ELEG4701

Intelligent Interactive Robot Practice

Lab 8: Lidar-based Navigation for Mobile Robots

Jiewen Lai

Research Assistant Professor

EE, CUHK

jiewen.lai@cuhk.edu.hk

Today's Agenda

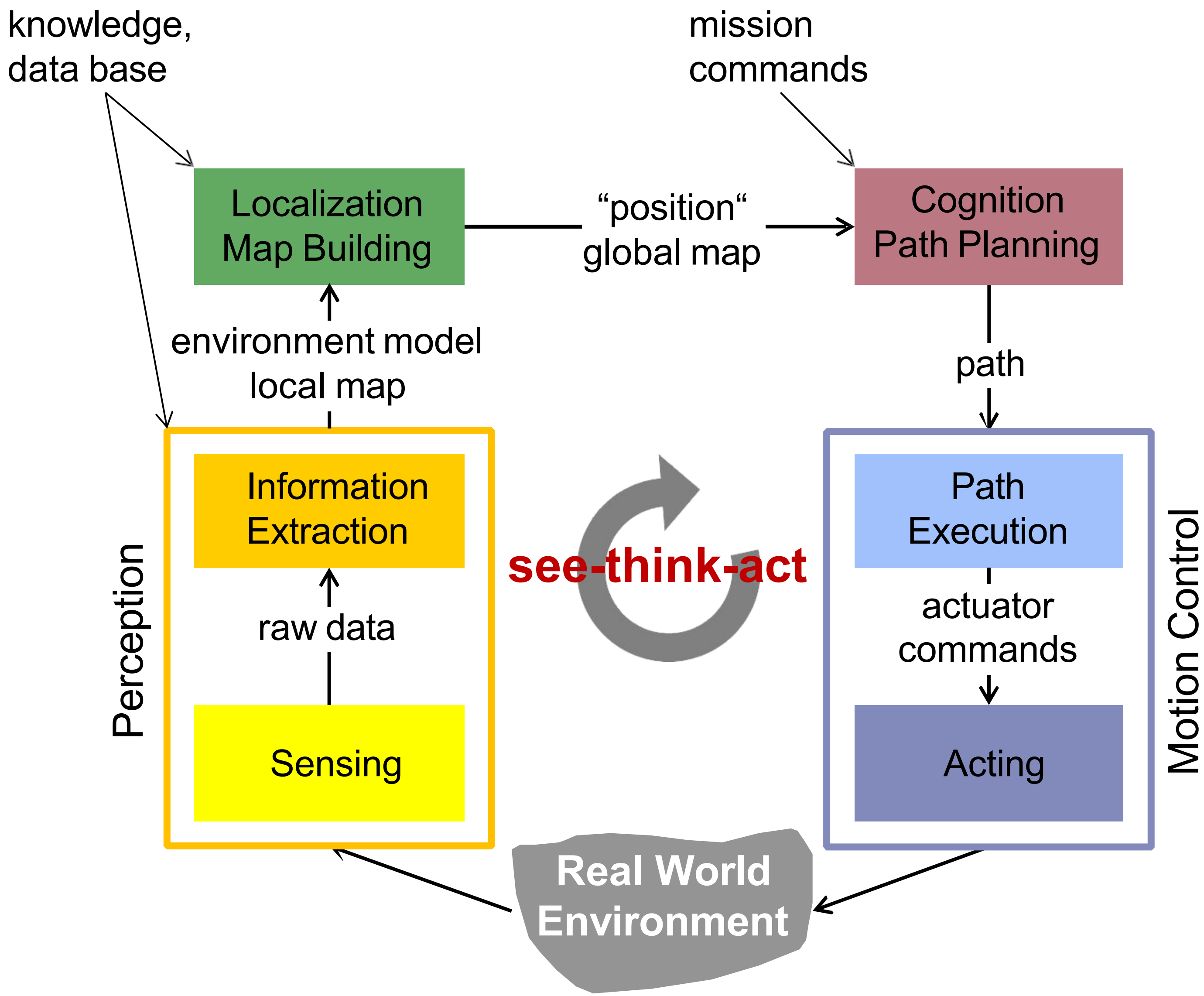
Lecture

1. Lidar
2. Rapidly exploring Random Tree (RRT)

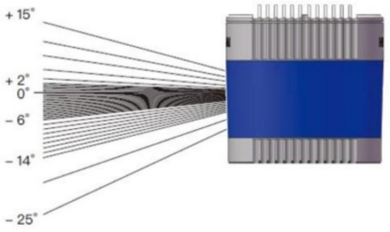
Tutorial

1. Lab Sheet 8

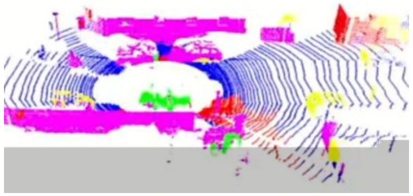
Mobile Robot Control Scheme



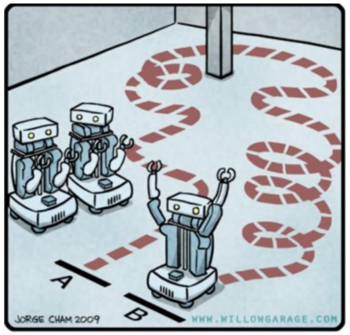
Navigation



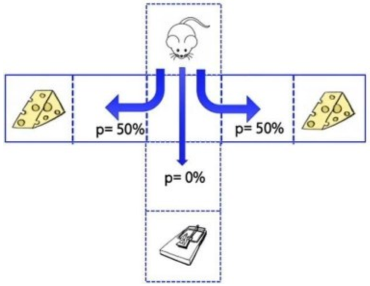
Lidar Signals



Perception



Motion



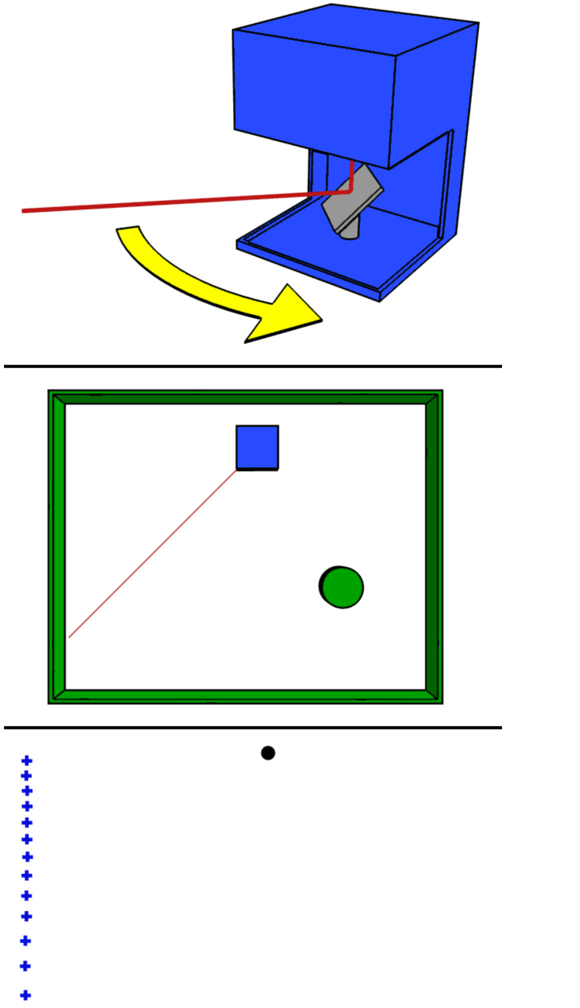
Policy

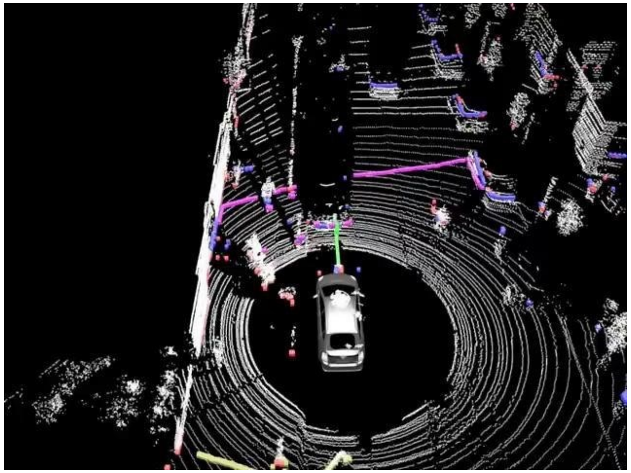
Navigation is concerned with finding the way to a desired destination

Lidar: Perception

Lidar / LIDAR / LiDAR / LADAR

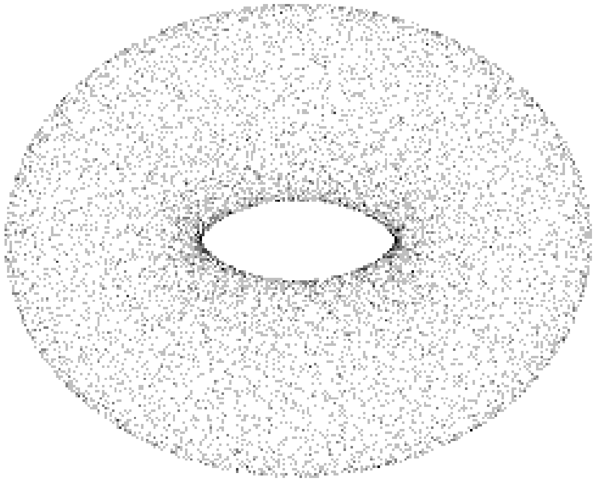
- **Li**ght **d**etection **a**nd **r**anging
- **L**aser **i**maging, **d**etection, **a**nd **r**anging
- A method for determining ranges by targeting an object or a surface with a laser and measuring the time for reflected light to return to the receiver.



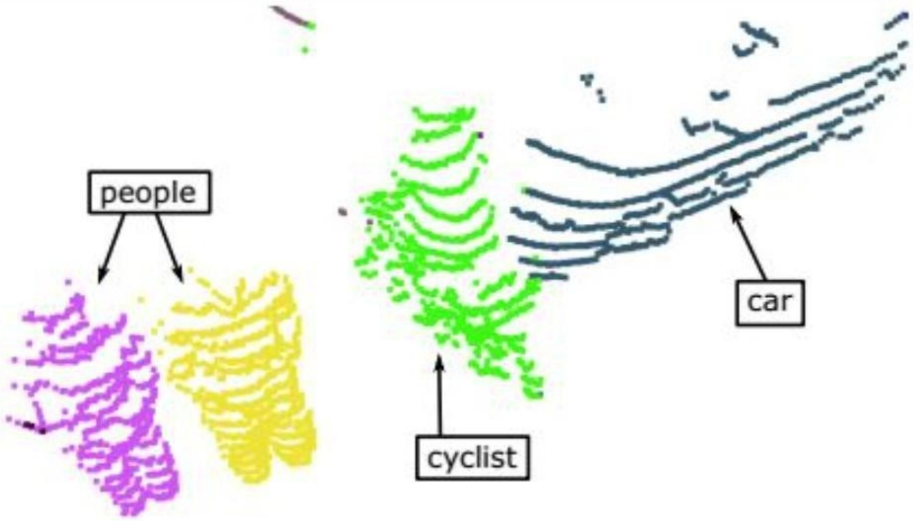


Lidar: Perception

Sensing of Lidar: **Point Clouds**

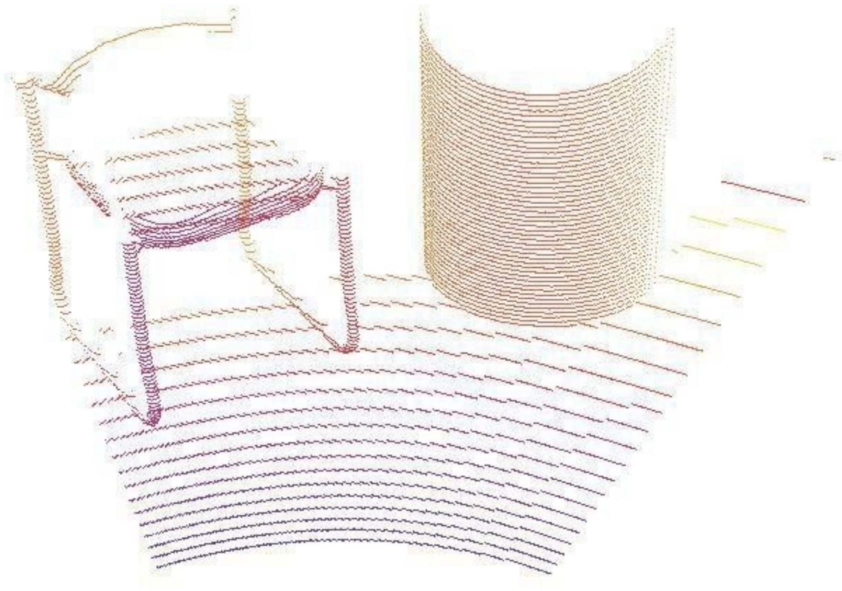


Dense point cloud



Sparse point cloud

Lidar: Perception

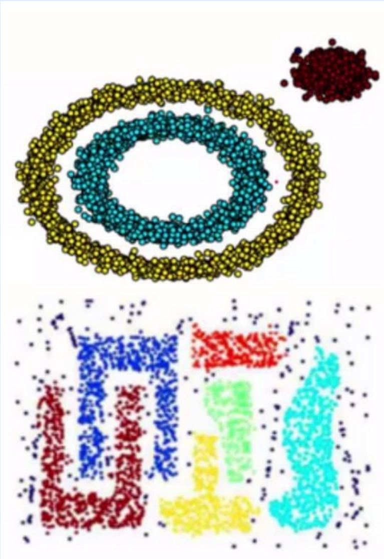


Which part is ground?

Which part is a chair?

Lidar: Observation

Point Clouds → Feature



An example rule:

```
for p in pointClouds:

   p.z>0?       → object
   p.z<=0?      → ground
   Dot(v,p)<0? → back
   Dot(v,p)>0? → front
```

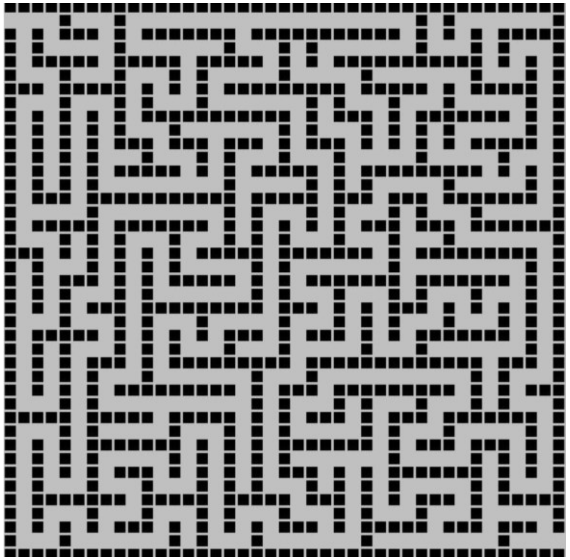
Machine Learning
Need Math (x)

Deep Learning
Need Big Data (x)

Rule Based
Beginner Friendly (√)

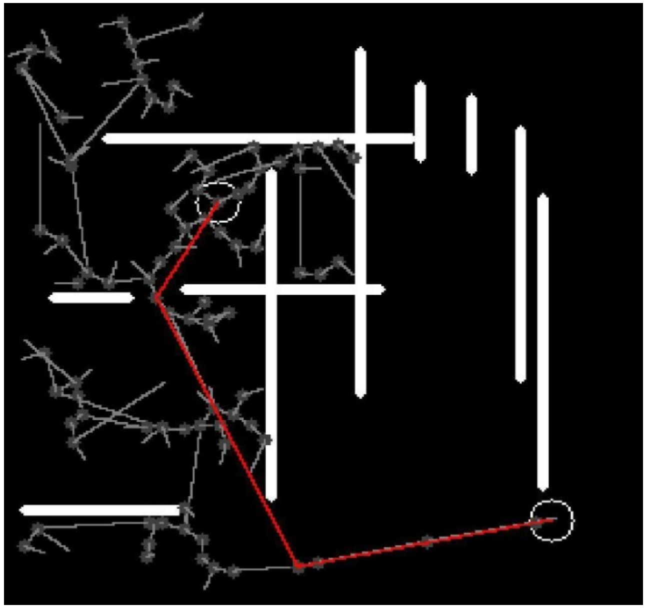
Lidar: Planning

Motion Planning



Path Planning

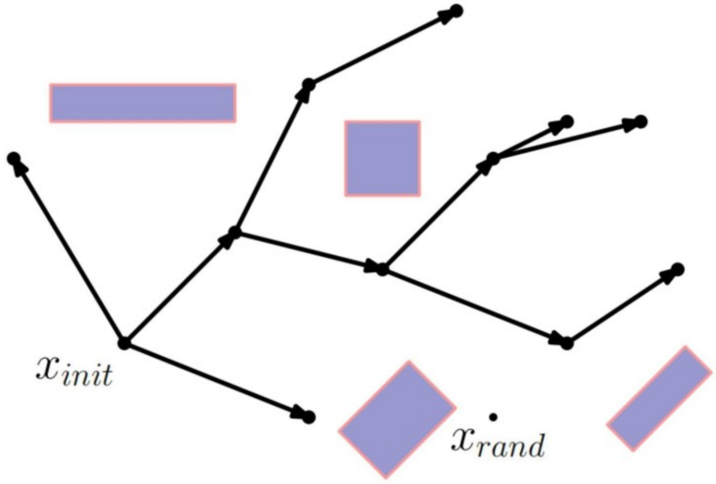
Lidar: Rapidly exploring Random Tree (RRT)



Continuous space: RRT

- RRT is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space-filling tree.
- Can run in any manifold
- Widely used in autonomous robotic motion planning
- Many variants: A*-RRT, LQR-RRT, CL-RRT, etc..

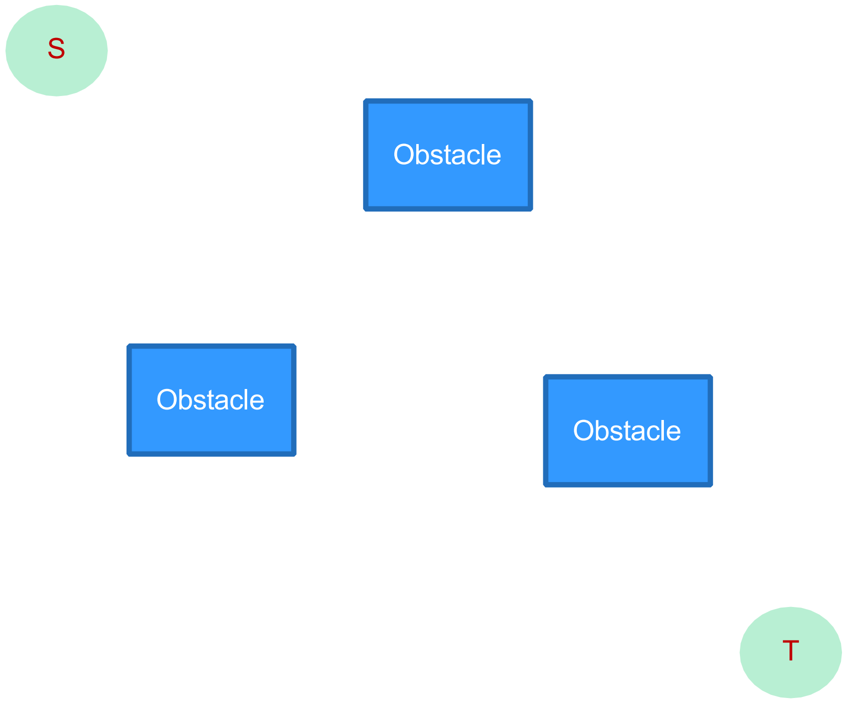
Lidar: Rapidly exploring Random Tree (RRT)



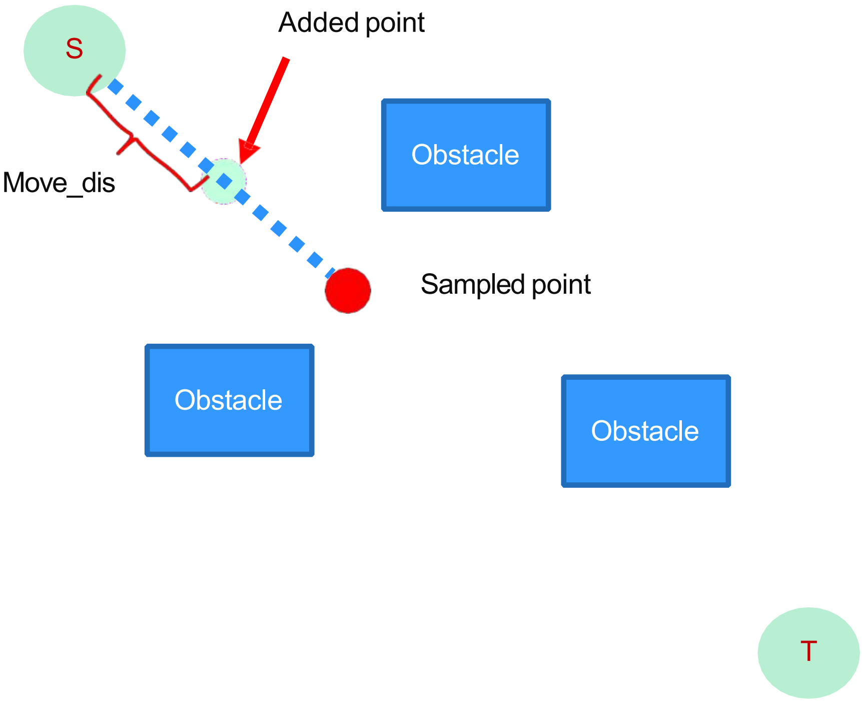
Run RRTBase.py and play with RRT algorithm

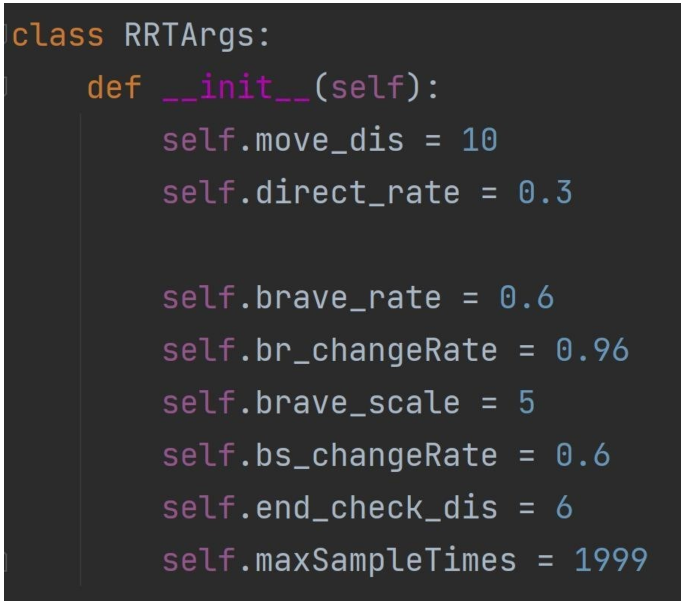
```
Algorithm BuildRRT
    Input: Initial config $q_{int}$
           Num of vertices $K$
           Incremental dist $\Delta q$
    Output: RRT graph $G$

    $G$.init($q_{int}$)
    for $k$ = $1$ to $K$ do
        $q_{rand}$ ← RAND_CONF()
        $q_{near}$ ← NEAREST_VERTEX($q_{rand}$, $G$)
        $q_{new}$ ← NEW_CONF($q_{near}$, $q_{rand}$, $\Delta q$)
        $G$.add_vertex($q_{new}$)
        $G$.add_edge($q_{near}$, $q_{new}$)
    return $G$
```

Lidar: RRT step by step



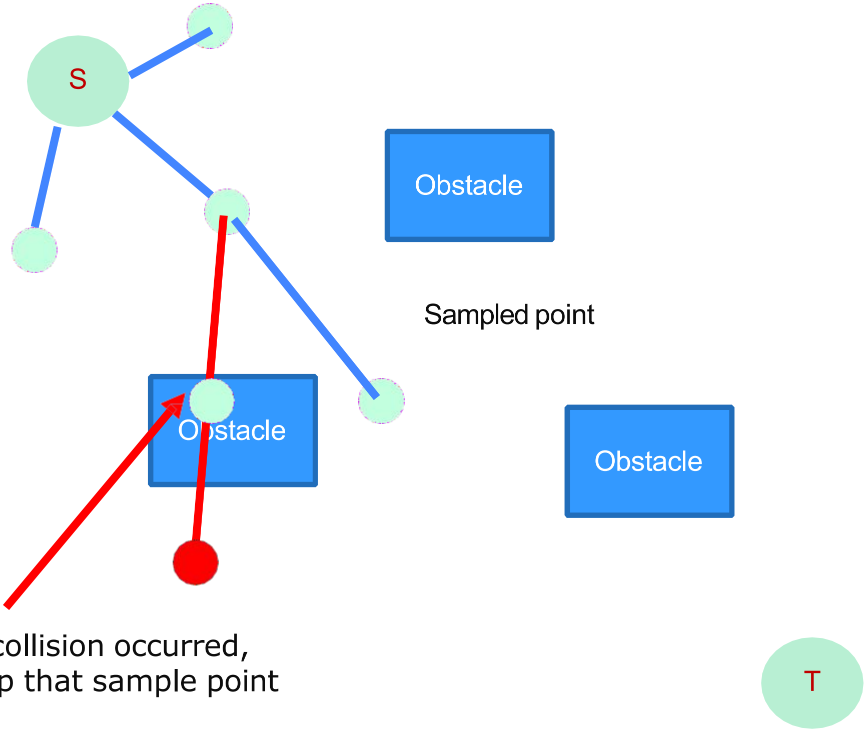
Lidar: RRT step by step



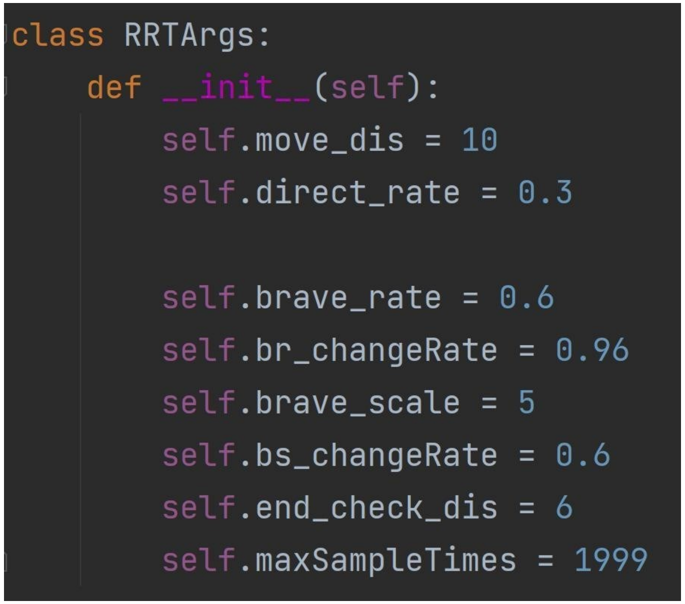


```
class RRTArgs:
    def __init__(self):
        self.move_dis = 10
        self.direct_rate = 0.3

        self.brave_rate = 0.6
        self.br_changeRate = 0.96
        self.brave_scale = 5
        self.bs_changeRate = 0.6
        self.end_check_dis = 6
        self.maxSampleTimes = 1999
```

Lidar: RRT step by step

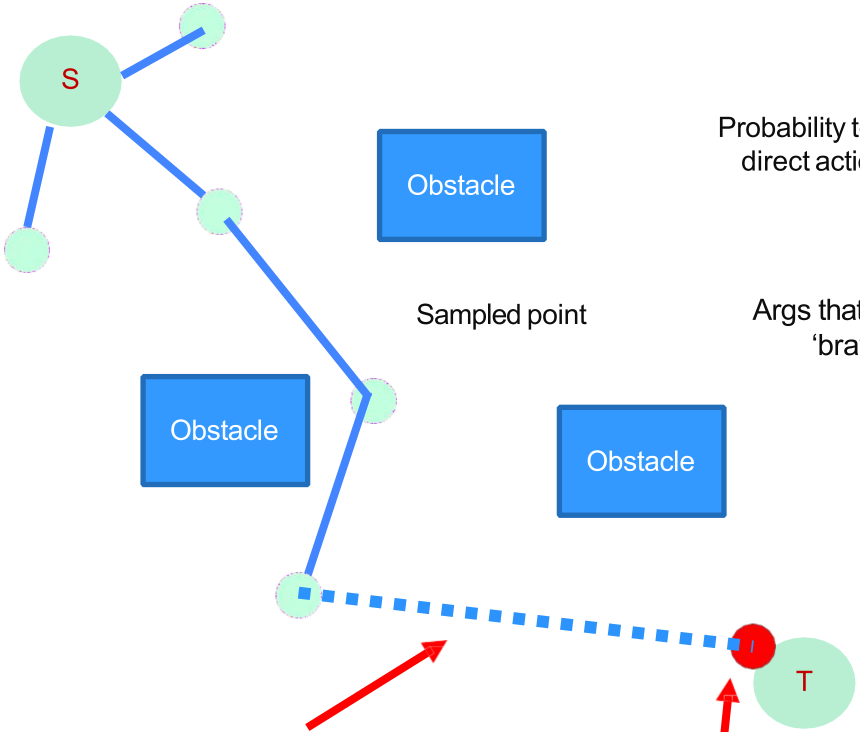


If a collision occurred,
Give up that sample point



```
class RRTArgs:
    def __init__(self):
        self.move_dis = 10
        self.direct_rate = 0.3

        self.brave_rate = 0.6
        self.br_changeRate = 0.96
        self.brave_scale = 5
        self.bs_changeRate = 0.6
        self.end_check_dis = 6
        self.maxSampleTimes = 1999
```

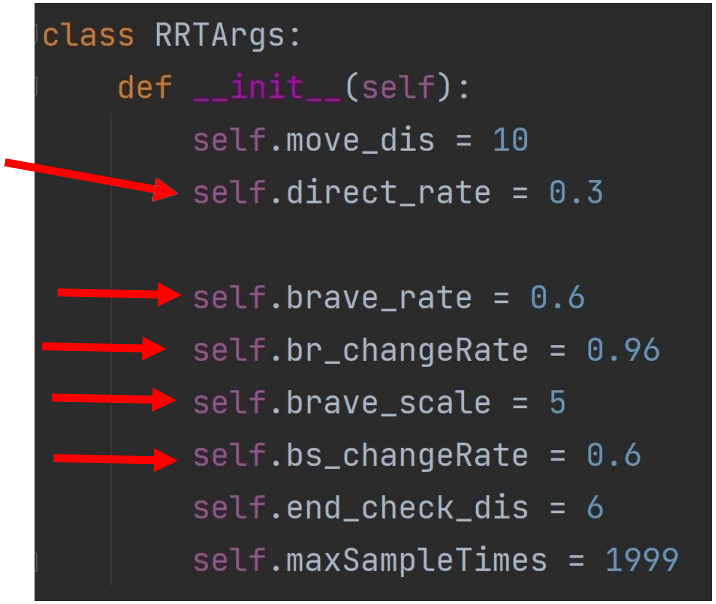
Lidar: RRT step by step



Sampled point

Probability to do direct action

Args that control 'brave'

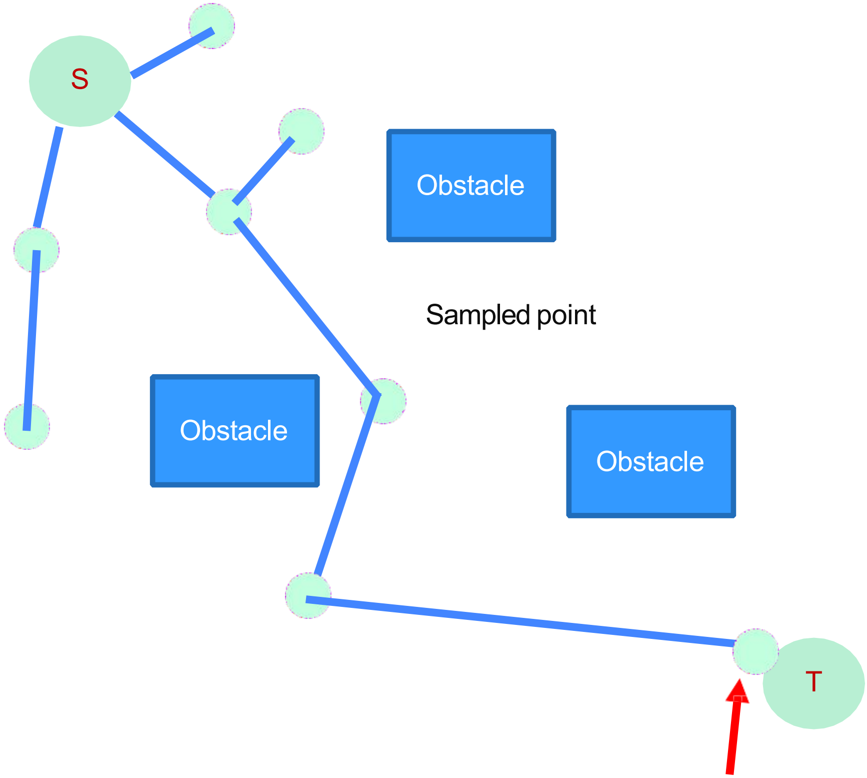


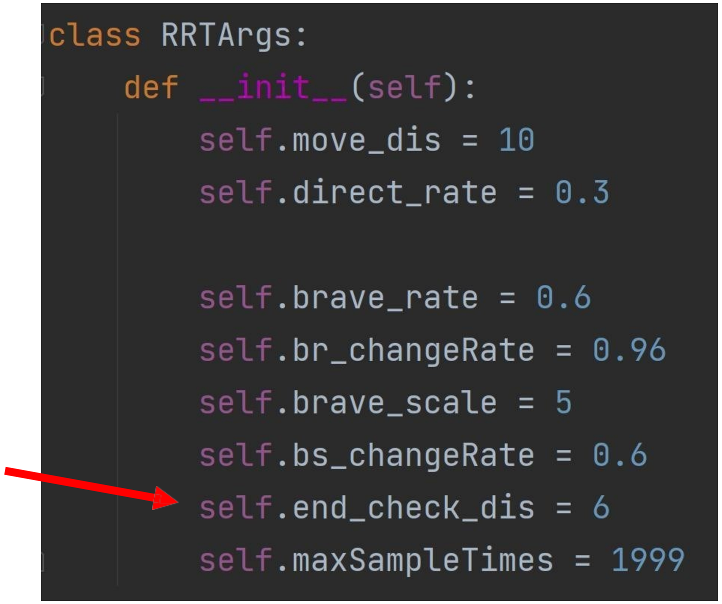
```
class RRTArgs:
    def __init__(self):
        self.move_dis = 10
        self.direct_rate = 0.3

        self.brave_rate = 0.6
        self.br_changeRate = 0.96
        self.brave_scale = 5
        self.bs_changeRate = 0.6
        self.end_check_dis = 6
        self.maxSampleTimes = 1999
```

If you walk longer than move_dis,
we call it 'brave scale'
Move = bs * move_dis

When you directly sample the target, we call it a 'direct action'

Lidar: RRT step by step

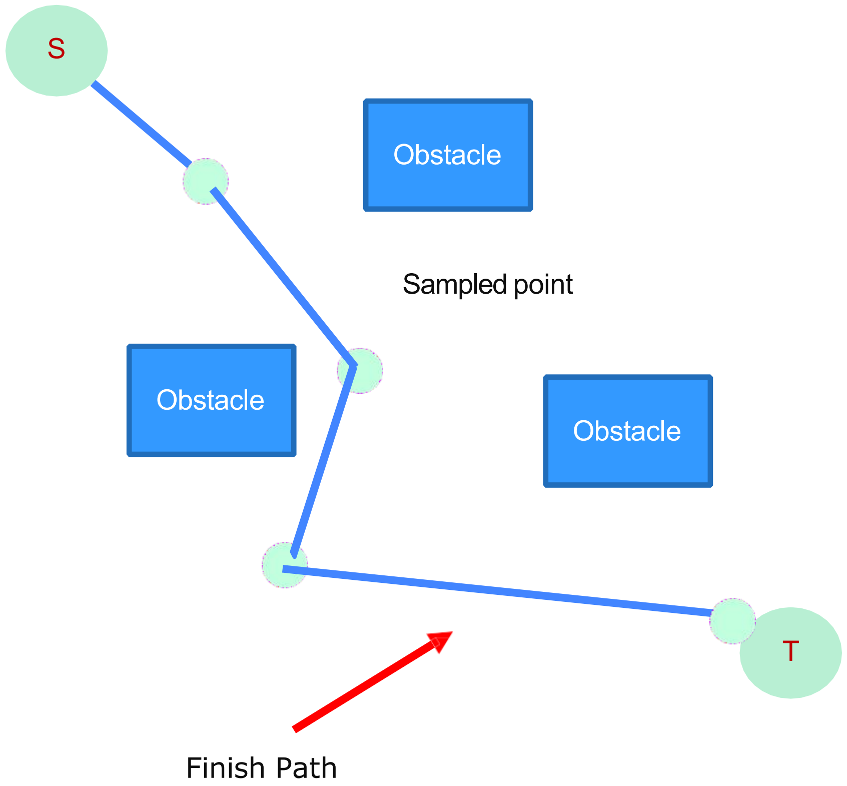


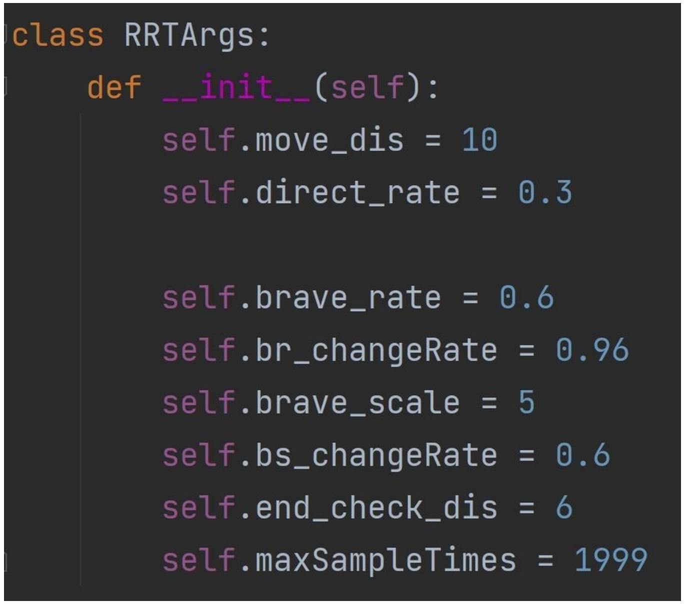


```
class RRTArgs:
    def __init__(self):
        self.move_dis = 10
        self.direct_rate = 0.3

        self.brave_rate = 0.6
        self.br_changeRate = 0.96
        self.brave_scale = 5
        self.bs_changeRate = 0.6
        self.end_check_dis = 6
        self.maxSampleTimes = 1999
```

If distance of (P, T) < end_check_dis, we admit you achieve the goal

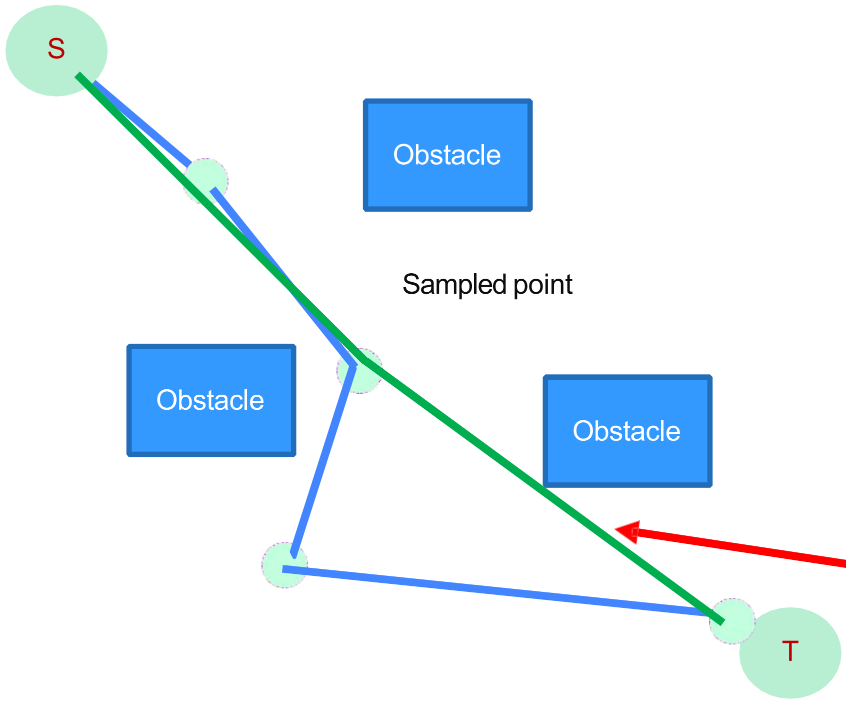
Lidar: RRT step by step

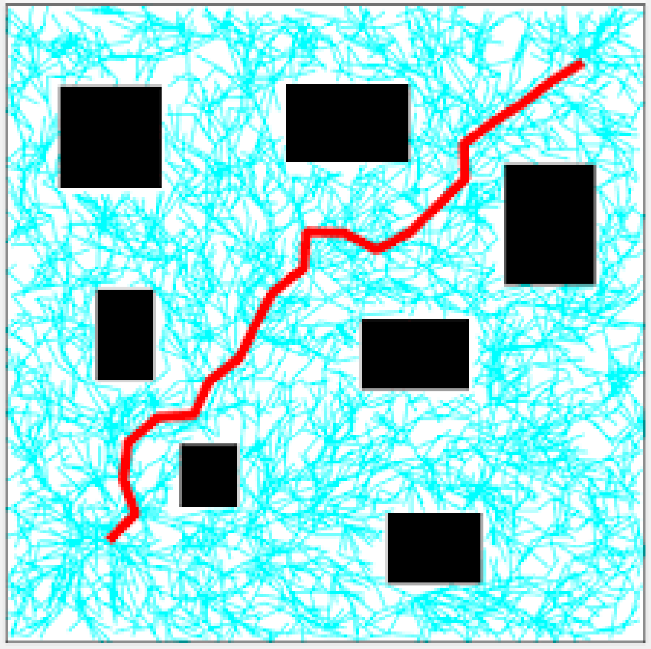




```
class RRTArgs:
    def __init__(self):
        self.move_dis = 10
        self.direct_rate = 0.3

        self.brave_rate = 0.6
        self.br_changeRate = 0.96
        self.brave_scale = 5
        self.bs_changeRate = 0.6
        self.end_check_dis = 6
        self.maxSampleTimes = 1999
```

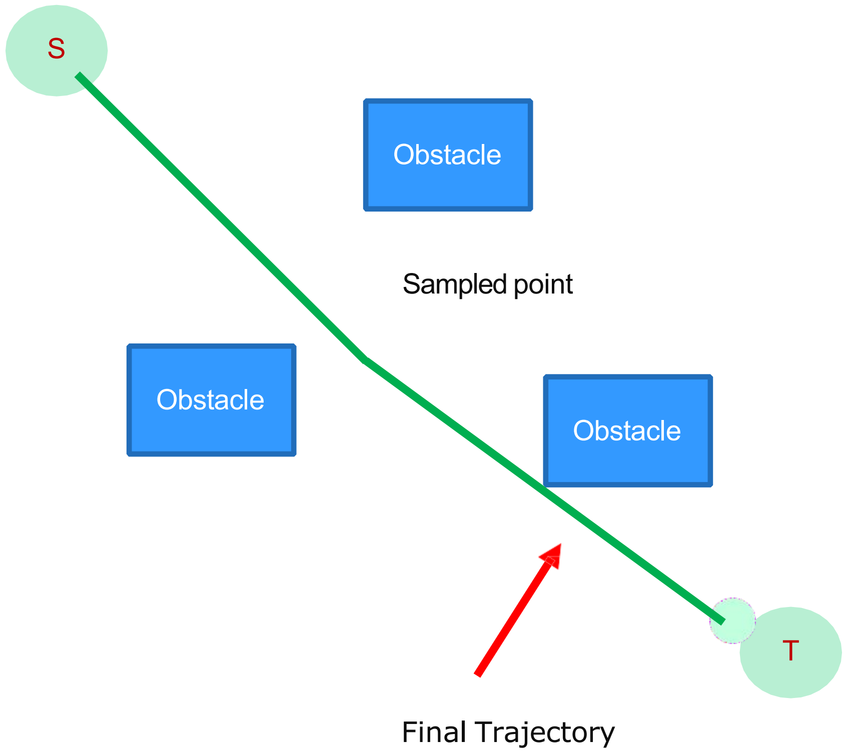
Lidar: RRT step by step

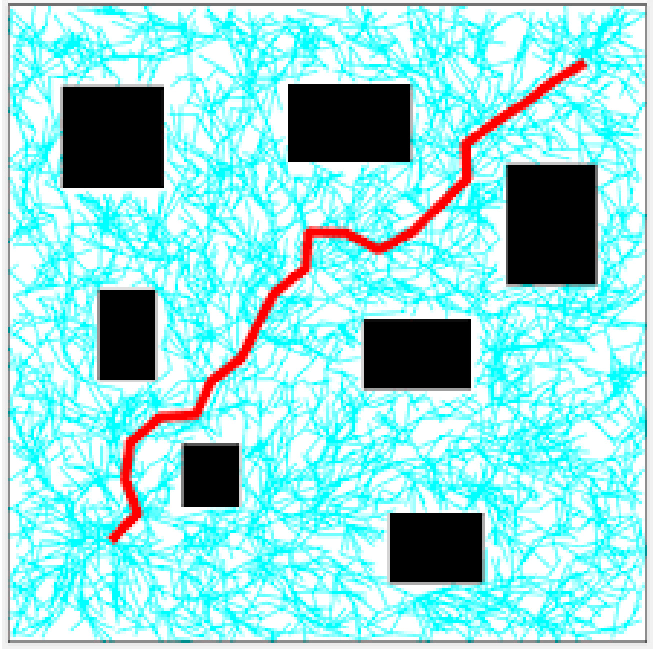




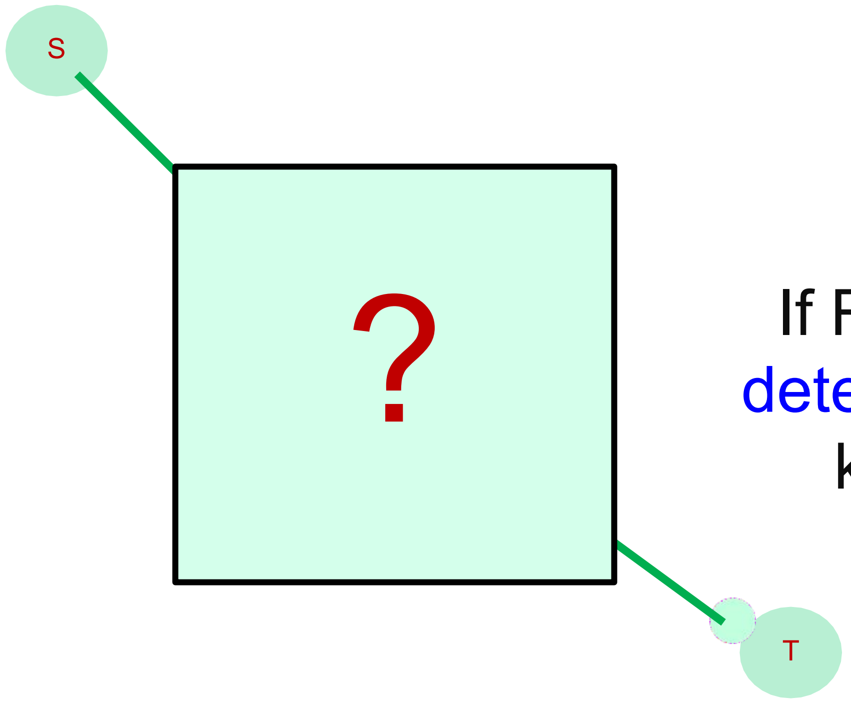
Simplify the trajectory

Lidar: RRT step by step



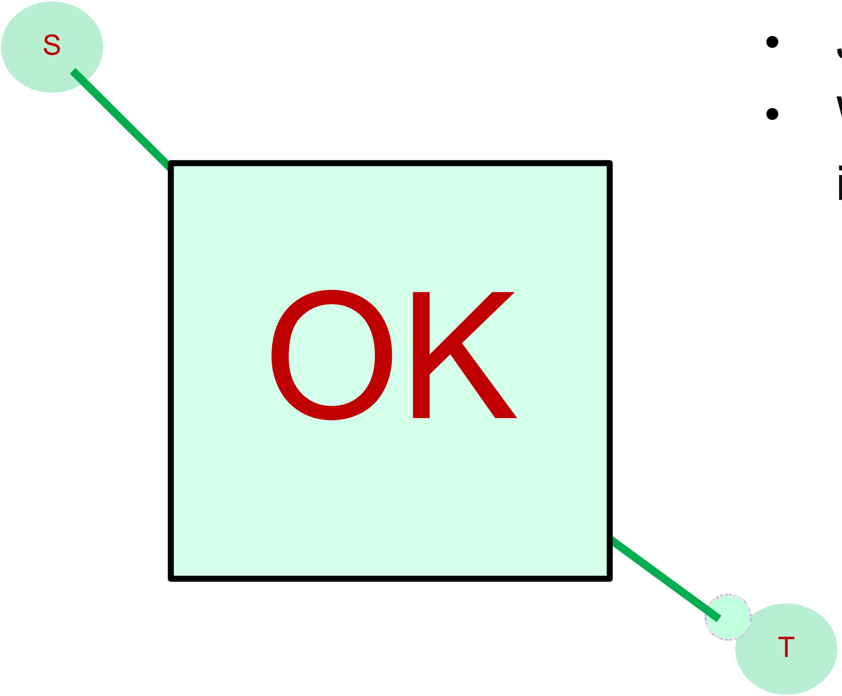


Lidar: RRT step by step

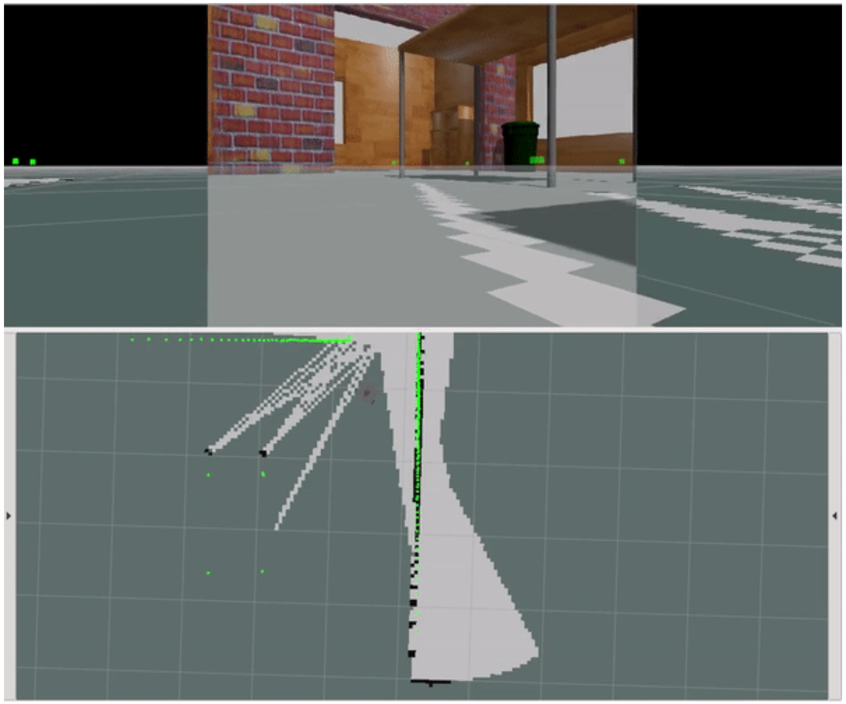


If RRT is based on collision detection, what if we DO NOT know the environment?

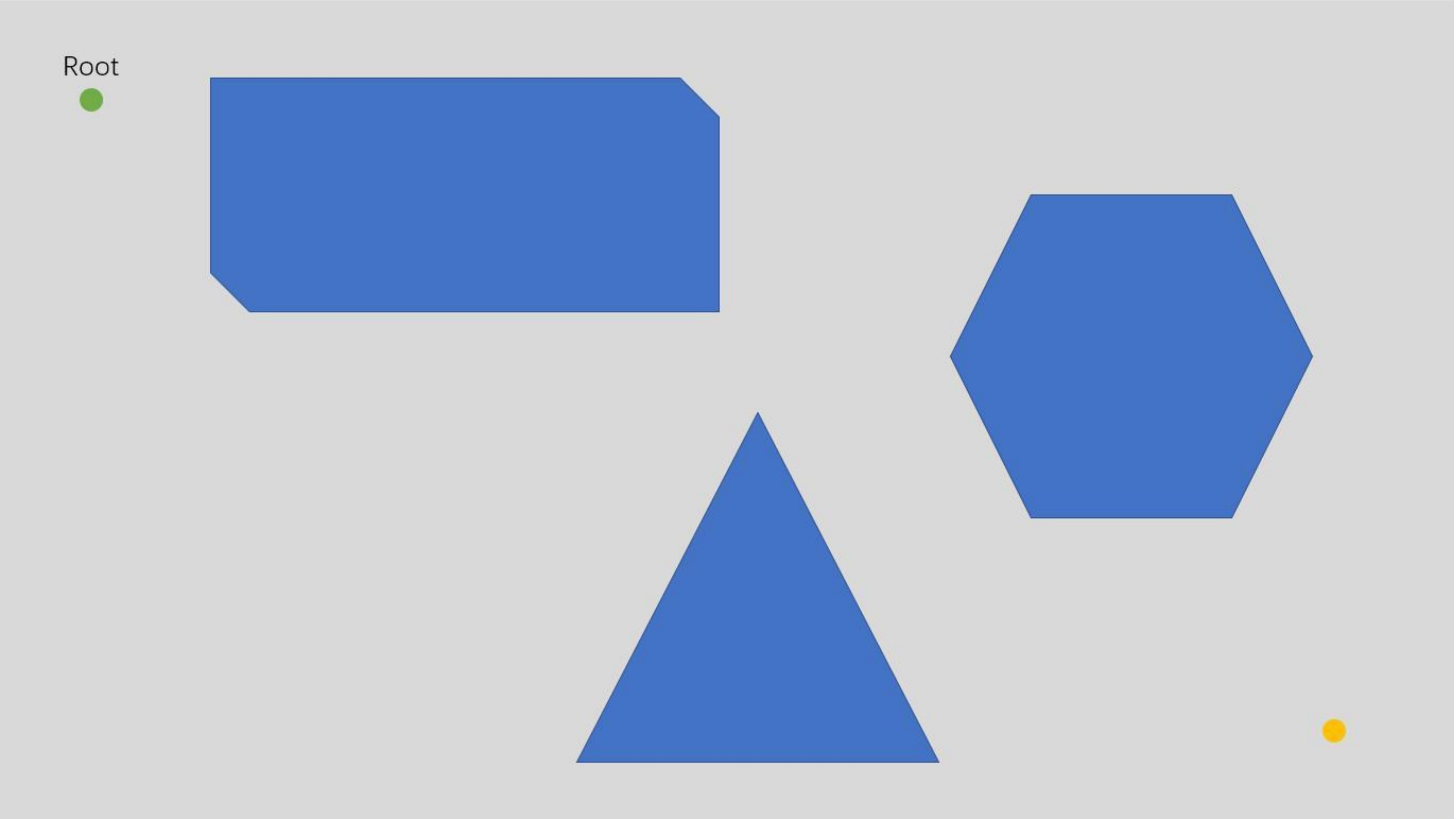
Lidar: RRT step by step



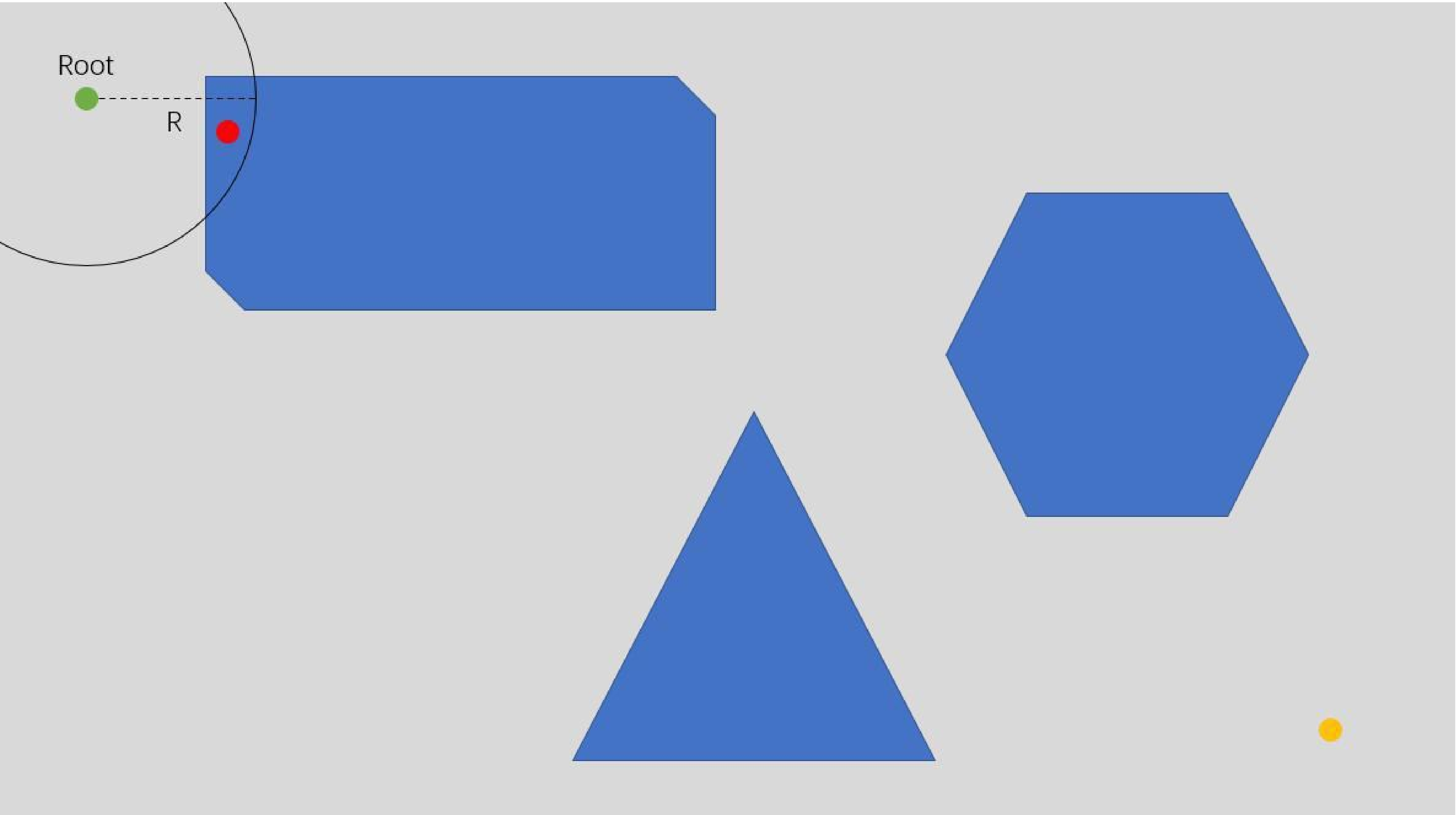
- The answer is, we do not care about it.
- Just plan based on what you have.
- When you move, you will get more information, and do re-planning again.



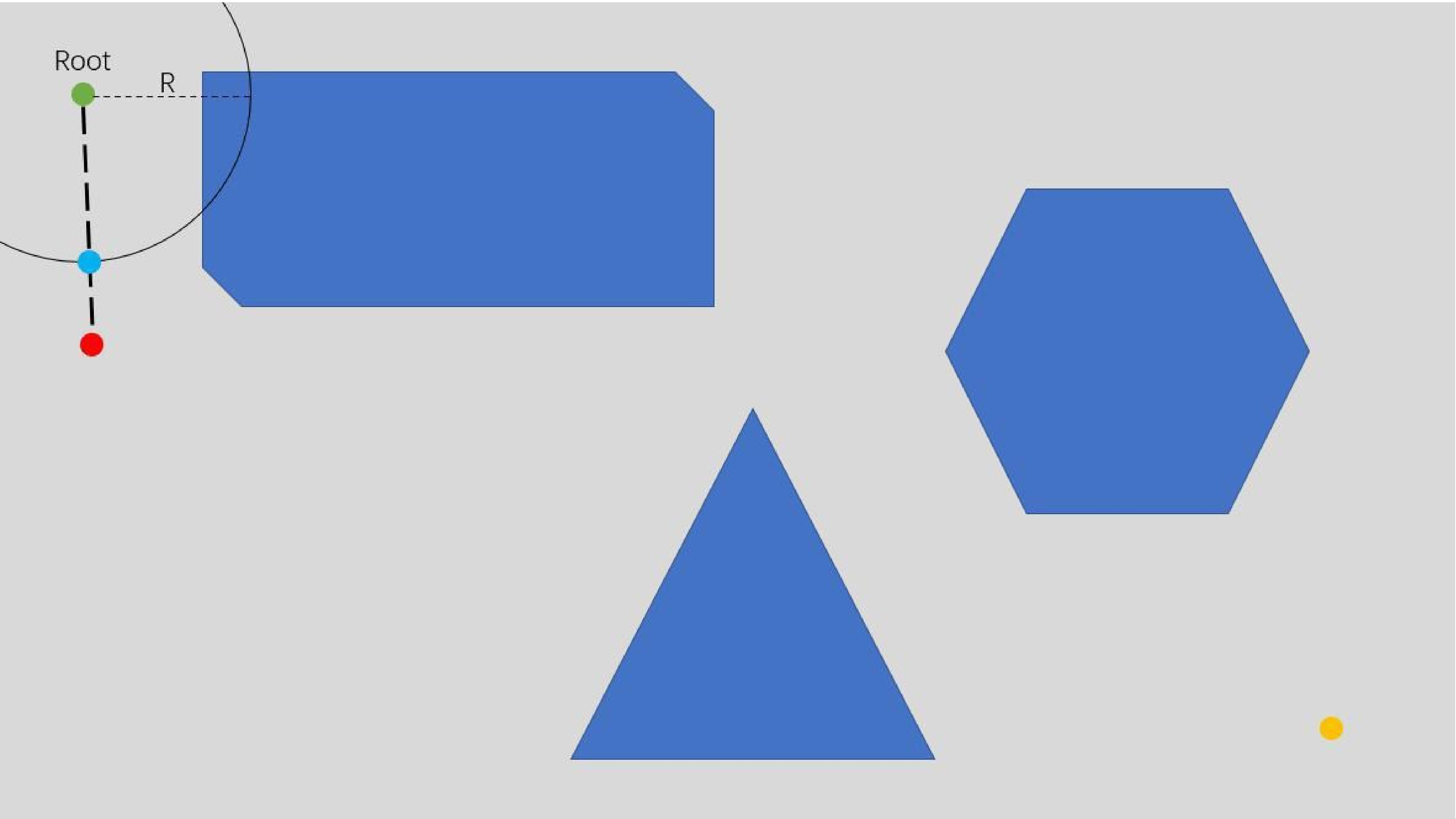
RRT Summary



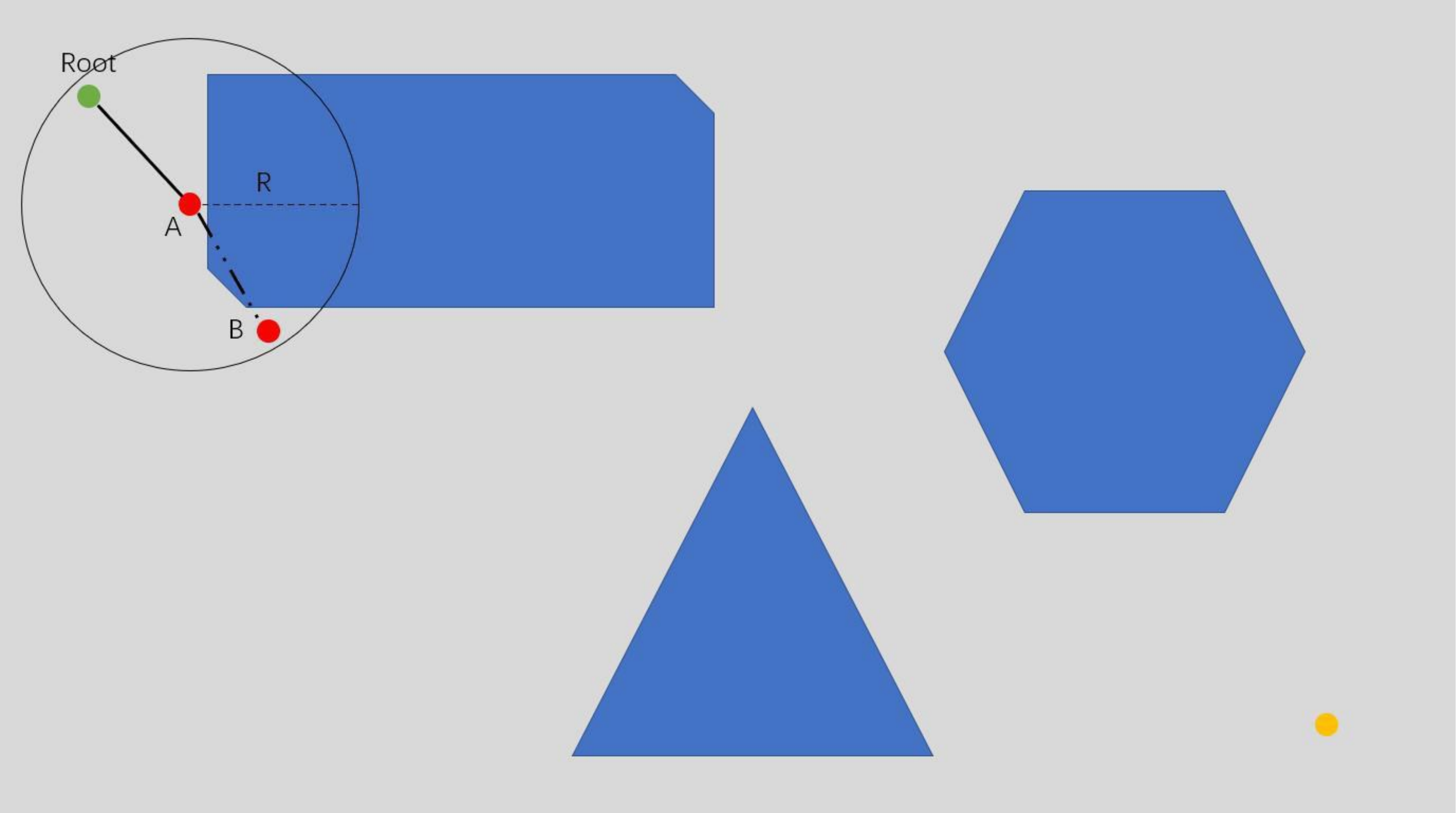
RRT Summary



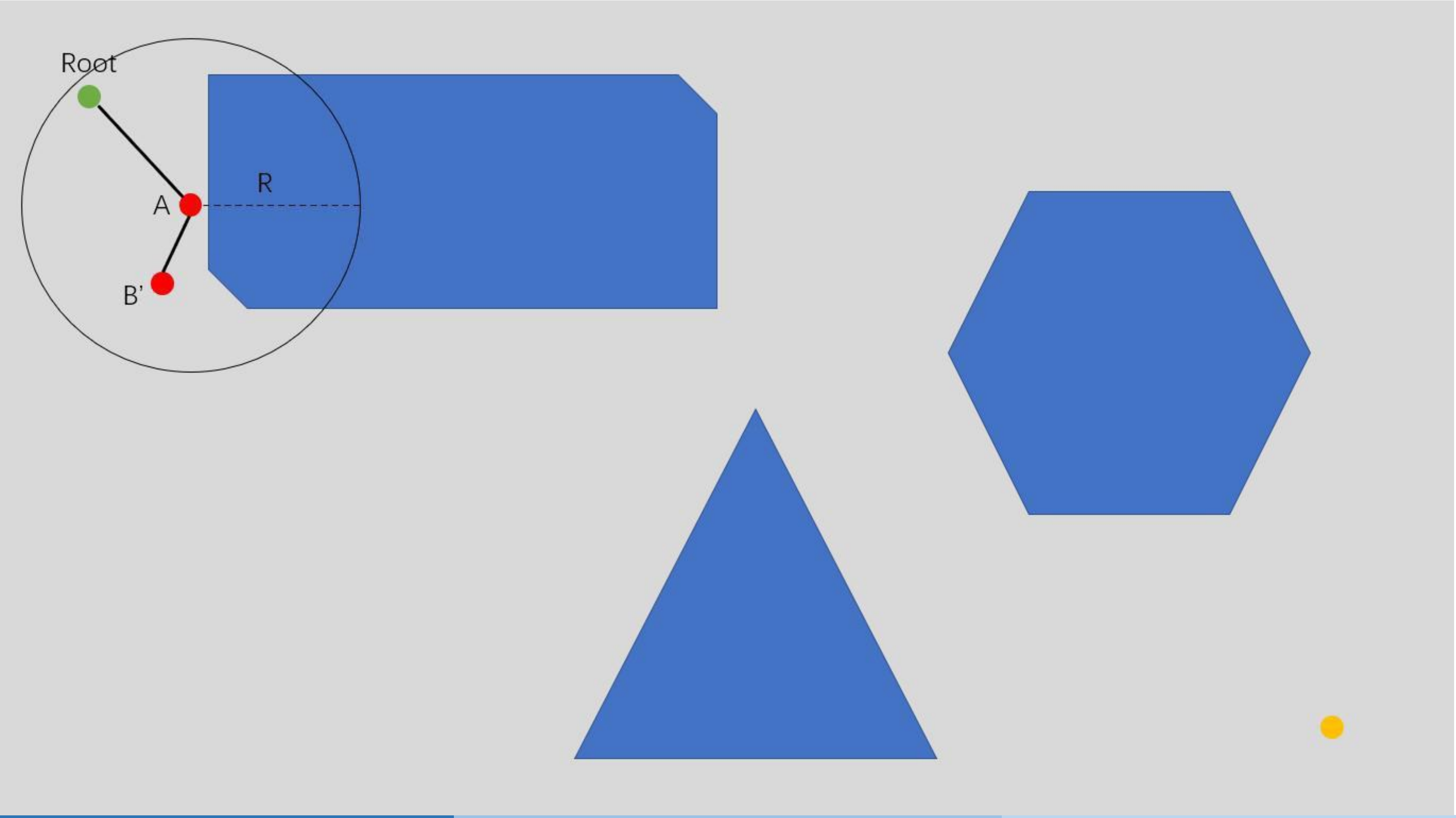
RRT Summary



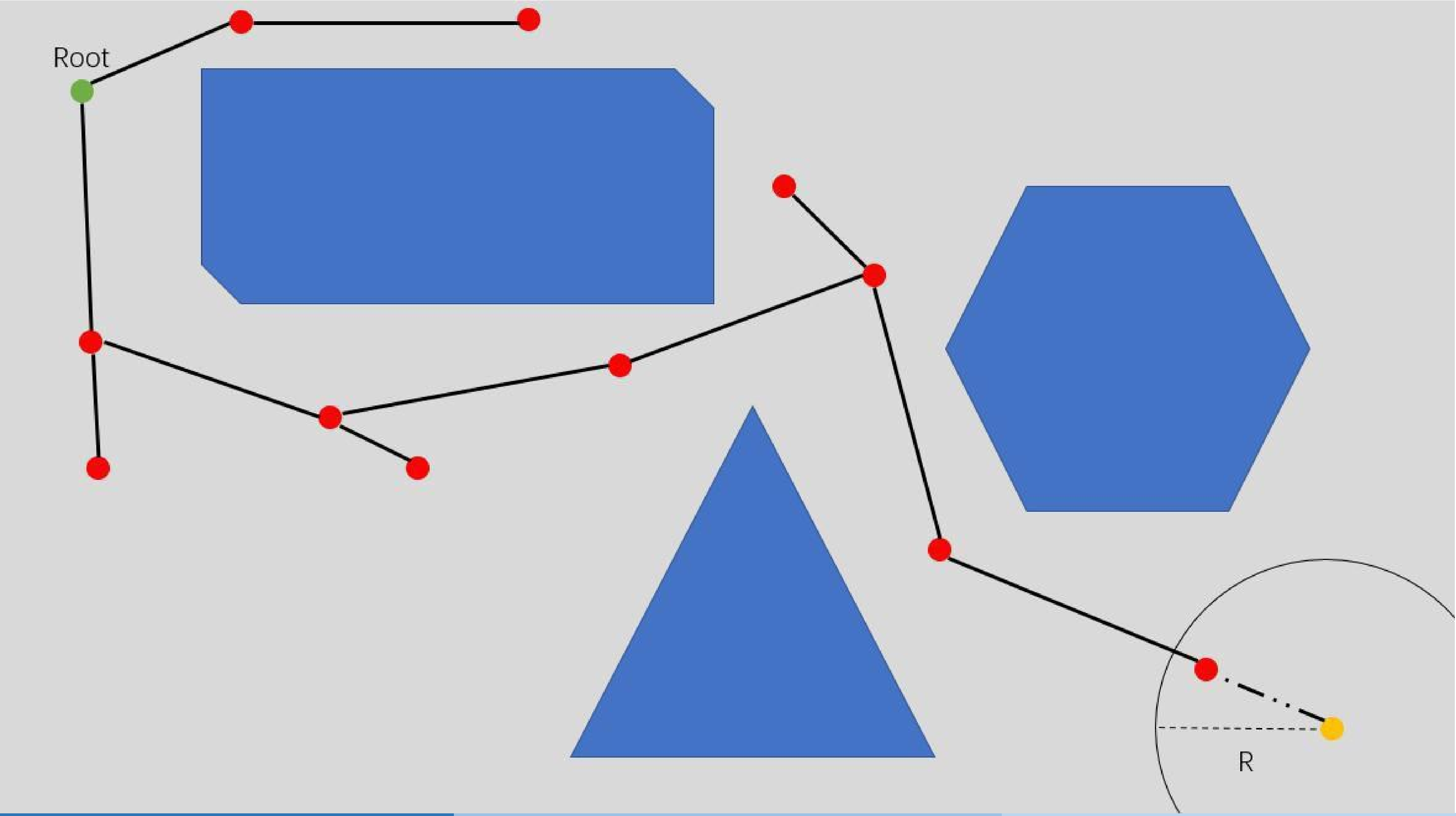
RRT Summary



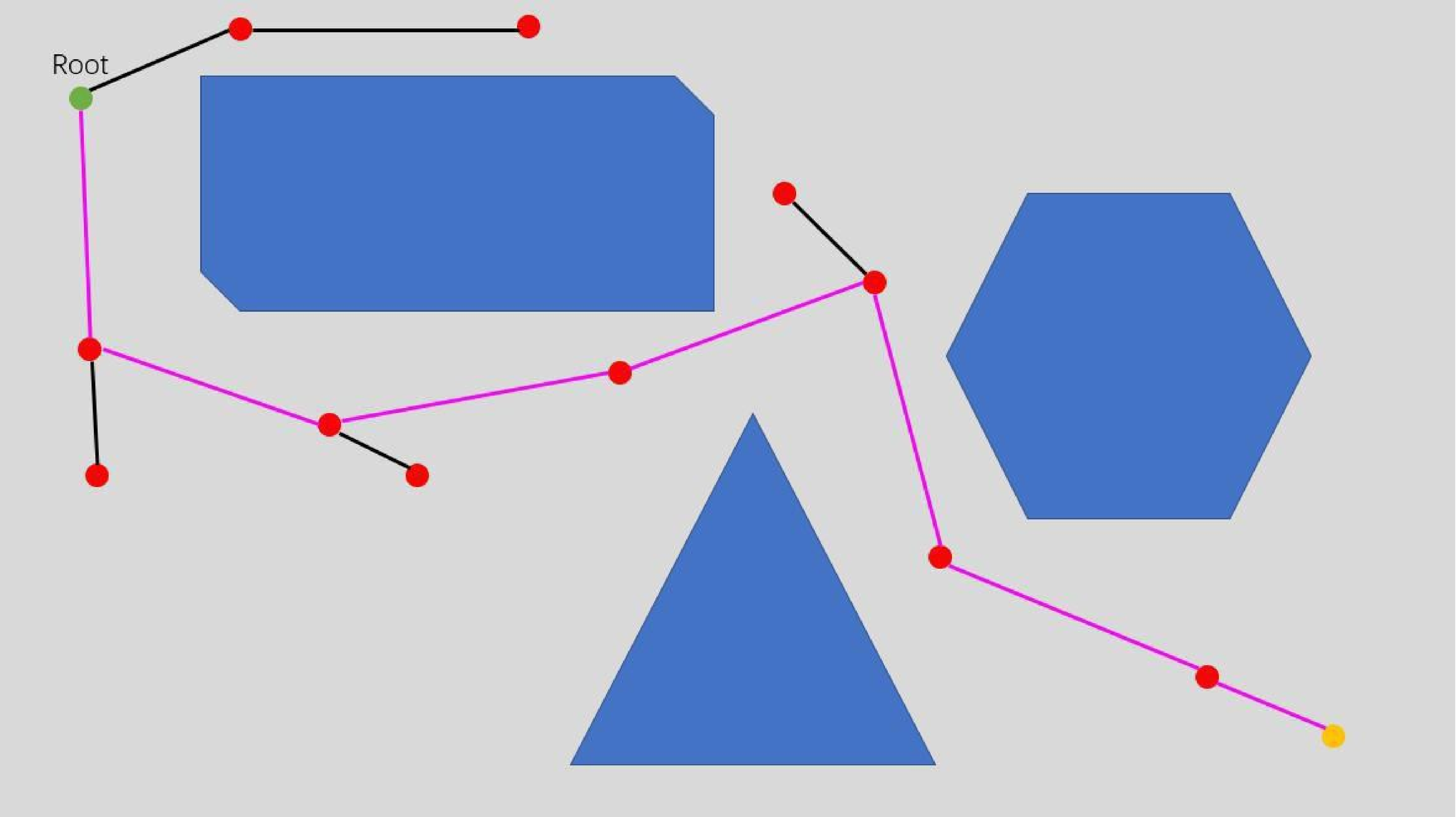
RRT Summary



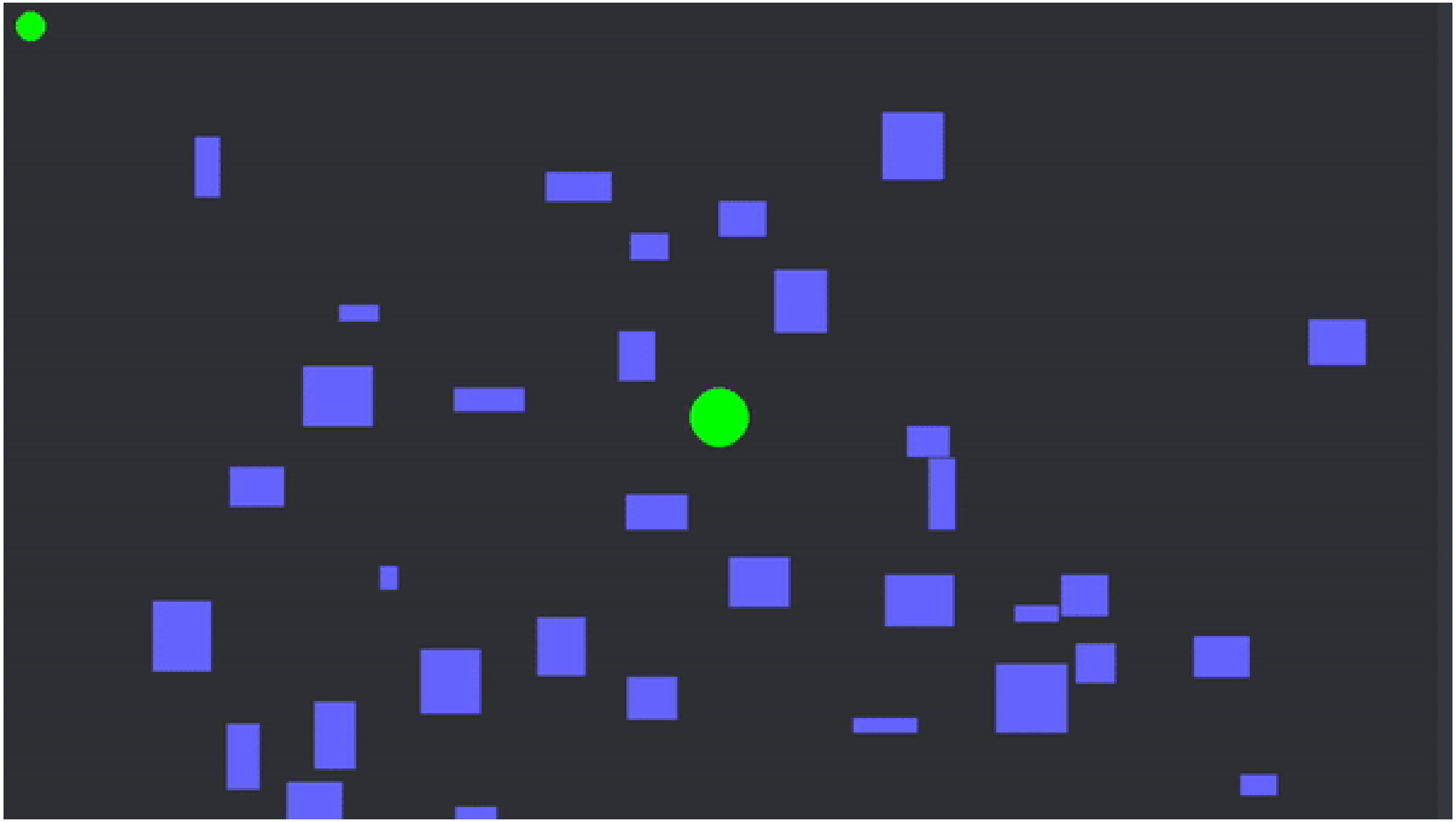
RRT Summary



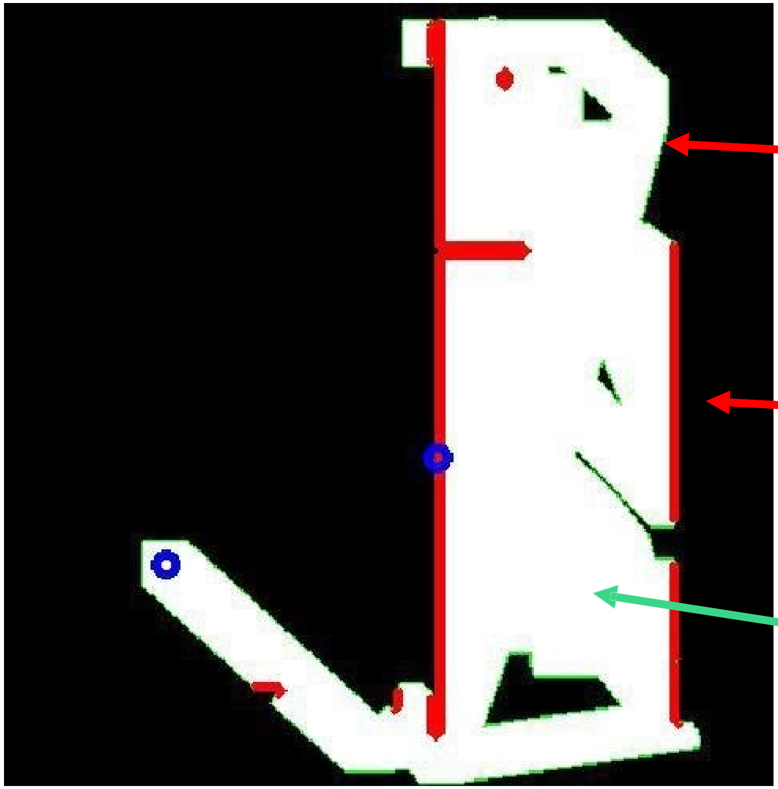
RRT Summary



RRT Summary



Lidar Policy

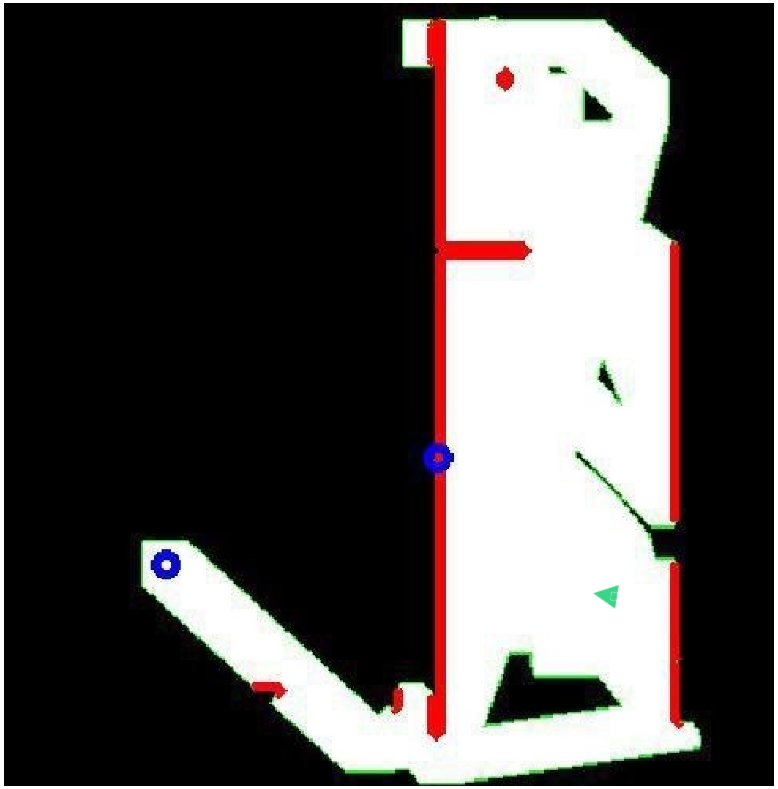


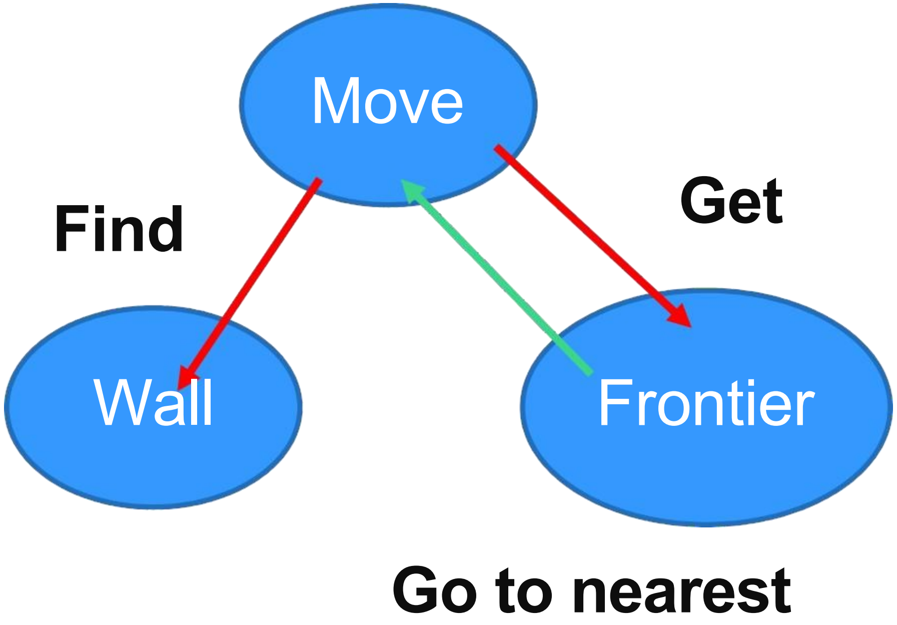
Frontier: the edge of detected space (except walls)

Wall detected

Space we scanned

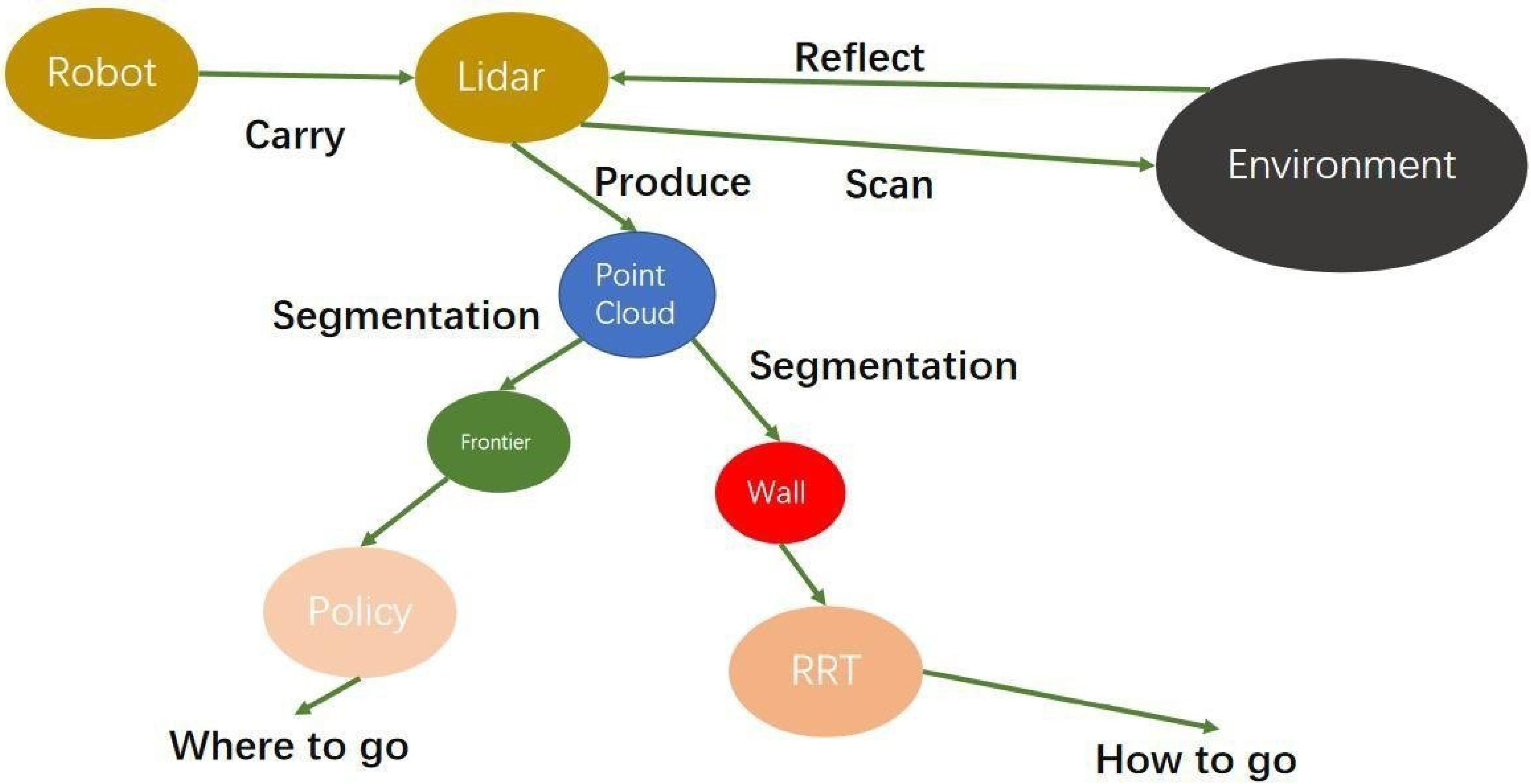
Lidar Policy

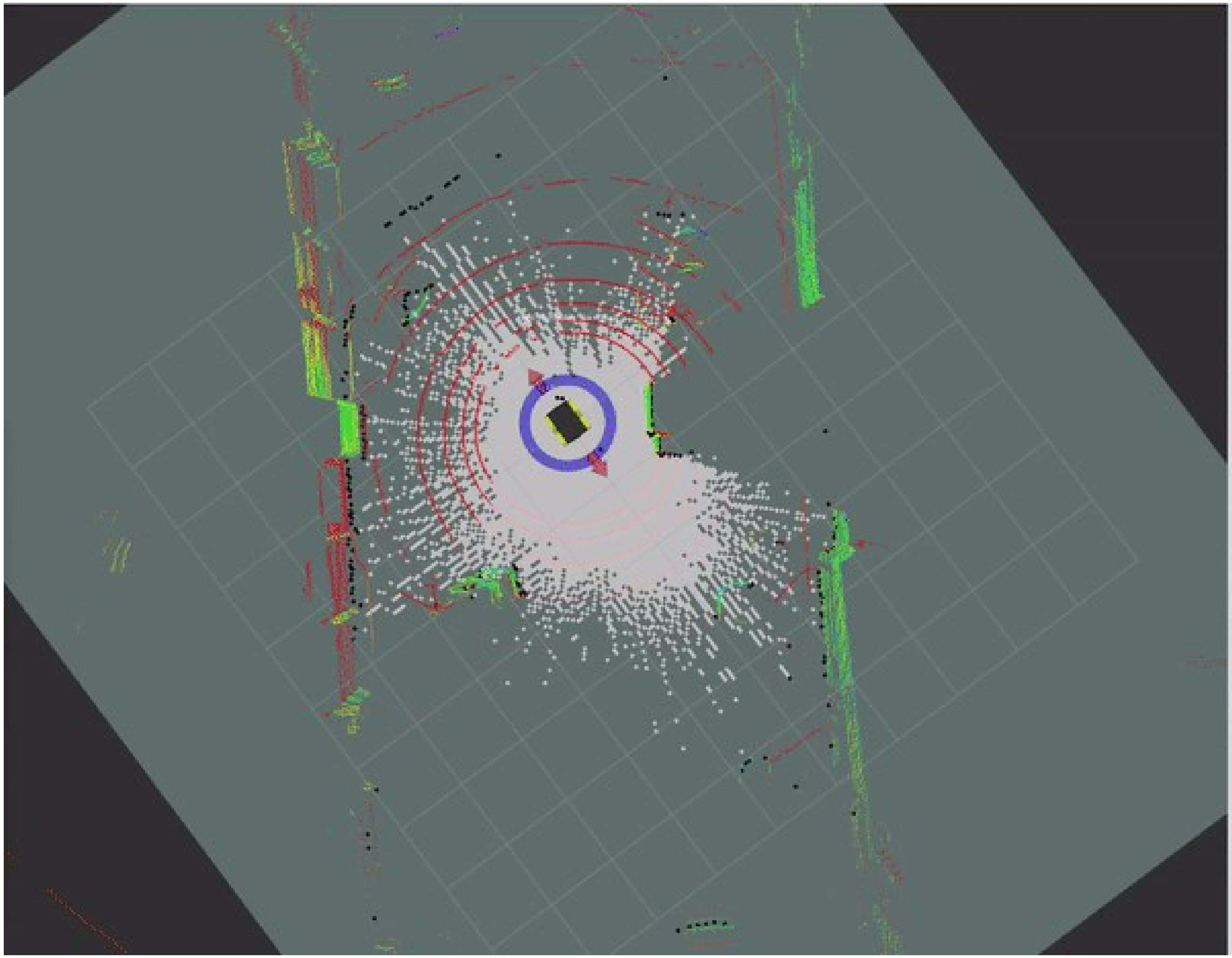




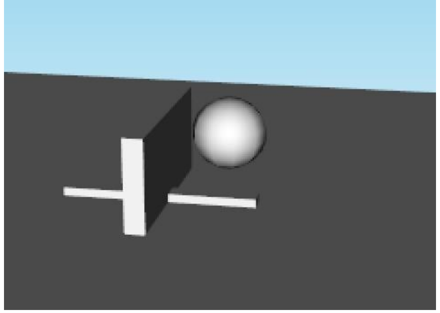
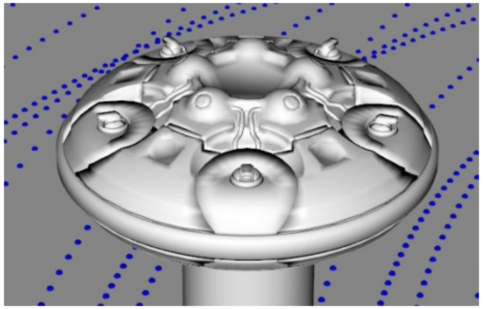
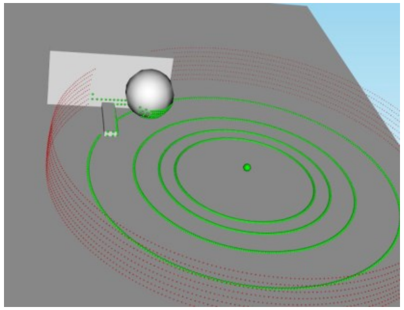
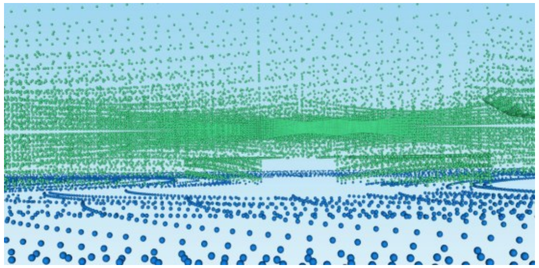
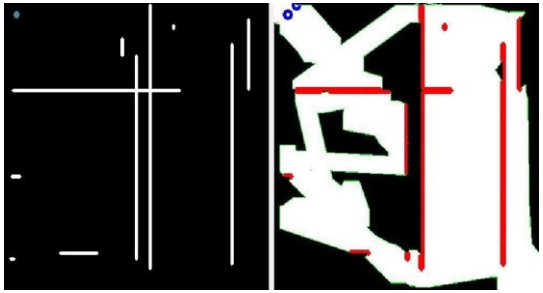
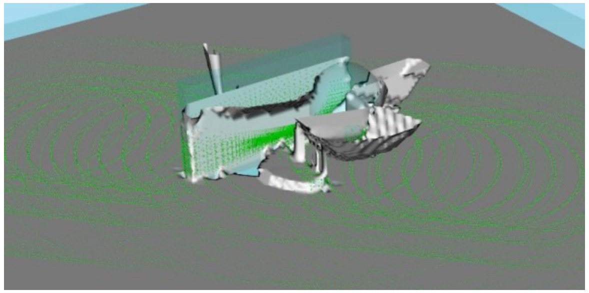
A simple planning policy for mobile robots in building a map

Lidar: Conclusion





Assignment

- Design your own lidar / Scan point clouds / simple segmentation / reconstruction / testing different RRTs
- All with examples
- **ROS-free**
- Better bring your own PC; Use IDE like VS Code / PyCharm