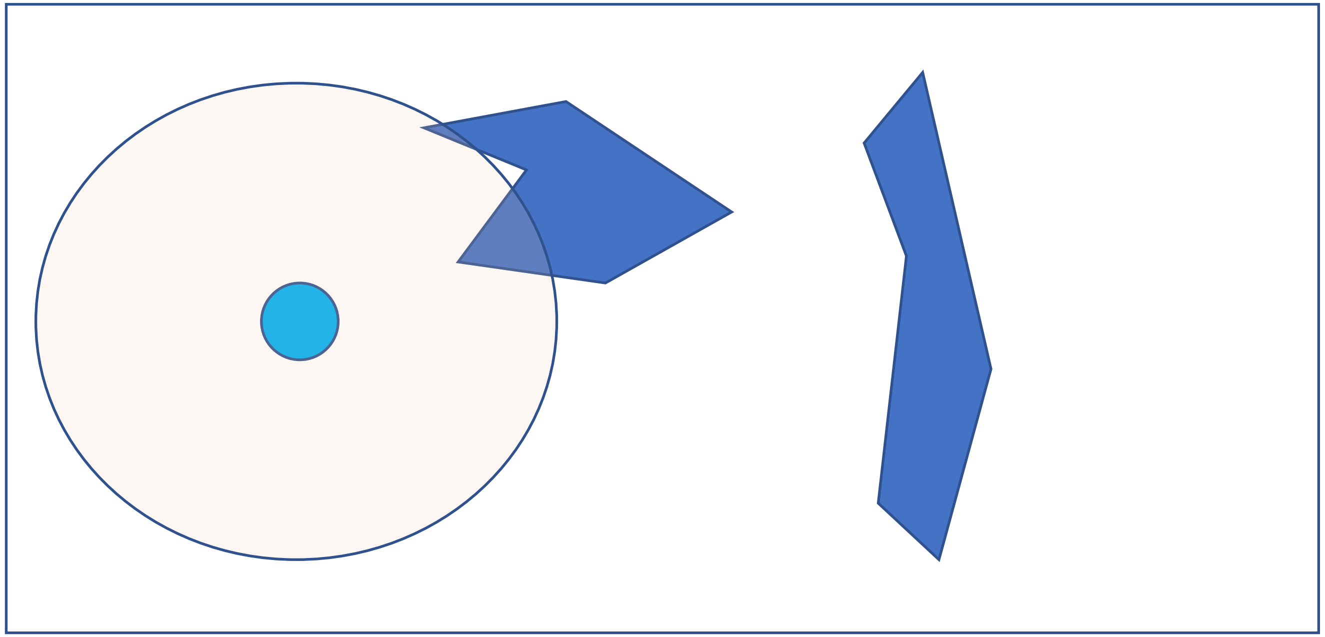
if collision_free(q_{rand})

g.add_node(q_{rand})

for node in neighbors (q_{rand}):

if problem.no_collision(q_{rand},node)

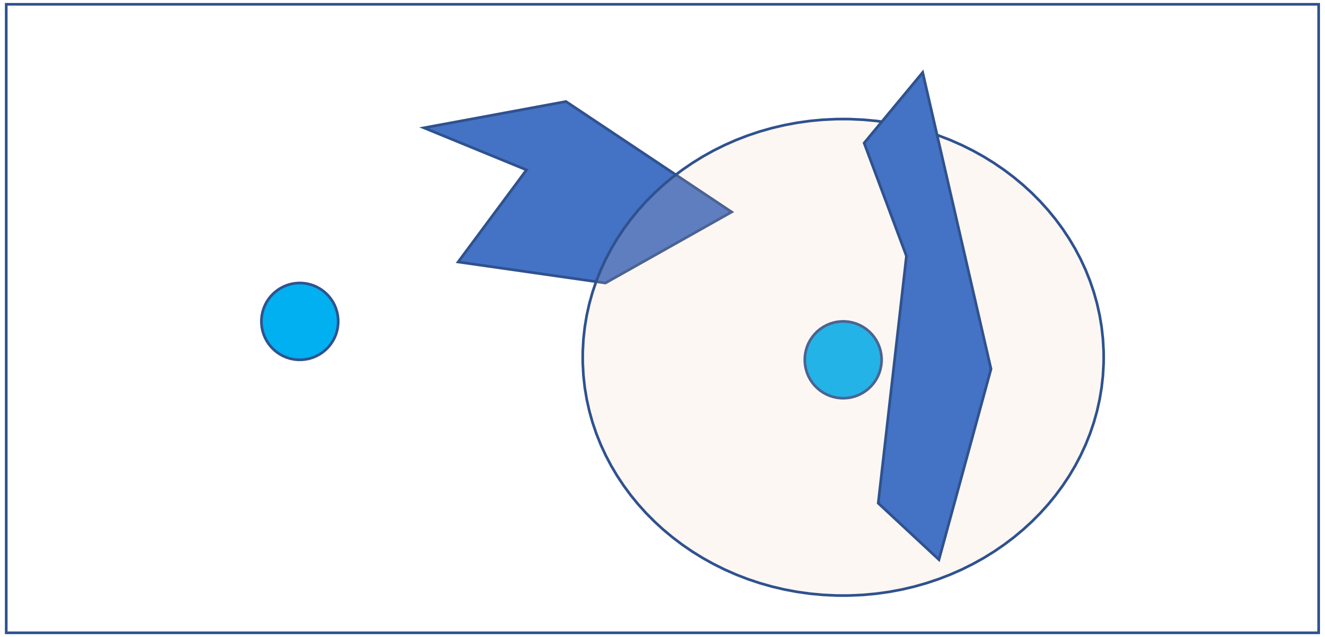
g.add_edge(problem.no_collision(q_{rand},node)



Probabilistic Roadmap (PRM)

Idea: Build a graph/roadmap through the configuration space

```
While (g.get_node_count() < n)
    q_rand = problem.random_state()
    if collision_free(q_rand)
        g.add_node(q_rand)
        for node in neighbors (q_rand):
         if problem.no_collision(q_rand,node)
                 g.add_edge(problem.no_collision(q_rand,node)
```



Probabilistic Roadmap (PRM)

Idea: Build a graph/roadmap through the configuration space

While (g.get_node_count() < n)

q_{rand} = problem.random_state()

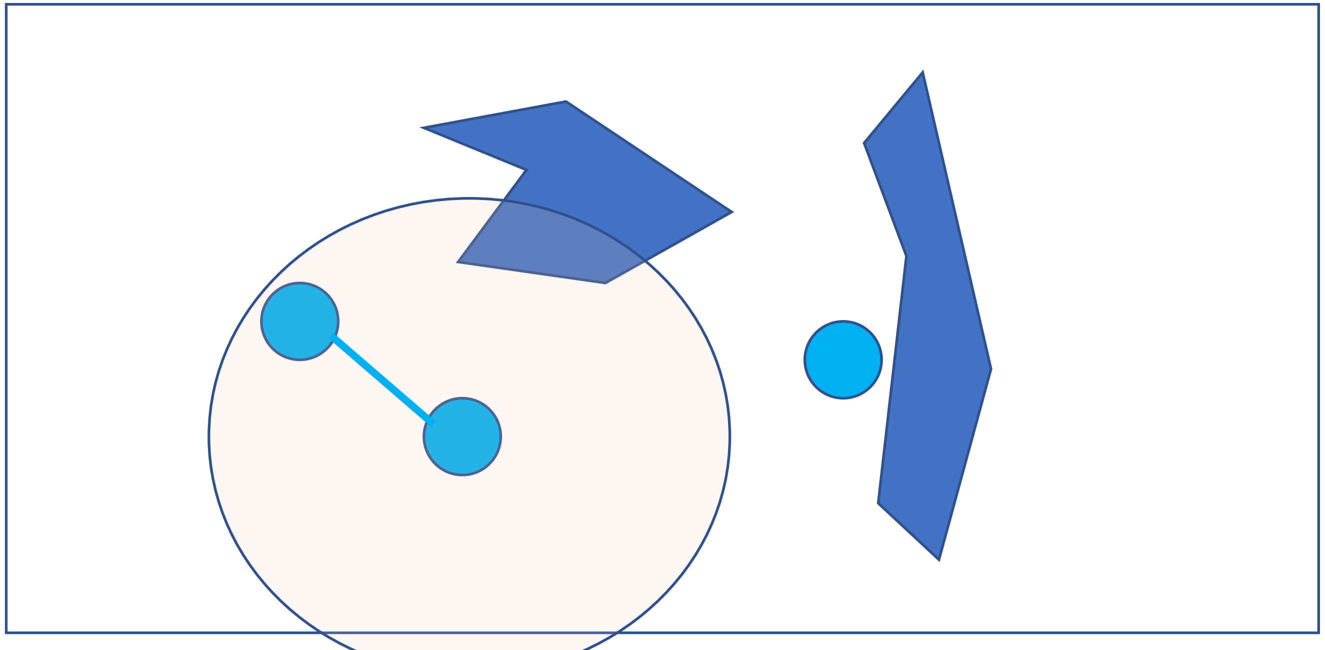
if collision_free(q_{rand})

g.add_node(q_{rand})

for node in neighbors (q_{rand}):

if problem.no_collision(q_{rand},node)

g.add_edge(problem.no_collision(q_{rand},node)



Probabilistic Roadmap (PRM)

Idea: Build a graph/roadmap through the configuration space

While (g.get_node_count() < n)

    q_{rand} = problem.random_state()

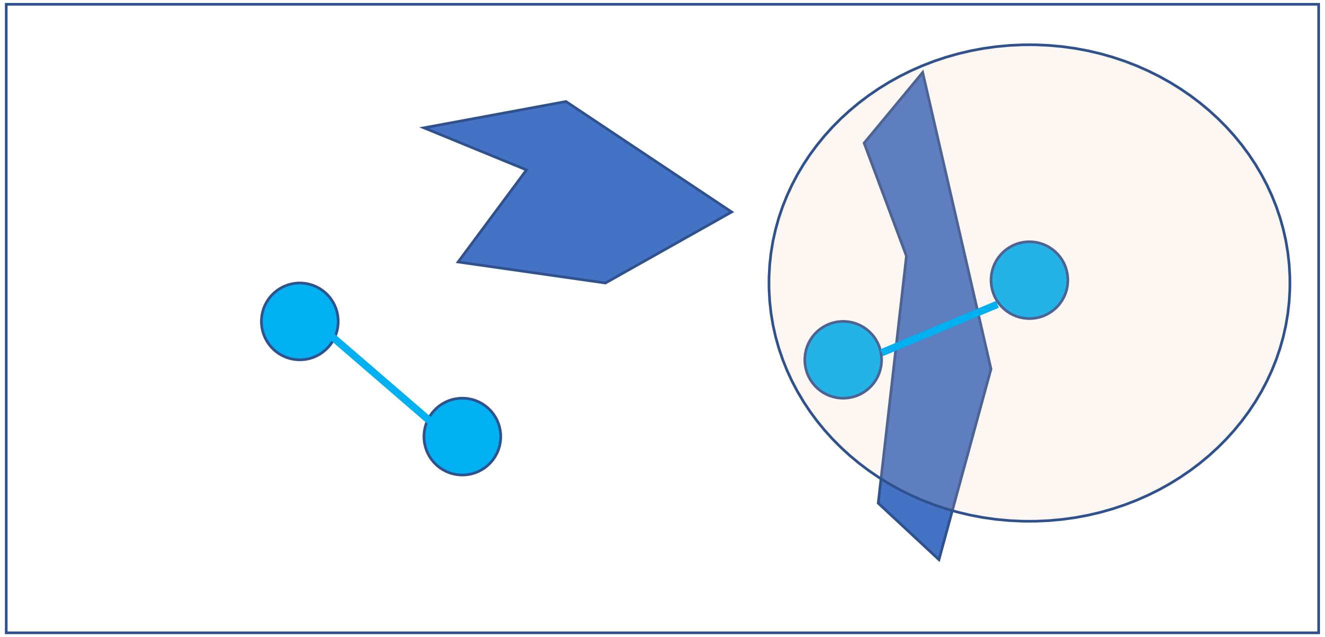
    if collision_free(q_{rand})

        g.add_node(q_{rand})

        for node in neighbors (q_{rand}):

         if problem.no_collision(q_{rand},node)

                g.add_edge(problem.no_collision(q_{rand},node)



Probabilistic Roadmap (PRM)

Idea: Build a graph/roadmap through the configuration space

While (g.get_node_count() < n)

    q_{rand} = problem.random_state()

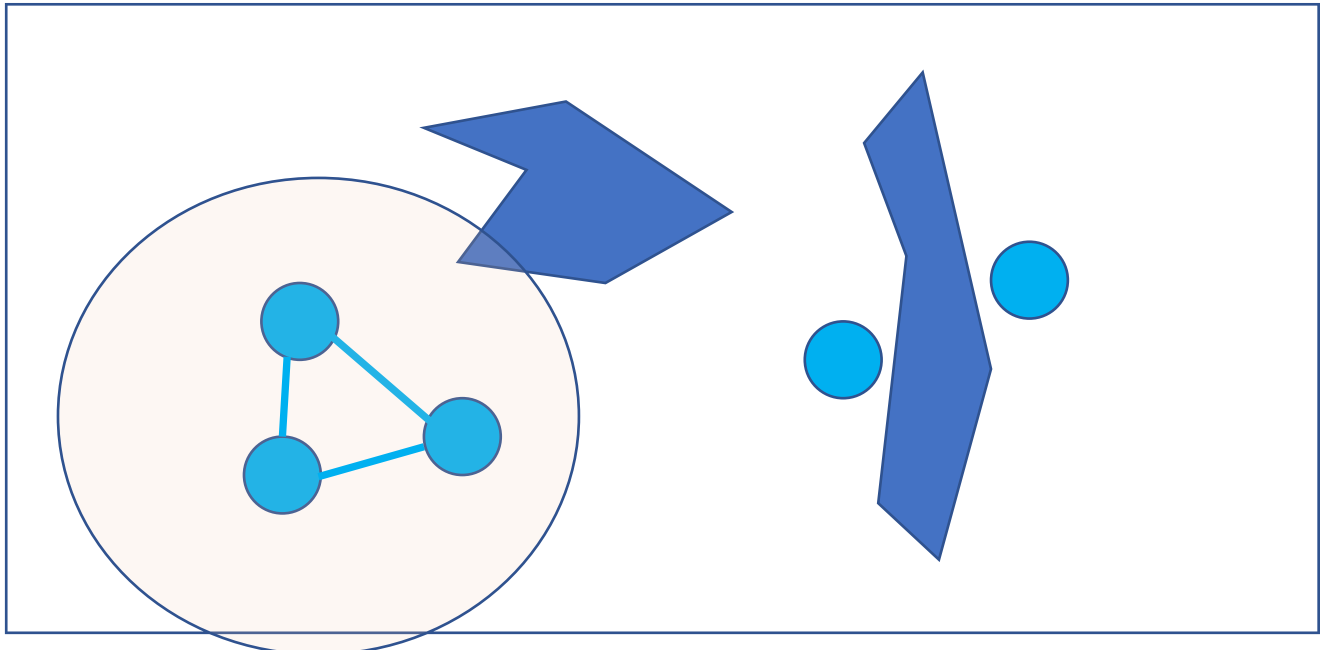
    if collision_free(q_{rand})

        g.add_node(q_{rand})

        for node in neighbors (q_{rand}):

            if problem.no_collision(q_{rand},node)

                g.add_edge(problem.no_collision(q_{rand},node)



Probabilistic Roadmap (PRM)

Idea: Build a graph/roadmap through the configuration space

While (g.get_node_count() < n)

    q_{rand} = problem.random_state()

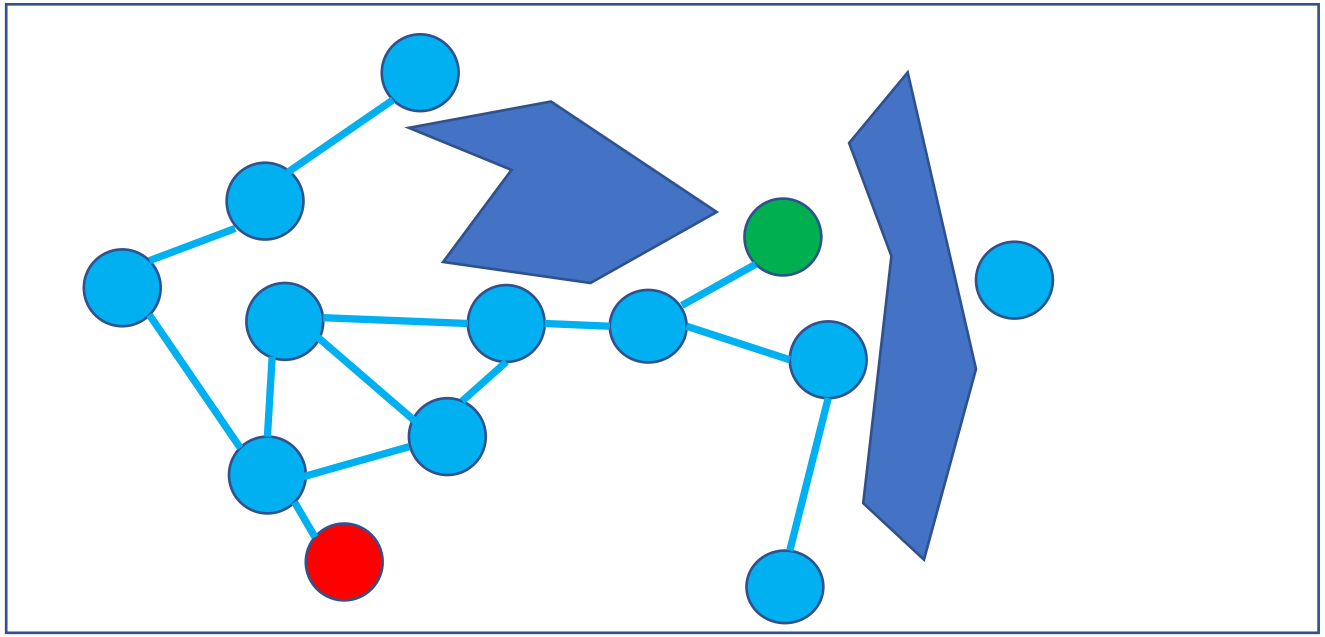
    if collision_free(q_{rand})

        g.add_node(q_{rand})

        for node in neighbors (q_{rand}):

            if problem.no_collision(q_{rand},node)

                g.add_edge(problem.no_collision(q_{rand},node)



Probabilistic Roadmap (PRM)

Idea: Build a graph/roadmap through the configuration space

While (g.get_node_count() < n)

  q_{rand} = problem.random_state()

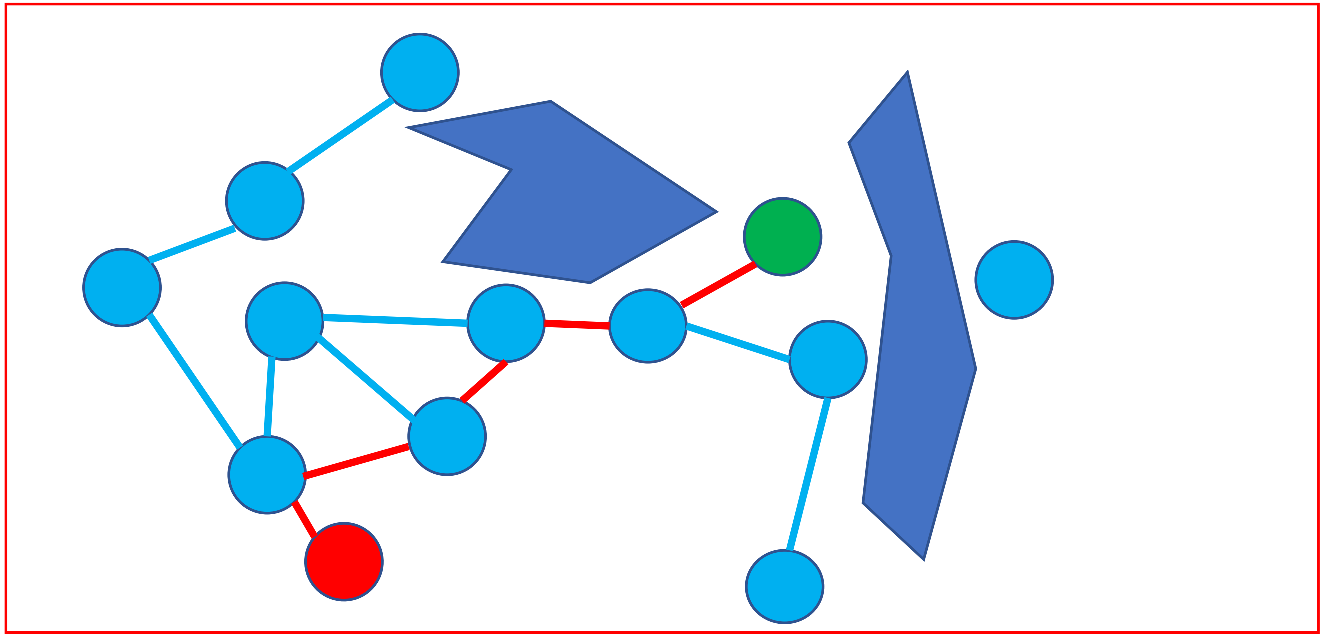
  if collision_free(q_{rand})

    g.add_node(q_{rand})

    for node in neighbors (q_{rand}):

     if problem.no_collision(q_{rand},node)

       g.add_edge(problem.no_collision(q_{rand},node)



Notes on Probabilistic Approaches

Computationally Demanding Tasks?

Neighborhood and q_{near} identification (as graph grows). For robots with many DOFs, this is **O**(n) per search.

When to stop?

Program would run infinitely when no solution exists.