Collision Detection and Planning

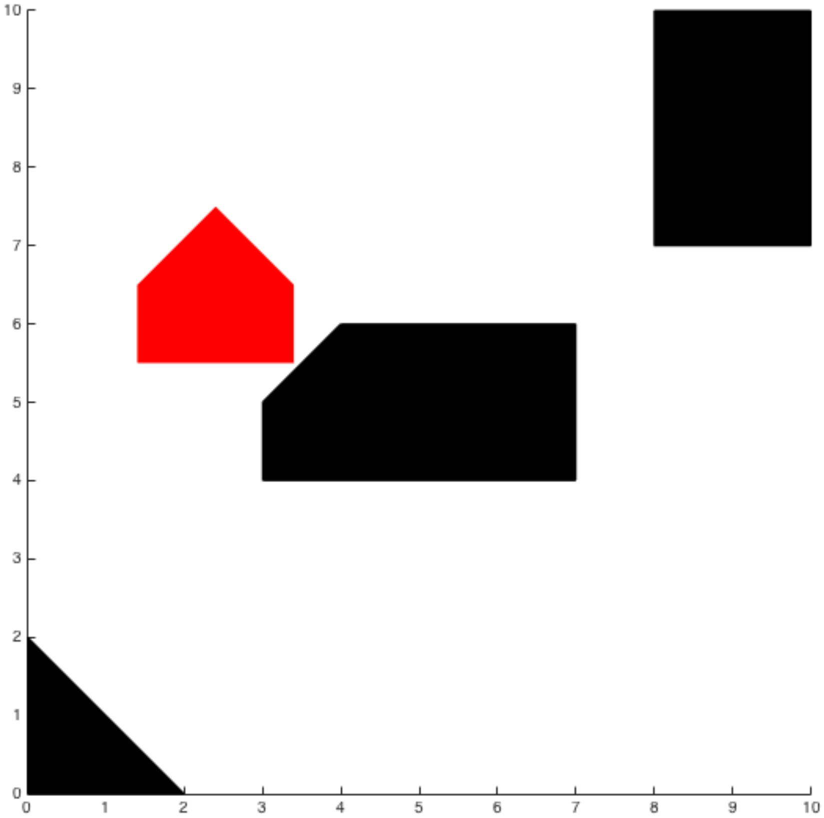
SECTION 2.6

- Polygonal obstacles are convenient to work with because they provide an explicit description of the configuration space obstacles.
- Oftentimes we do not have this luxury and the obstacles are instead defined implicitly by a collision function.

Collision Detection Function

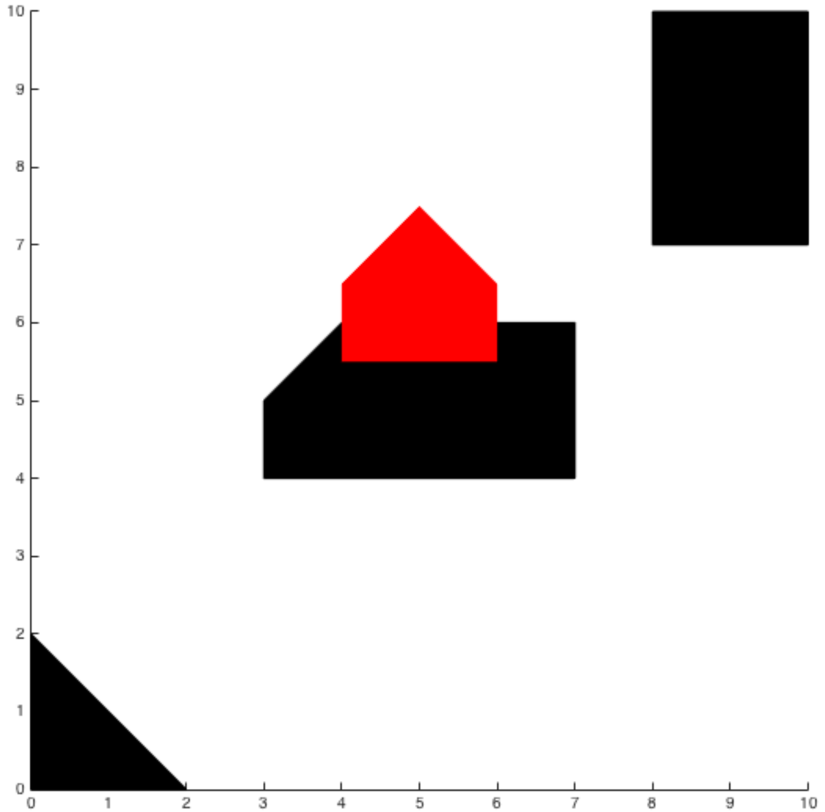
- Let x denote the coordinates of a point in configuration space.
- CollisionCheck (x) should return 0 if x is in freespace and 1 if x results in a collision with the obstacles.

Collision Detection



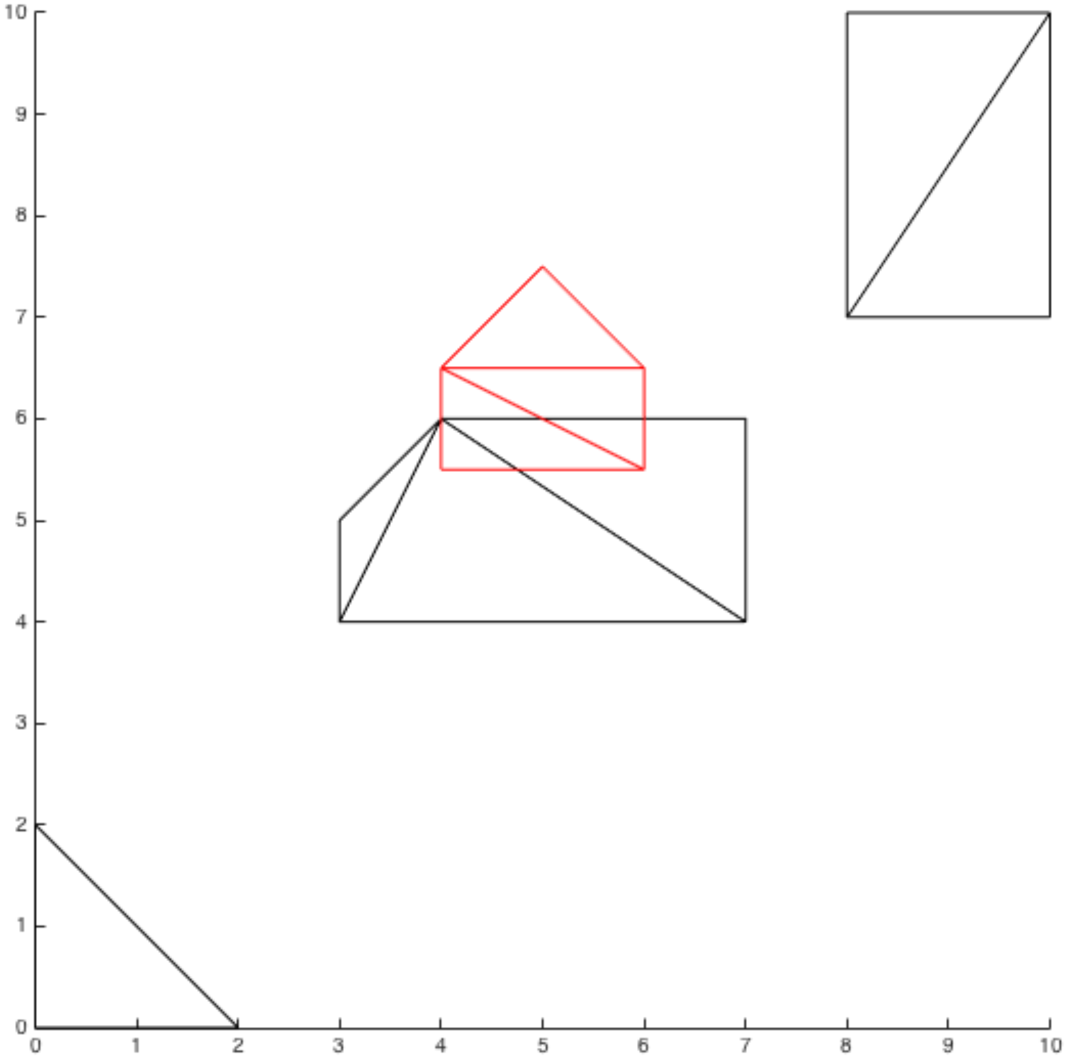
CollisionCheck (x) -> 0

Collision Detection



CollisionCheck (x) -> 1

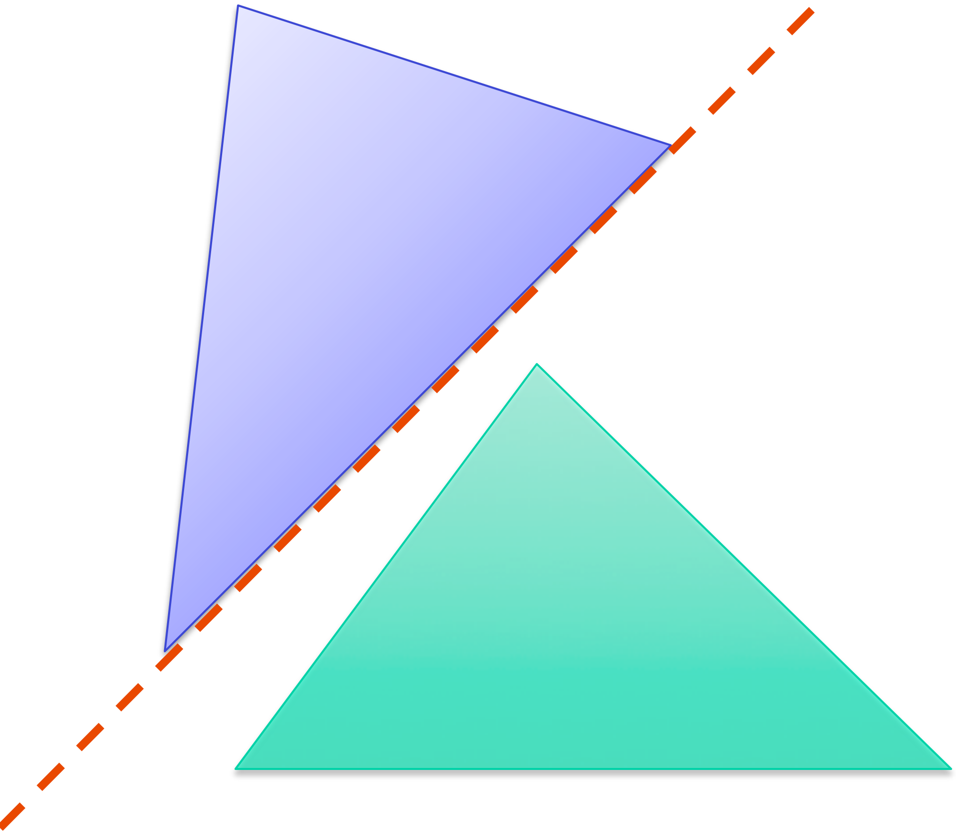
Collision Detection



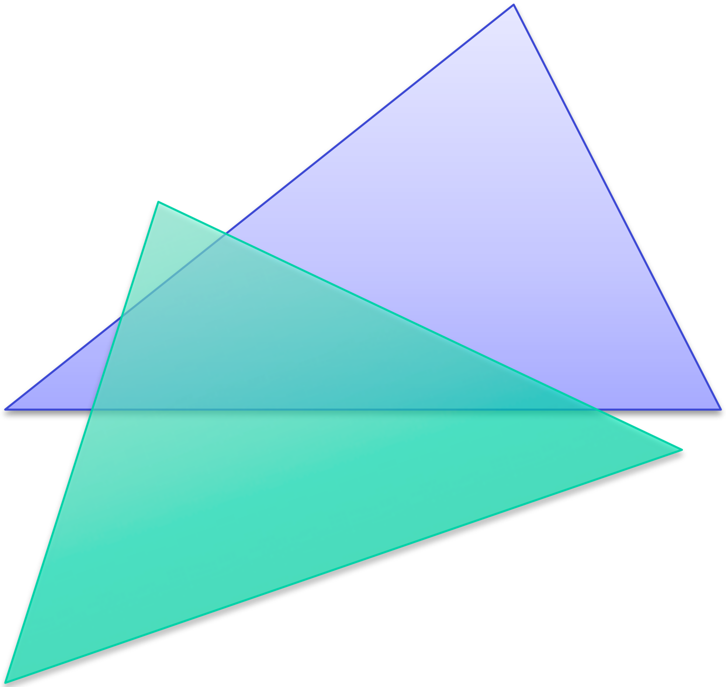
- **Deciding whether the robot and the obstacle intersect is now a matter of determining whether any of the robot triangles overlap any of the obstacle triangles.**

- Here we can make use of the fact that triangles are convex polygons. In this circumstance it means that we can test whether two triangles intersect by checking all of the sides on both triangles and testing whether that side acts as a separating line where all of the points from one triangle lie on one side and all those from the other lie on the opposite side.
- If you can find such a separating edge you have proved that the triangles don’t intersect, if not you can conclude that they do.

Testing Triangles for Overlap



Separating Edge -> No Overlap

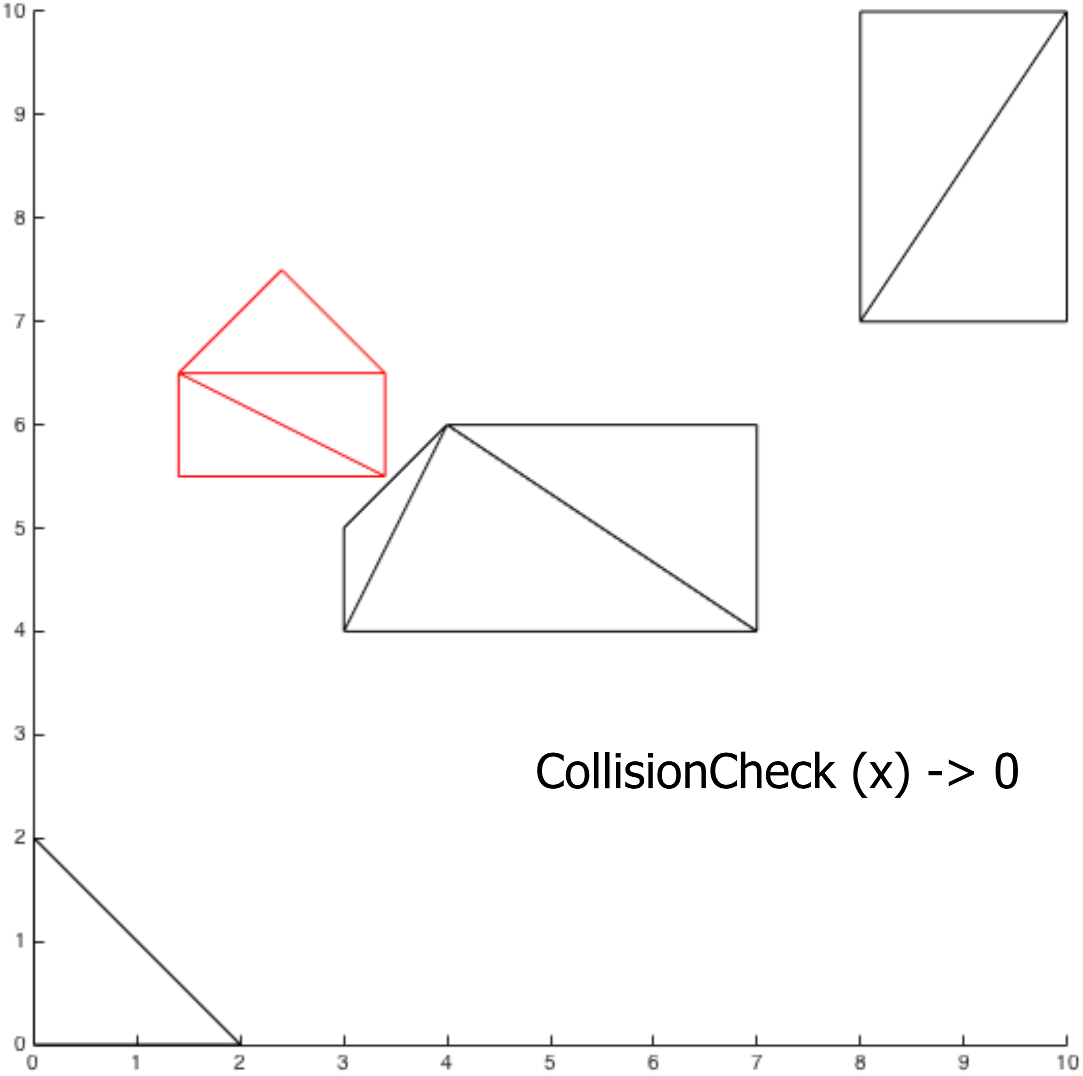


No Separating Edge -> Overlap

- This idea of finding a separating line or plane actually generalizes to higher dimension. For instance if you have convex polygons in three dimensions like boxes or pyramids you can check for collision by testing if any of the faces act as separating planes.

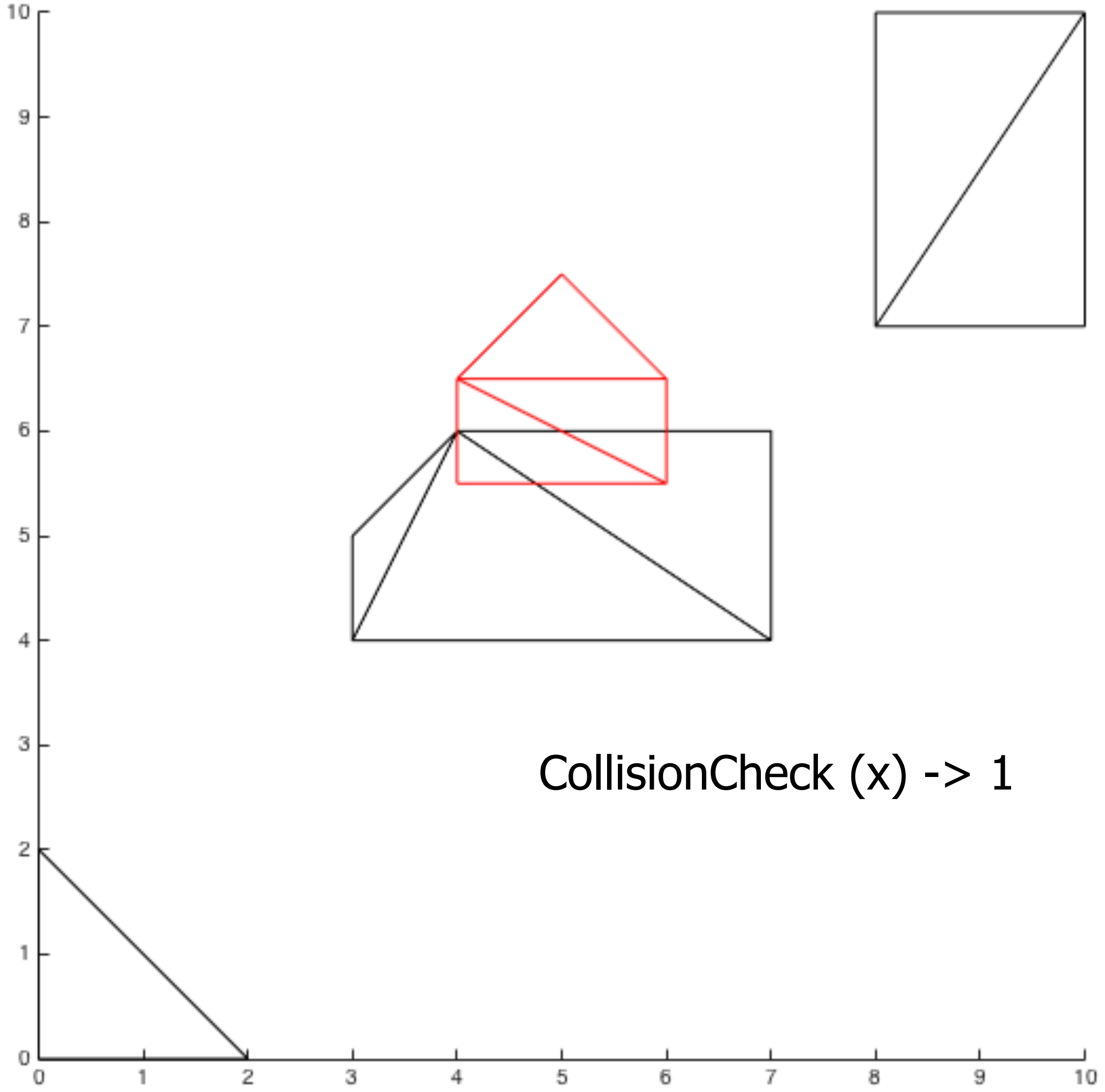
- In the cases that we have considered so far the robots and the obstacles are basically composed of simple polygons, so to test for collision we first transform the robot according to the configuration space parameters and then test for collision between the robot components and the obstacles.

Collision Checking



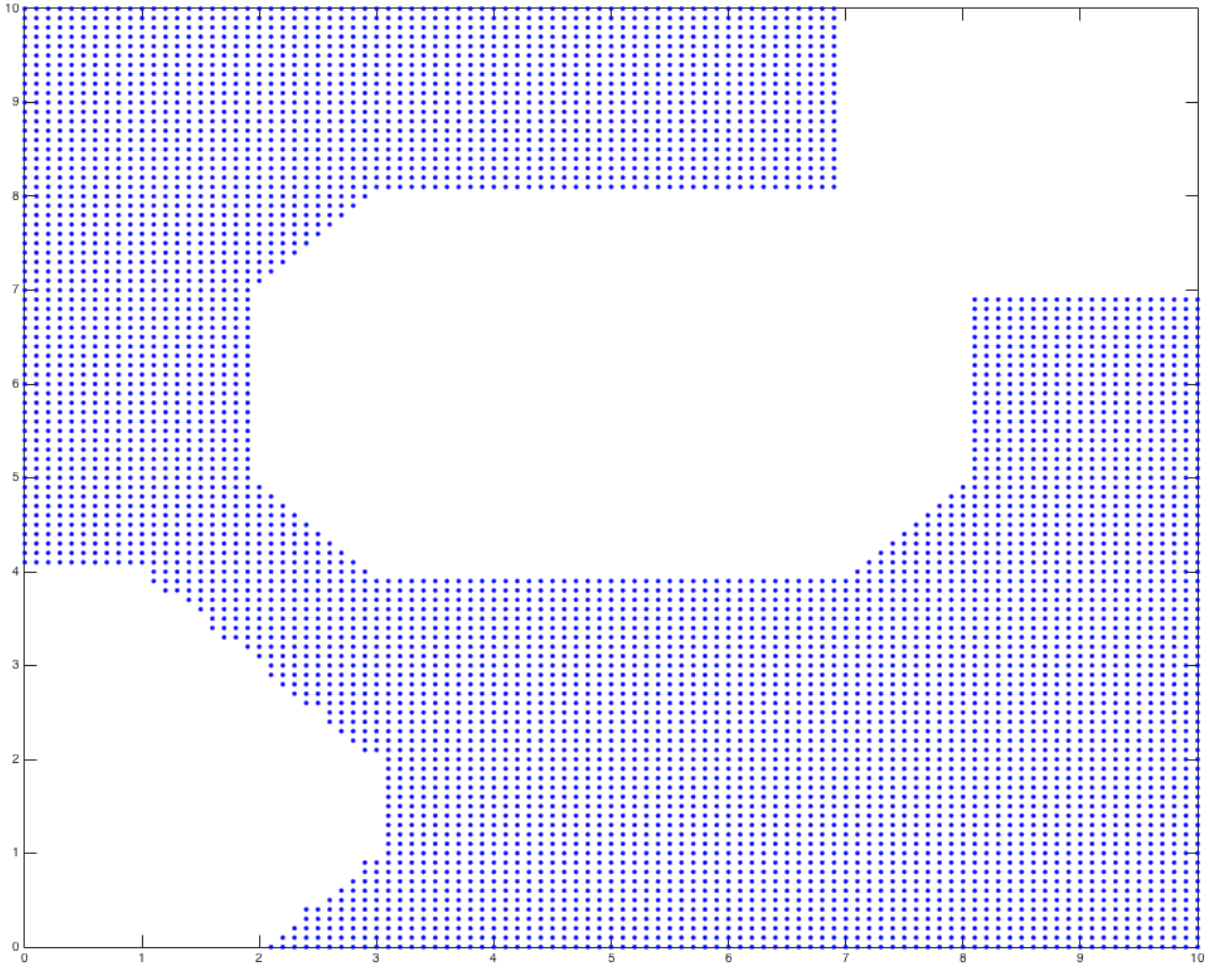
CollisionCheck (x) -> 0

Collision Checking

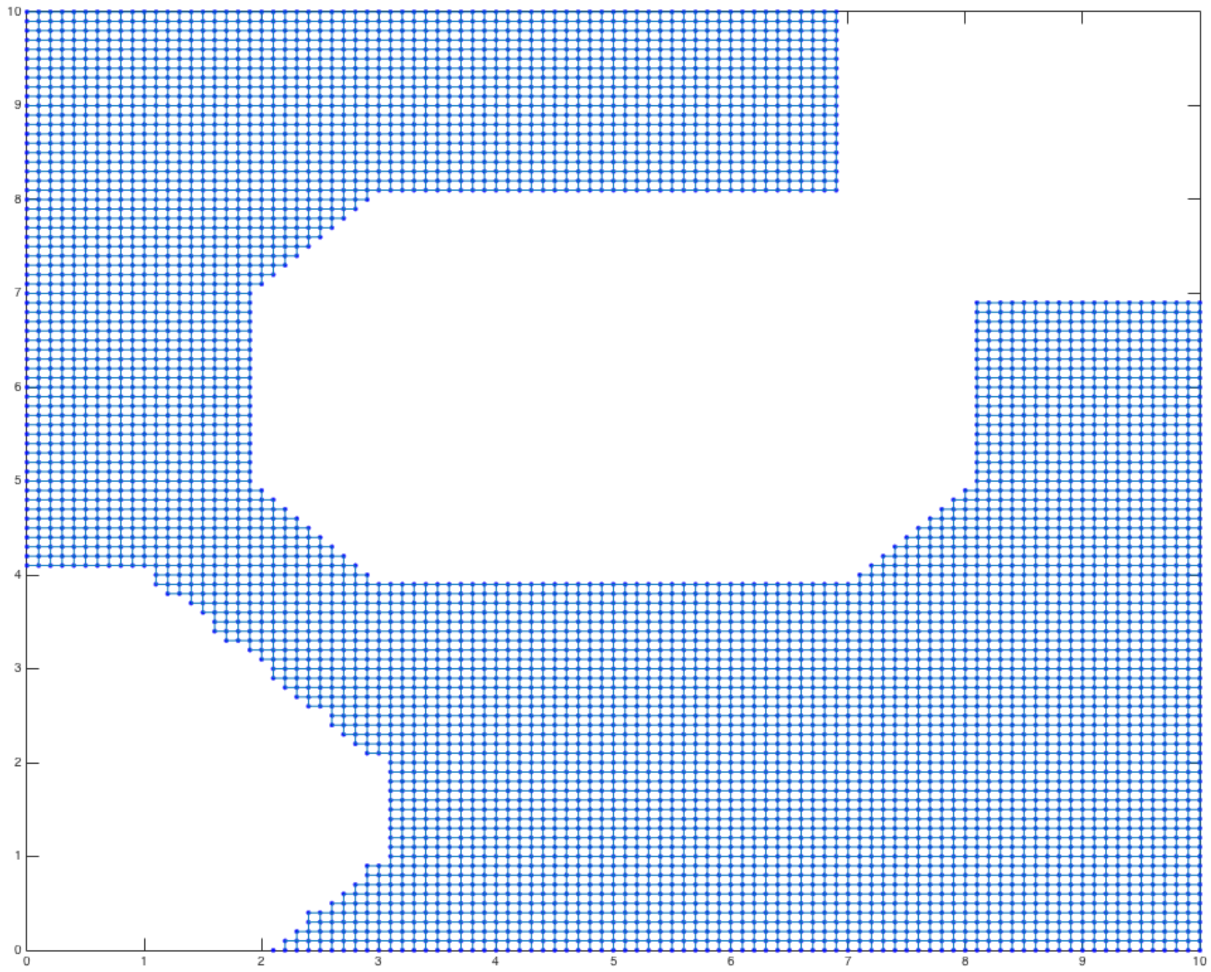


CollisionCheck (x) -> 1

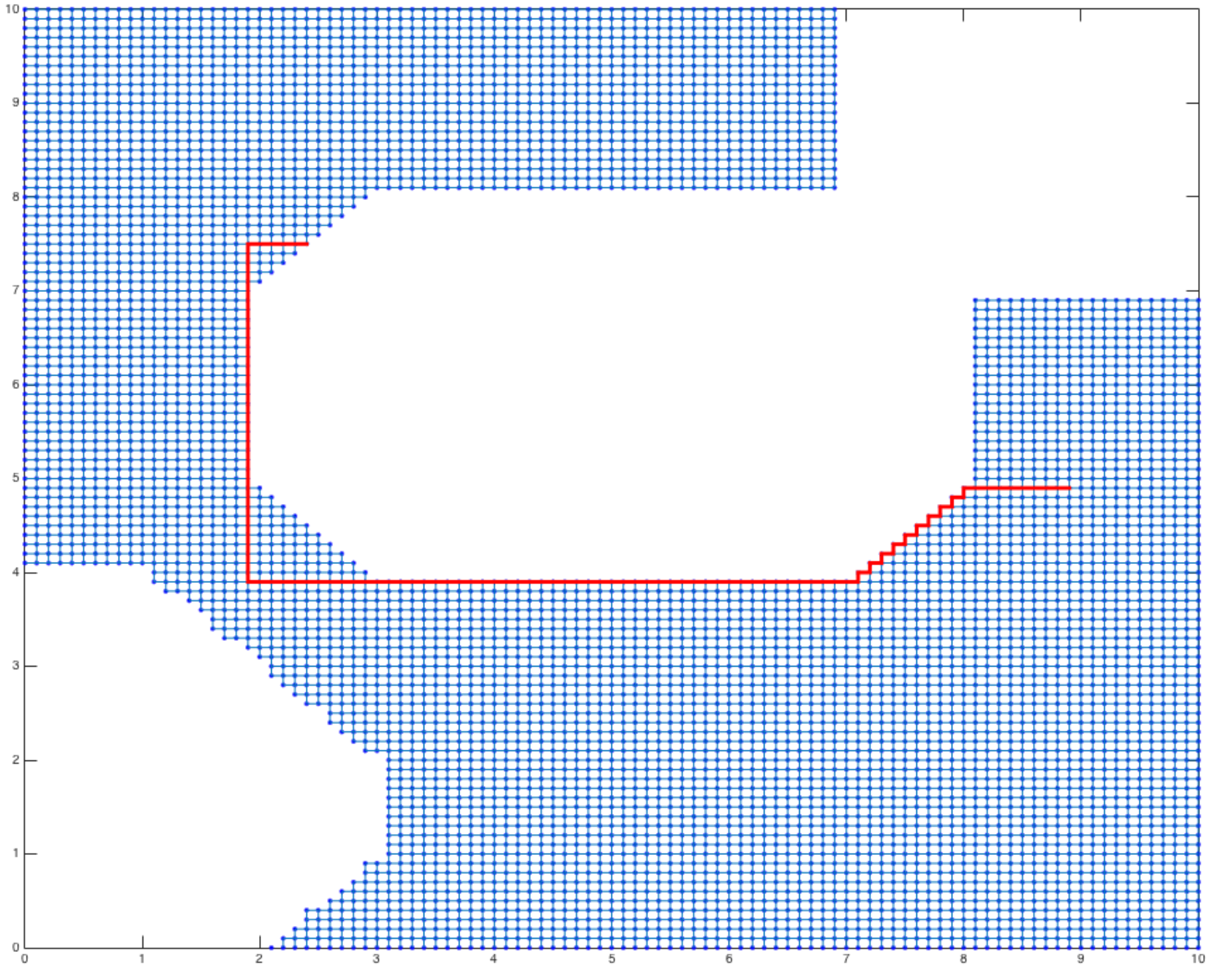
Sampled Points in freespace



Graph Made from Sampled Points



Path through graph


10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
10