ROBOT INTELLIGENCE LAB

Good Old Fashioned Path-Planning

Subtitle

Sungjoon Choi, Korea University

kakaobrain

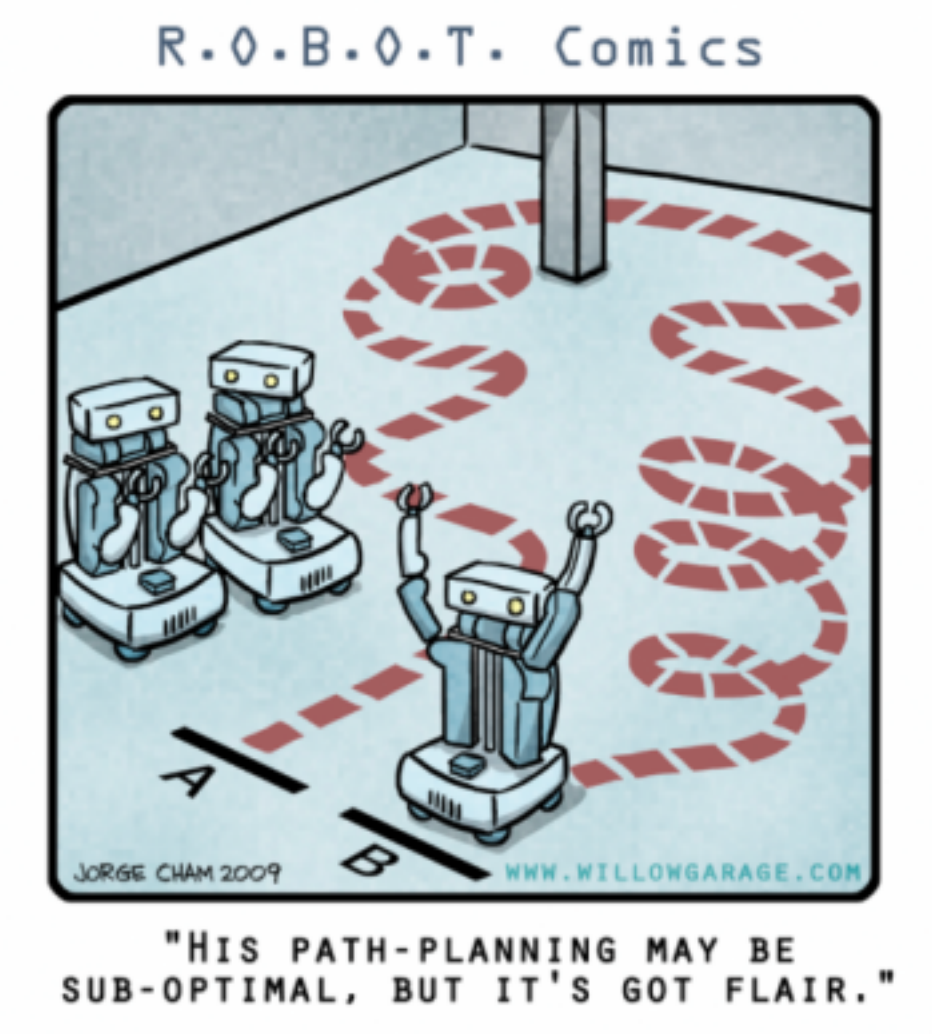
Introduction

ROBOT INTELLIGENCE LAB

We will be learning about good-old fashioned **path planning** methods in a chronological manner.

Path Planning

ROBOT INTELLIGENCE LAB



http://www.willowgarage.com/sites/default/files/motionplanning.png

Contents

ROBOT INTELLIGENCE LAB

- A* Algorithm (1968)
- Visibility Graph (1969)
- Potential Field (1987)
- Vector Field Histogram (1990)
- Pure Pursuit (1992)
- Probabilistic Roadmap (1996)
- Dynamic Window Approach (1996)
- Velocity Obstacles (1998)
- Rapidly-Exploring Random Trees (2001)
- RRT* (2010)
- My Work (GP Navigation, Gaussian Random Path and Occupancy Flow)

ROBOT INTELLIGENCE LAB

A* Algorithm

"A Formal Basis for the Heuristic Determination of Minimum Cost Paths,"1968

A* Algorithm

ROBOT INTELLIGENCE LAB

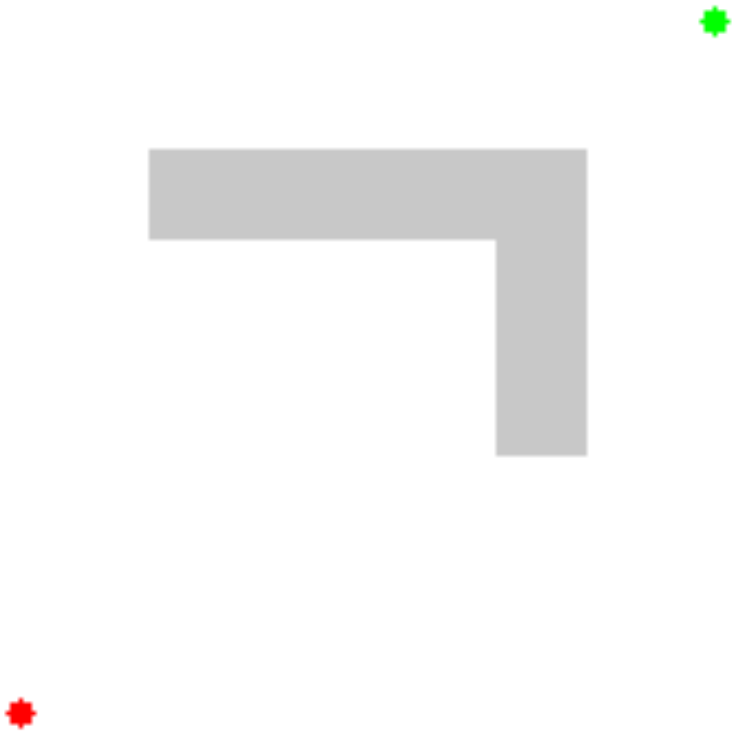
- A* algorithm is a searching algorithm for **finding the shortest path** between the initial and final state.

- It requires two cost functions:
  - $g(s)$: the cost of moving from the initial cell to the current cell.
  - $h(s)$: the estimated cost of moving from the current cell to the final cell (also known as the **heuristic function**). The estimated cost should be **smaller** than the actual cost (optimistic estimation).

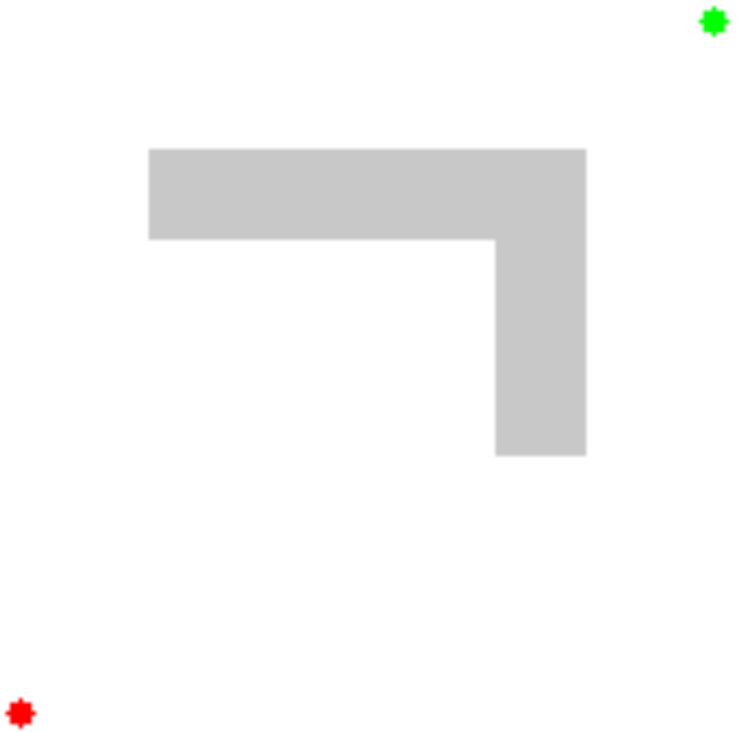
- What happens if the consistency (optimistic estimation) does not hold?

A* Algorithm

ROBOT INTELLIGENCE LAB



Inconsistent heuristics



Consistent heuristics

ROBOT INTELLIGENCE LAB

Visibility Graph

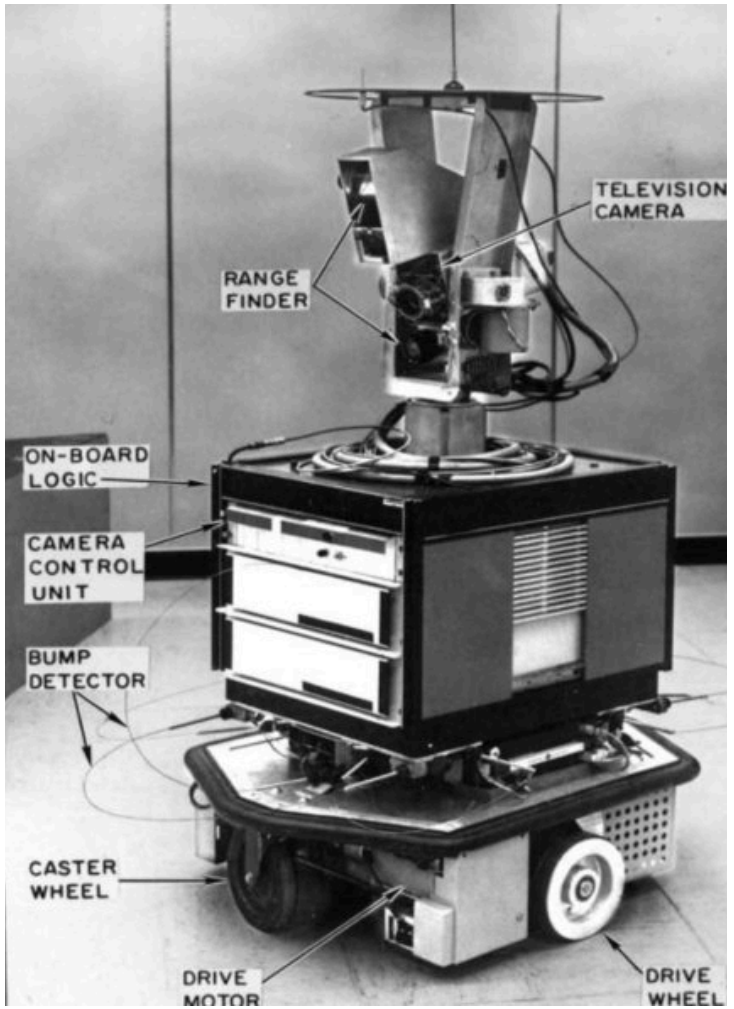
"An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," 1969

Visibility Graph

ROBOT INTELLIGENCE LAB

- Visibility graph is a collision avoidance algorithm for planning a safe path among polyhedral objects for **Shakey the robot**.

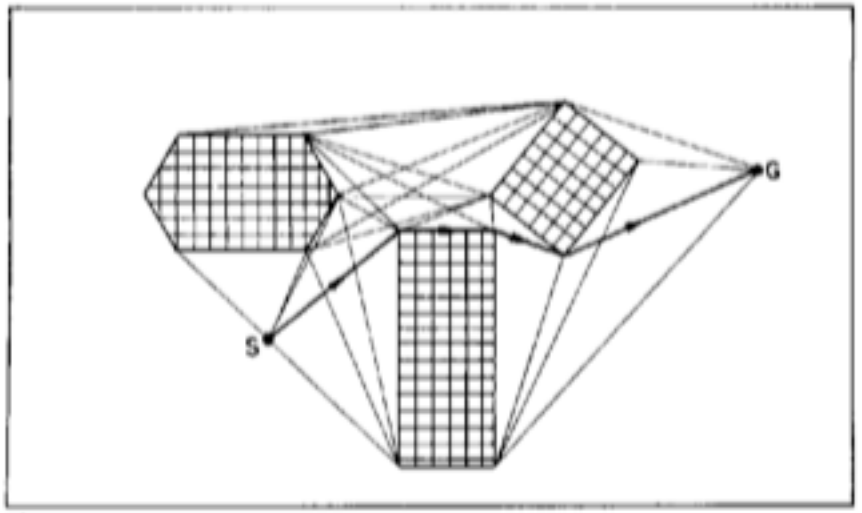
- According to wikipedia, **Shakey the robot** (1966-1972) was the first general-purpose mobile robot to be able to reason about its own actions.
- It was developed at the Artificial Intelligence Center of Stanford Research Institute.
- Most notable results of this project include the A* search algorithm, the Hough transform, and the visibility graph method.



Shakey the robot

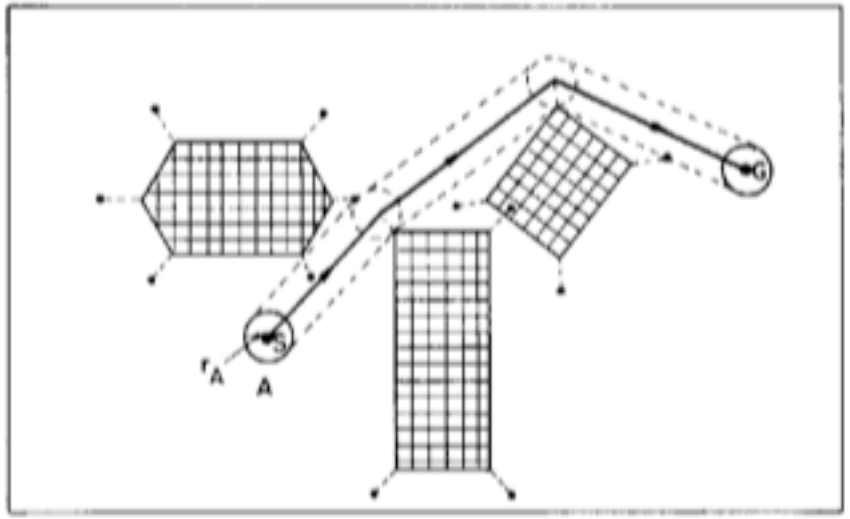
Visibility Graph

ROBOT INTELLIGENCE LAB



- The undirected visibility graph, $VG(N, L)$, is defined:
  - The **node set** N is $V \cup (S, G)$ where V is the set of all vertices of obstacles and the **link set** L is the set of all links such that a straight like connecting two elements of N does not overlap any obstacle.
  - The the shortest collision-free path can be found by **graph search**.

Visibility Graph



- The dimensions of the robot is considered by giving margins to the obstacles (growing the obstacles) and treat the robot as a point.
- For the robots with **non-circle shapes**, additional effects of rotation must be incorporated.
- The biggest problem is the **quantization** of configuration parameters.

ROBOT INTELLIGENCE LAB

Potential Field

"Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations," 1987

Potential Field

ROBOT INTELLIGENCE LAB

- Assuming a point robot, potential function assumes the existence of repulsive and attractive forces acting on a point robot derived from potential fields.

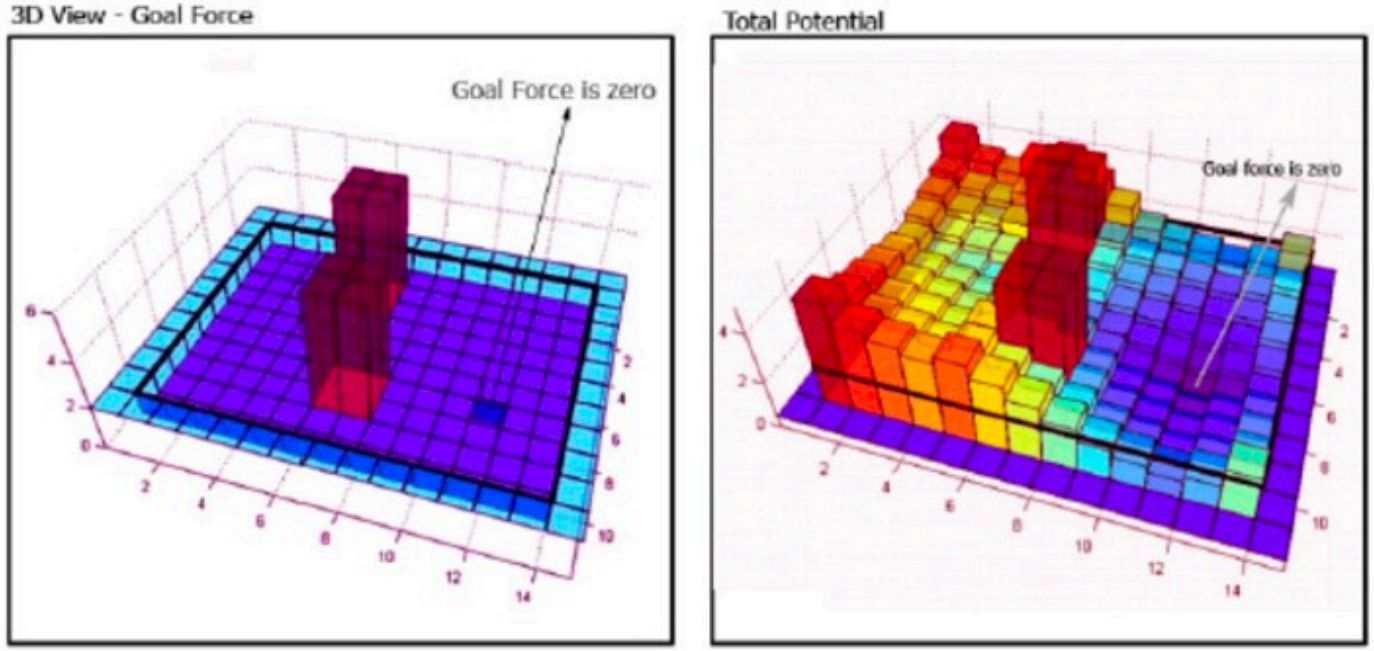


Figure 9. Path planning with algorithm of potential fields

Potential Field

ROBOT INTELLIGENCE LAB

- Albeit its simplicity, the main caveat is the possible existence of multiple **local minima** and the **balance** between attractive and repulsive forces.
- To handle this, a navigation function is often used.
  - The navigation function is the result of extensive research on creating an artificial potential field that will have only unique minima, so that it is guaranteed that the robot will reach to goal.

Local Minima Problem

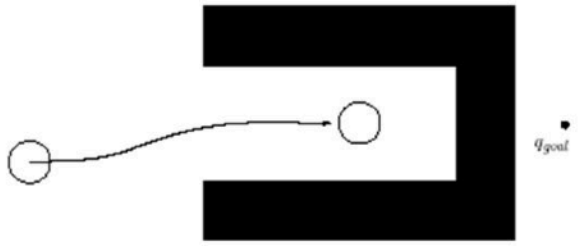


FIGURE 4.10. Local minimum inside the concavity.

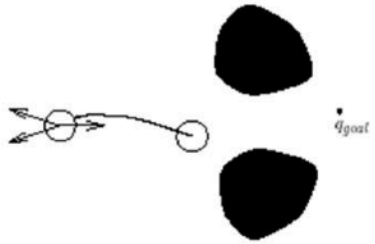


FIGURE 4.11. Local minimum without concave obstacles.

https://slideplayer.com/slide/5237975/

ROBOT INTELLIGENCE LAB

Vector Field Histogram

"Real-time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," 1990

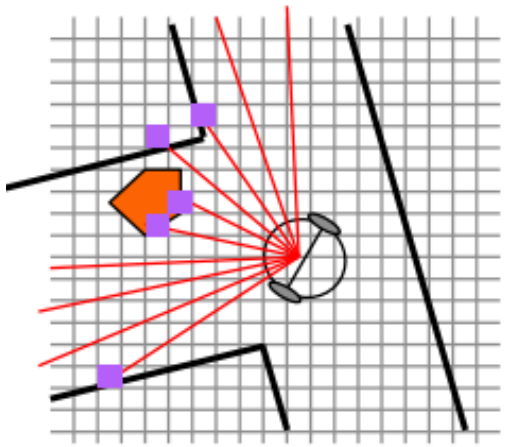
"VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots," 1997

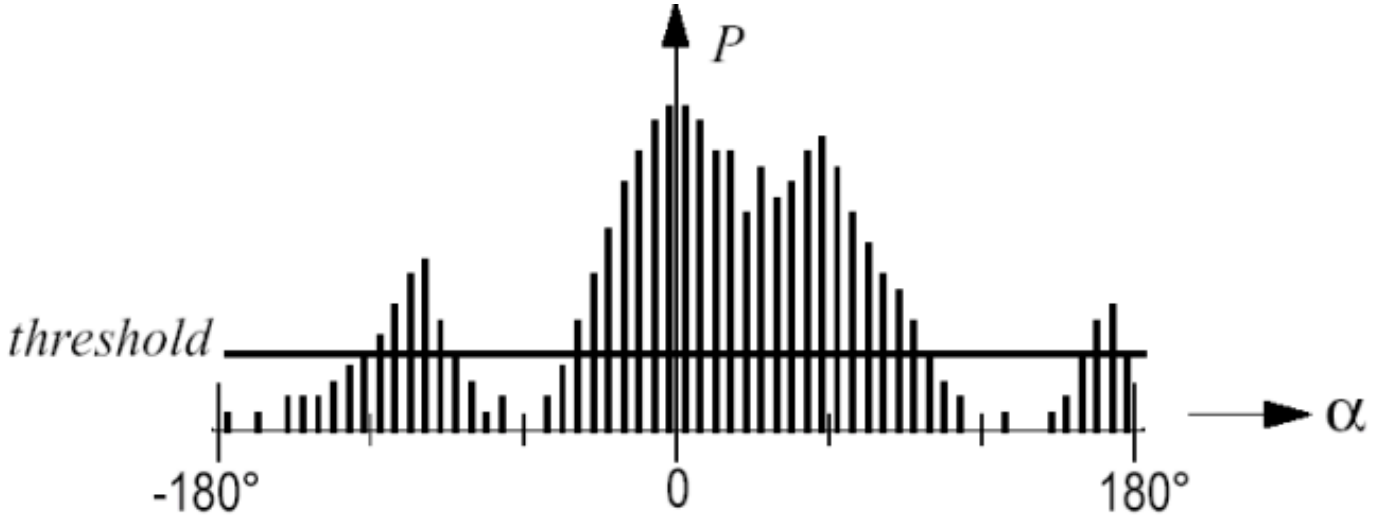
"VFH*: Local Obstacle with Look-Ahead Verification," 2000

Vector Field Histogram

ROBOT INTELLIGENCE LAB

- The vector field histogram (VFH) is a reactive navigation algorithm suitable for using LiDar sensors.
- In VFH, the environment is represented in a two-dimensional **occupancy grid** map with the probability that there is an obstacle.
- Then, the **polar histogram** is constructed from the observation where the VFH algorithm choses the opening with the lowest cost.



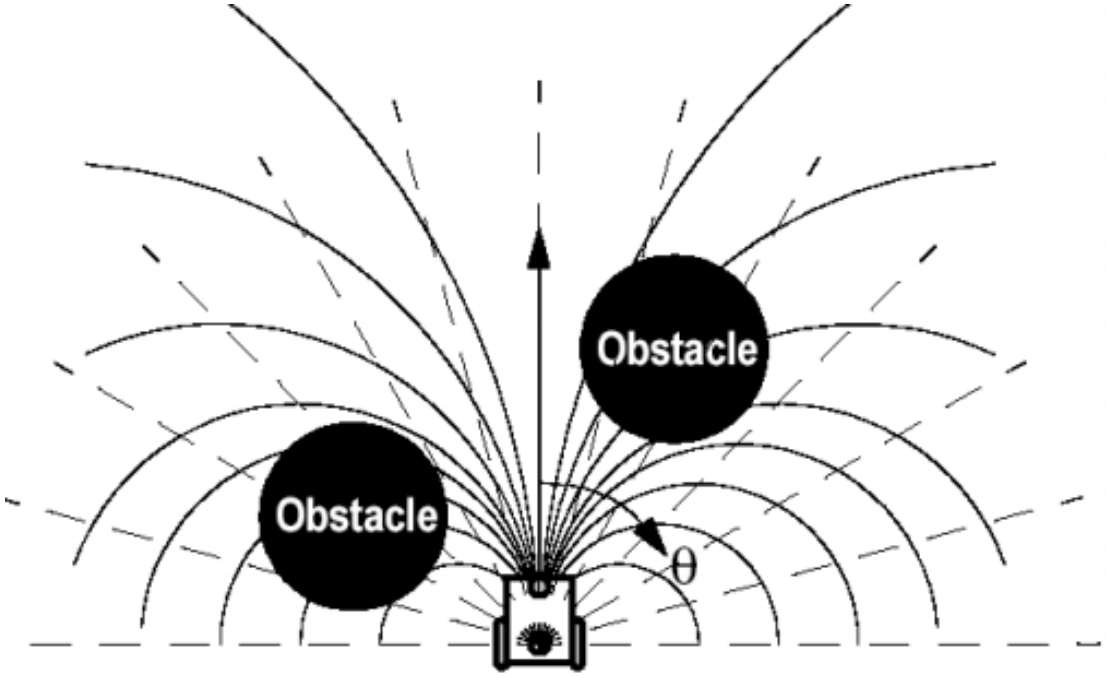


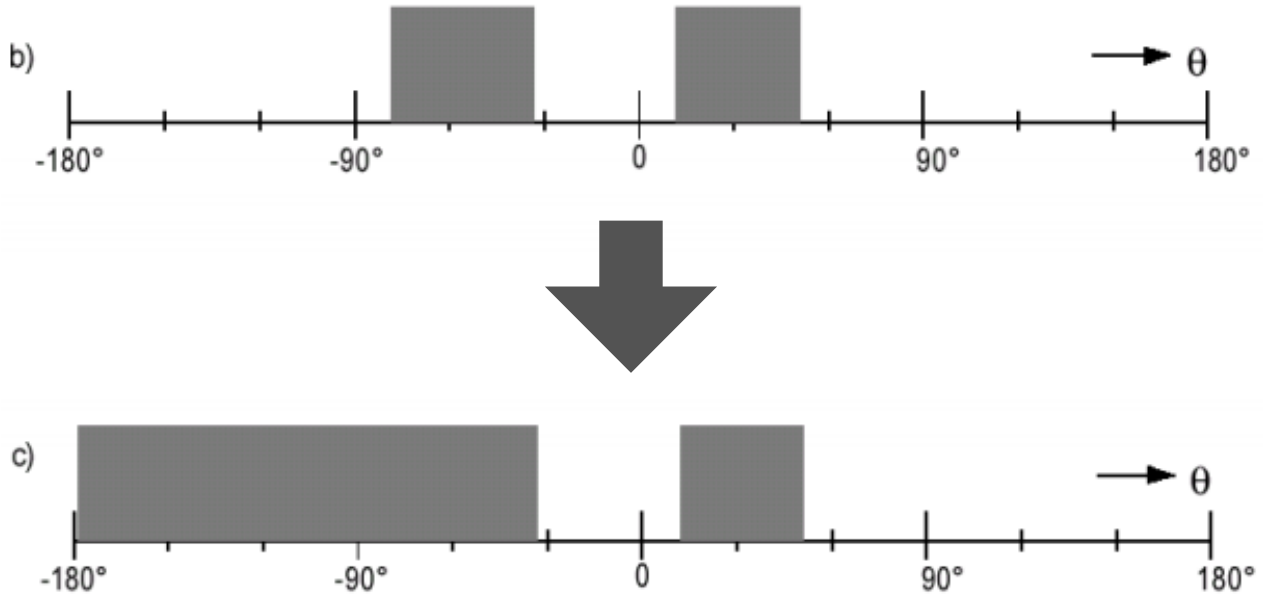
http://web.eecs.utk.edu/~leparker/Courses/CS594-fall08

VFH++

ROBOT INTELLIGENCE LAB

- "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots," 1997
  - **VFH+** extends the original VFH by considering mobile robot dynamics.

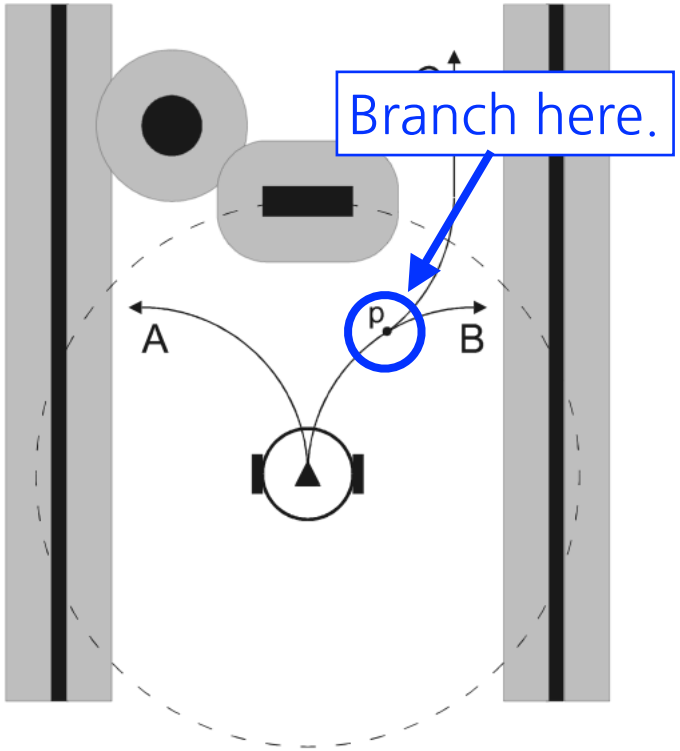




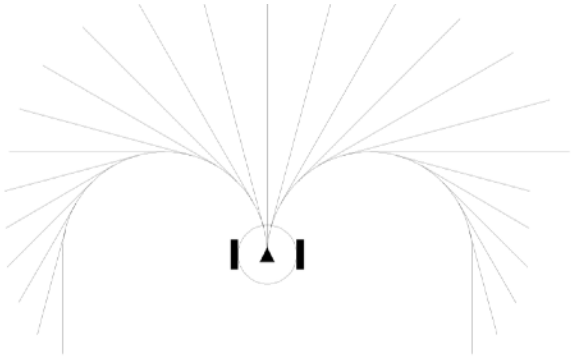
VFH*

ROBOT INTELLIGENCE LAB

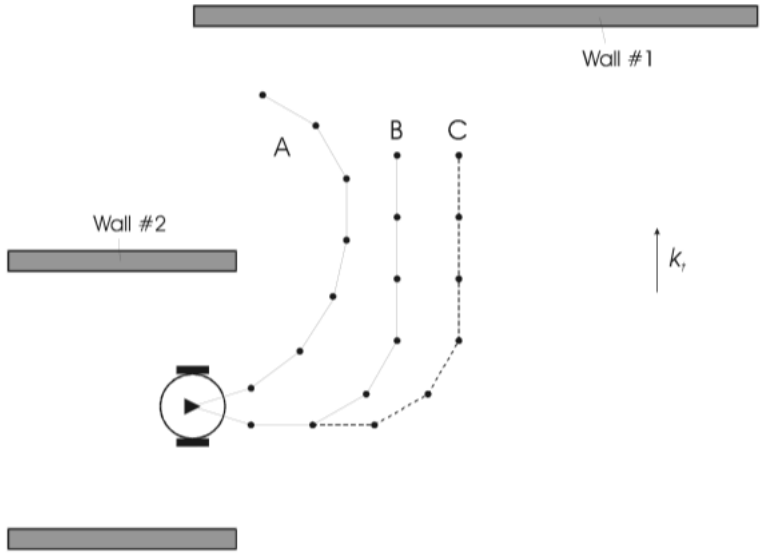
- "VFH*: Local Obstacle with Look-Ahead Verification," 2000
  - **VFH*** extends VFH and VFH+ with look ahead verification using A* algorithm.



Problematic situation



Trajectory approximation



Tree search (A*)

ROBOT INTELLIGENCE LAB

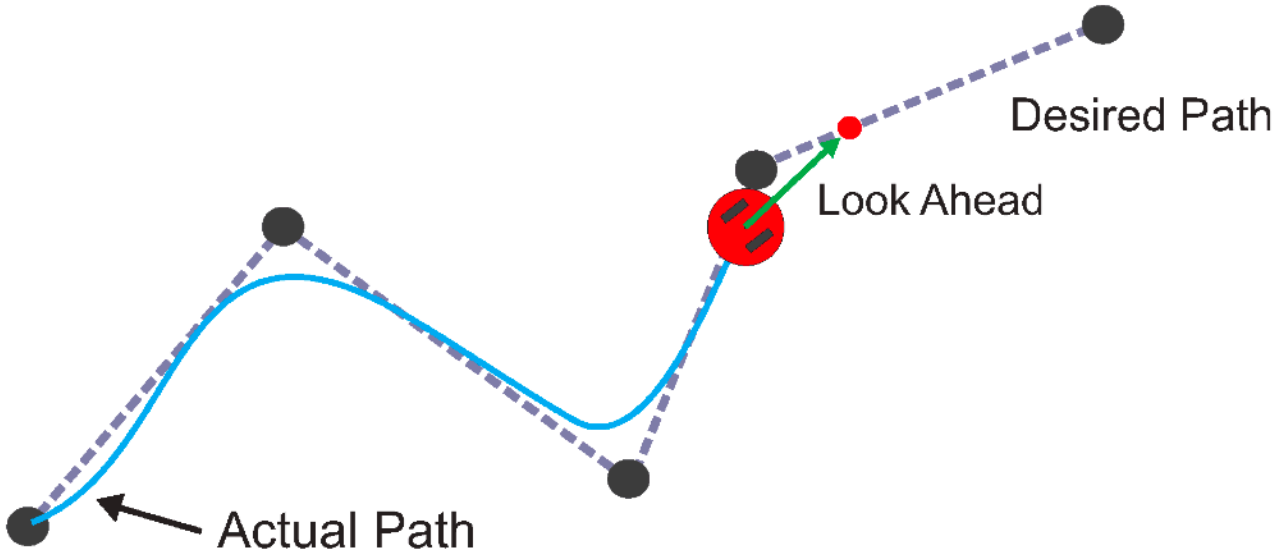
Pure Pursuit

"Implementation of the Pure Pursuit Path Tracking Algorithm," 1992

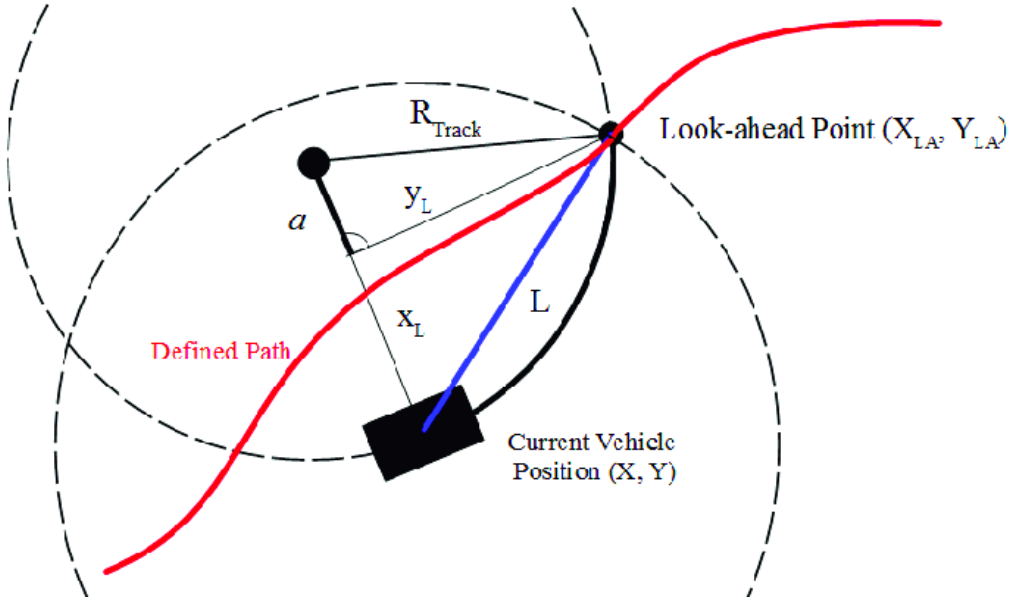
Pure Pursuit

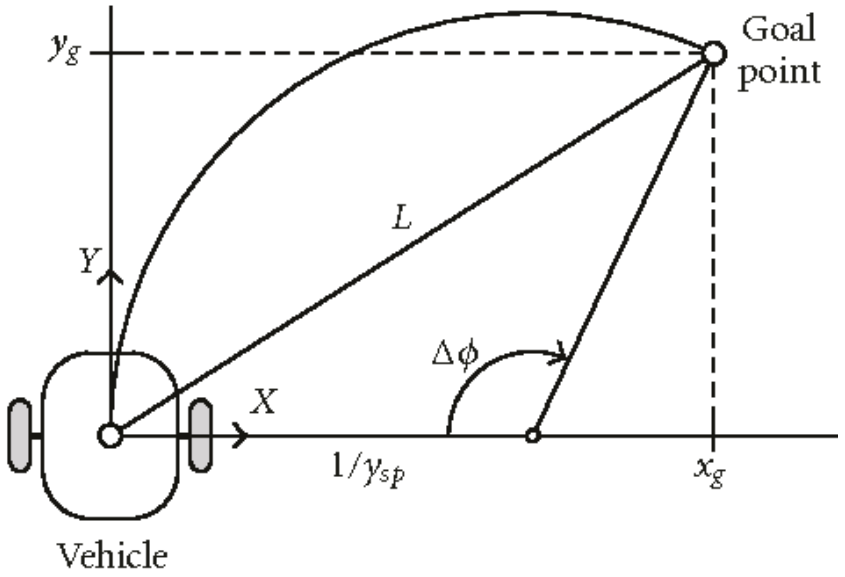
ROBOT INTELLIGENCE LAB

- Pure pursuit is a tracking algorithm that works by **calculating the curvature** that will move a vehicle from its current position to the goal position.
- The importance point of the algorithm is to choose the goal position (a **lookahead point**) that is some distance ahead of the vehicle on the path.



Pure Pursuit



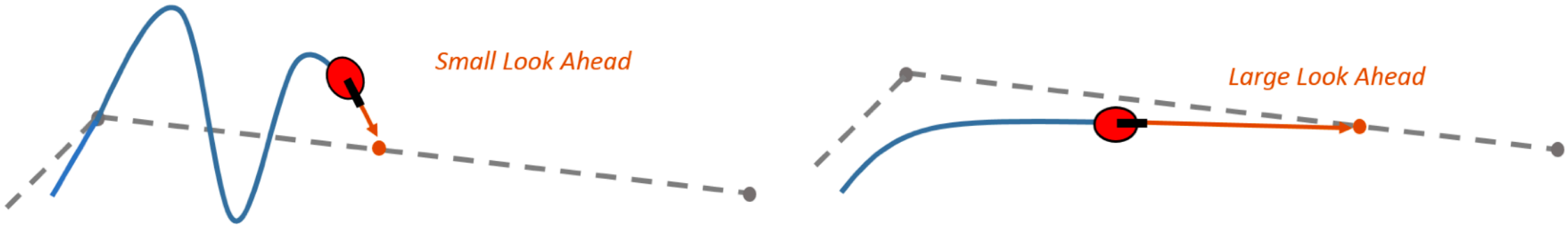


- Outline of the algorithm
  - Given the desired path to track, find the **look ahead point**.
  - Compute the **curvature** with the constant velocity assumption.
  - Update and iterate.

Pure Pursuit

ROBOT INTELLIGENCE LAB

- Limitations



  - A sharp change in curvature can be requested at a high speed, causing the vehicle's rear end to skid.
  - The vehicle will not be close on the path as quickly as desired due to the first order lag in steering.
  - Hard to find the optimal lookahead distance.

ROBOT INTELLIGENCE LAB

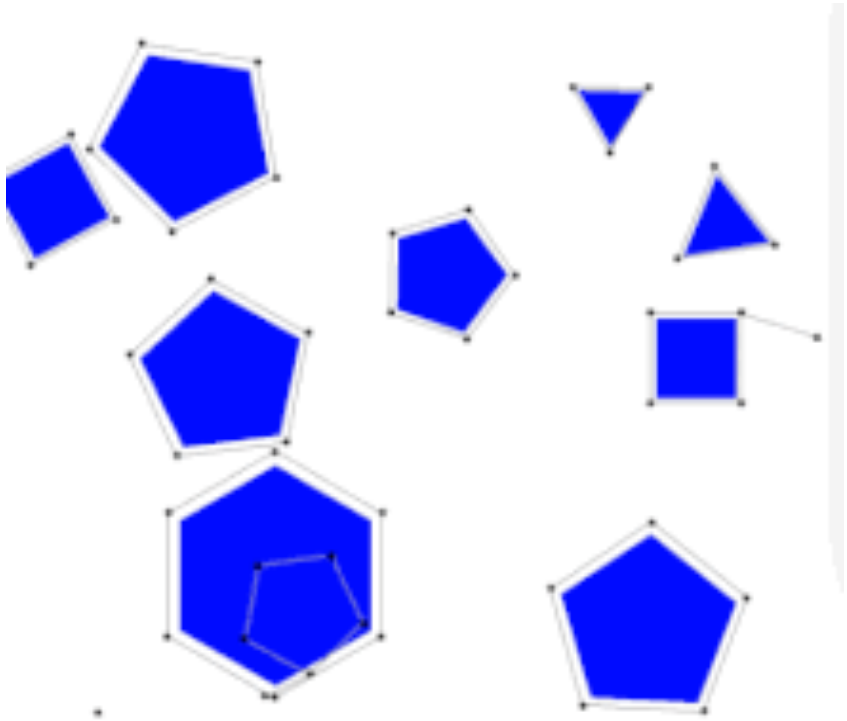
Probabilistic Roadmap

"Probabilistic Roadmaps for Path Planning High-Dimensional Configuration Spaces," 1996

Probabilistic Roadmap

ROBOT INTELLIGENCE LAB

- Probabilistic Roadmap (PRM) is a motion planning algorithm that solves the problem of determining a path between a start and a goal configuration while avoiding collisions.



https://en.wikipedia.org/wiki/Probabilistic_roadmap

Probabilistic Roadmap

ROBOT INTELLIGENCE LAB

- **PRM** proceeds in two phases:
  - **Learning phase**: a probabilistic roadmap is constructed and stored as a graph whose nodes correspond to collision-free configurations and whose edges correspond to feasible paths. These paths are computed using a simple and fast local planner.
  - **Query phase**: any given start and goal configurations, the roadmap is then searched (e.g., Dijkstra's algorithm or A* algorithm) for a path joining these two nodes.

- While the **learning phase** takes some time, the **query phase** can be extremely fast. However, it may not be useful for reactive planning.

Dynamic Window Approach

"Controlling synchro-drive robots with the dynamic window approach to collision avoidance," 1996

Dynamic Window Approach

ROBOT INTELLIGENCE LAB

- Dynamic window approach (DWA) is **reactive collision avoidance** algorithm for mobile robots. DWA is derived directly from the **motion dynamics** making it suitable for robots operating at high speed.
  - It has been successfully used to control RHINO with speeds of up to 95 cm/sec.

- The core idea is the construction of the **dynamic window** which consists of the velocities reachable within a short time interval.

RHINO

Dynamic Window Approach

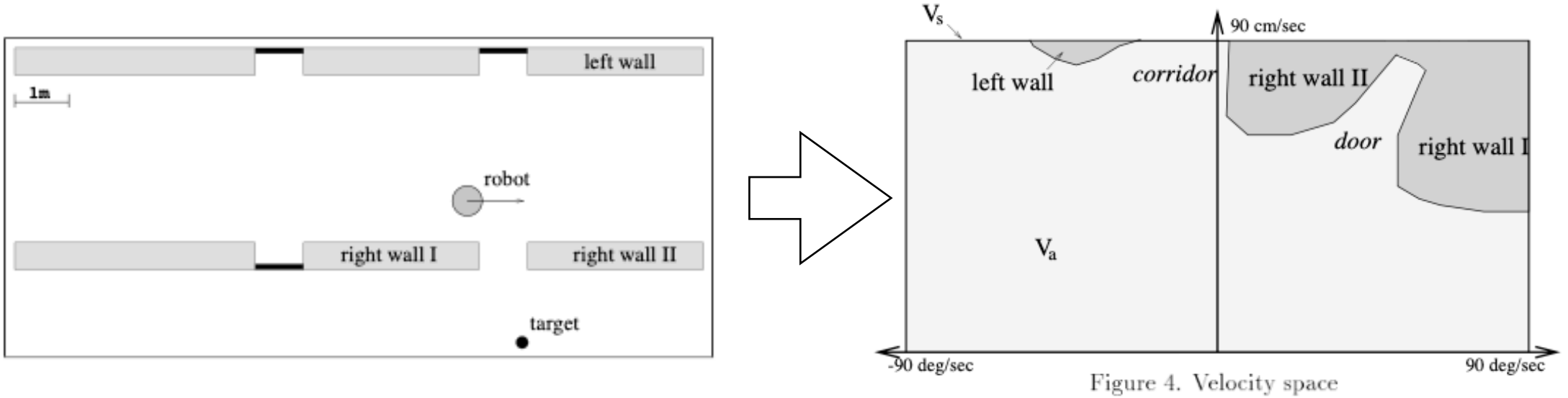


Figure 4. Velocity space

- The future trajectories of a robot is approximated with **circulars arcs** (curvatures) uniquely determined by the velocity vector (v_i, w_i) where It assumes the **constant velocity**.
- The velocity is considered **admissible** if the robot is able to stop before it reaches the closest obstacle.
- The **dynamic window** restricts the admissible velocities to those that can be reached within a short time interval given the **limited accelerations** of the robot.

Dynamic Window Approach

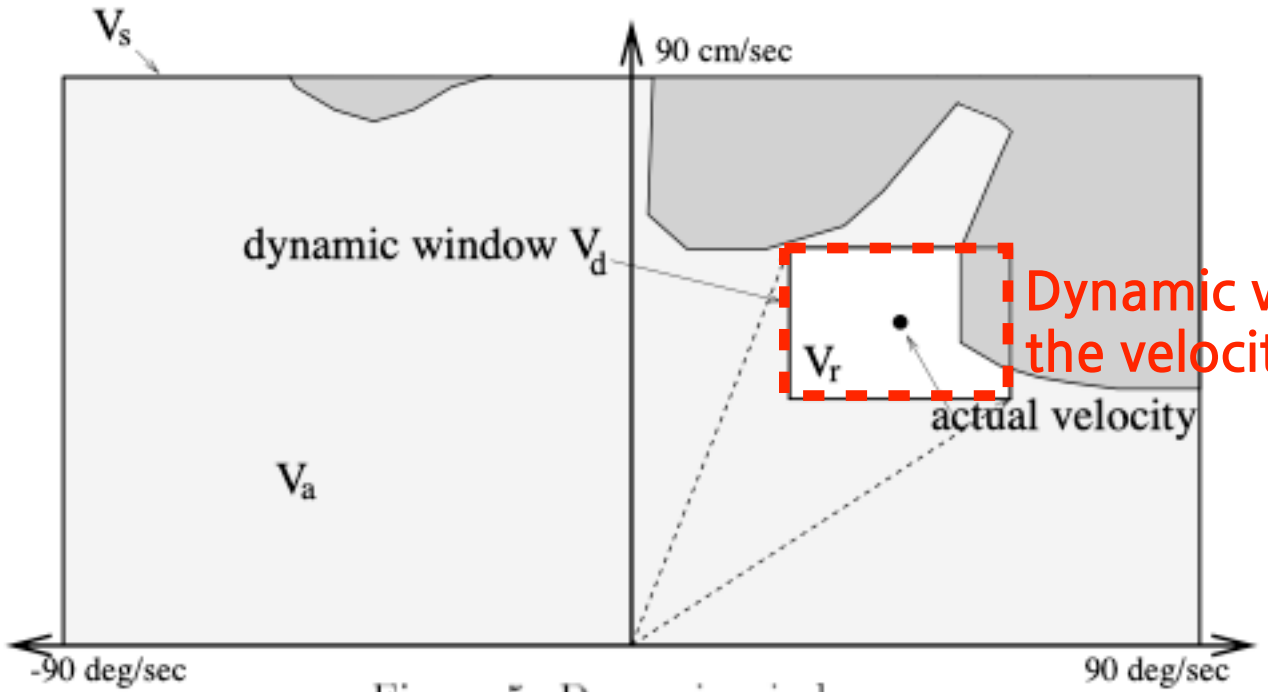


Figure 5. Dynamic window

Dynamic window within the velocity space.

Dynamic Window Approach

ROBOT INTELLIGENCE LAB

ROBOT INTELLIGENCE LAB

Velocity Obstacles

"Motion Planning in Dynamic Environments using Velocity Obstacles," 1998

Velocity Obstacles

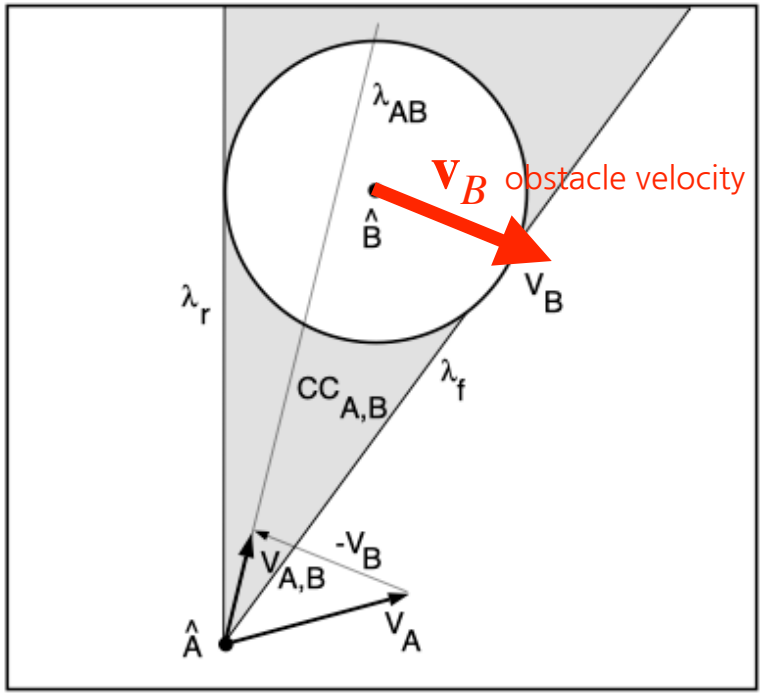
ROBOT INTELLIGENCE LAB

- The velocity obstacles (VO) algorithm presents obstacle avoidance maneuvers to avoid static and moving obstacles in the **velocity space** based on the current positions and velocities of the robot and obstacles.

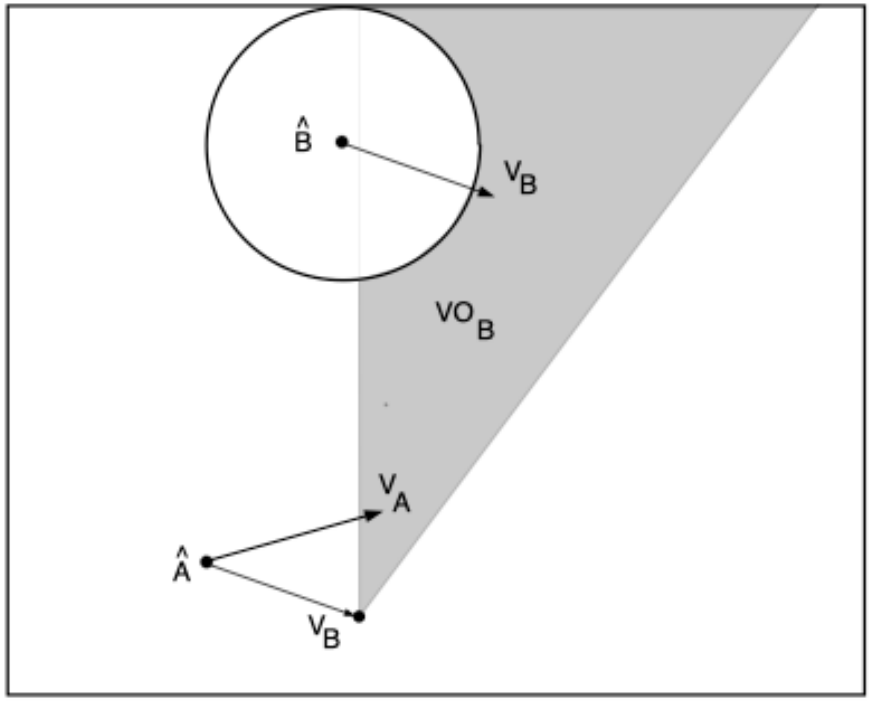
- The avoidance maneuvers are generative by selecting robot velocities outside of the velocity obstacles, which represents the set of robot velocities that would result in a collision with a given obstacle that moves at a given (constant) velocity, at some future time.

Velocity Obstacles

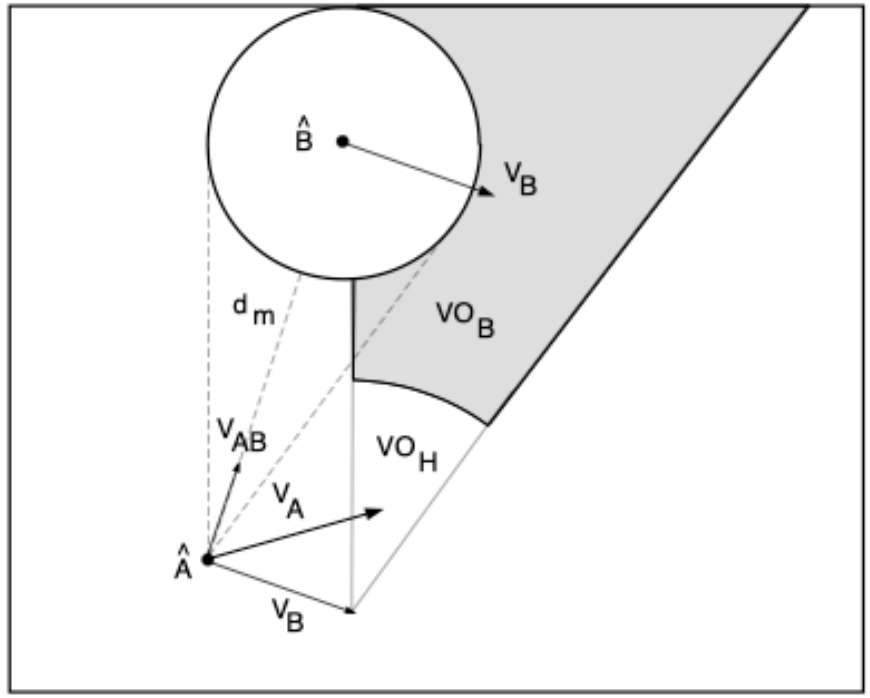
ROBOT INTELLIGENCE LAB



Collision Cone $CC_{A,B}$

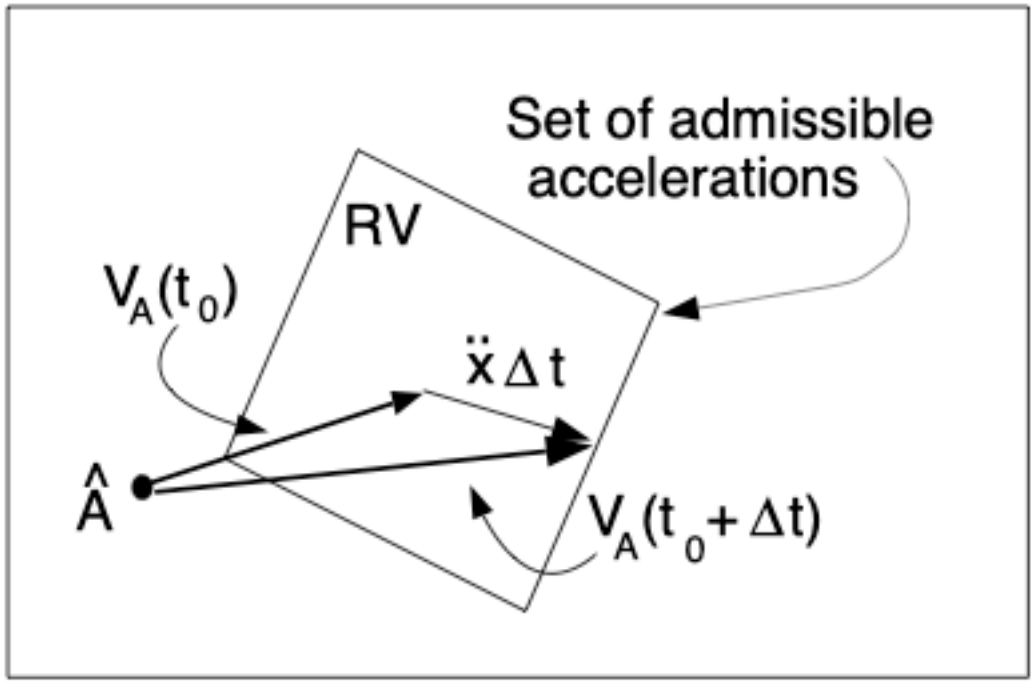


Velocity Obstacle $VO = CC_{A,B} \oplus \mathbf{v}_B$

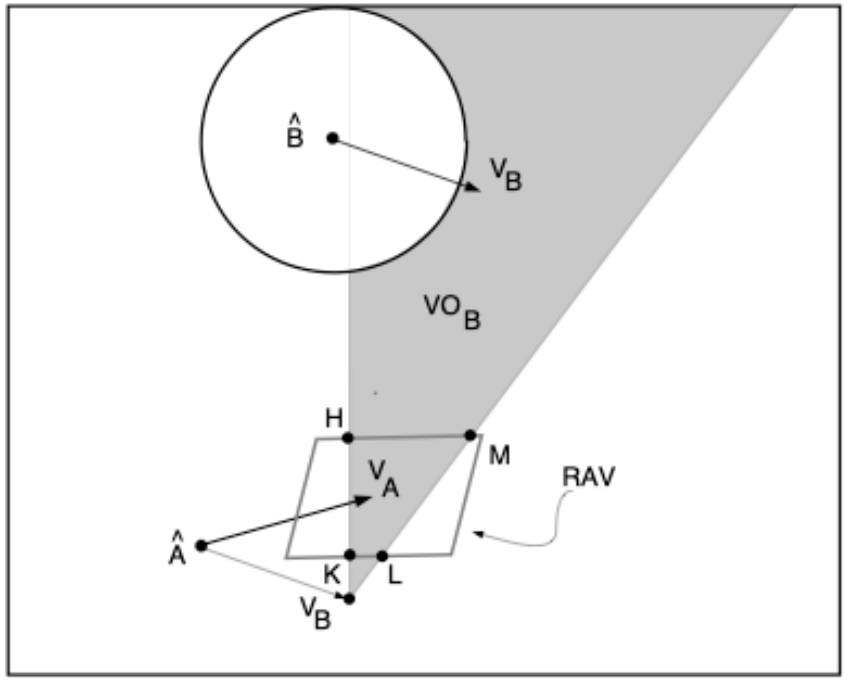


VO for a short time horizon

Velocity Obstacles

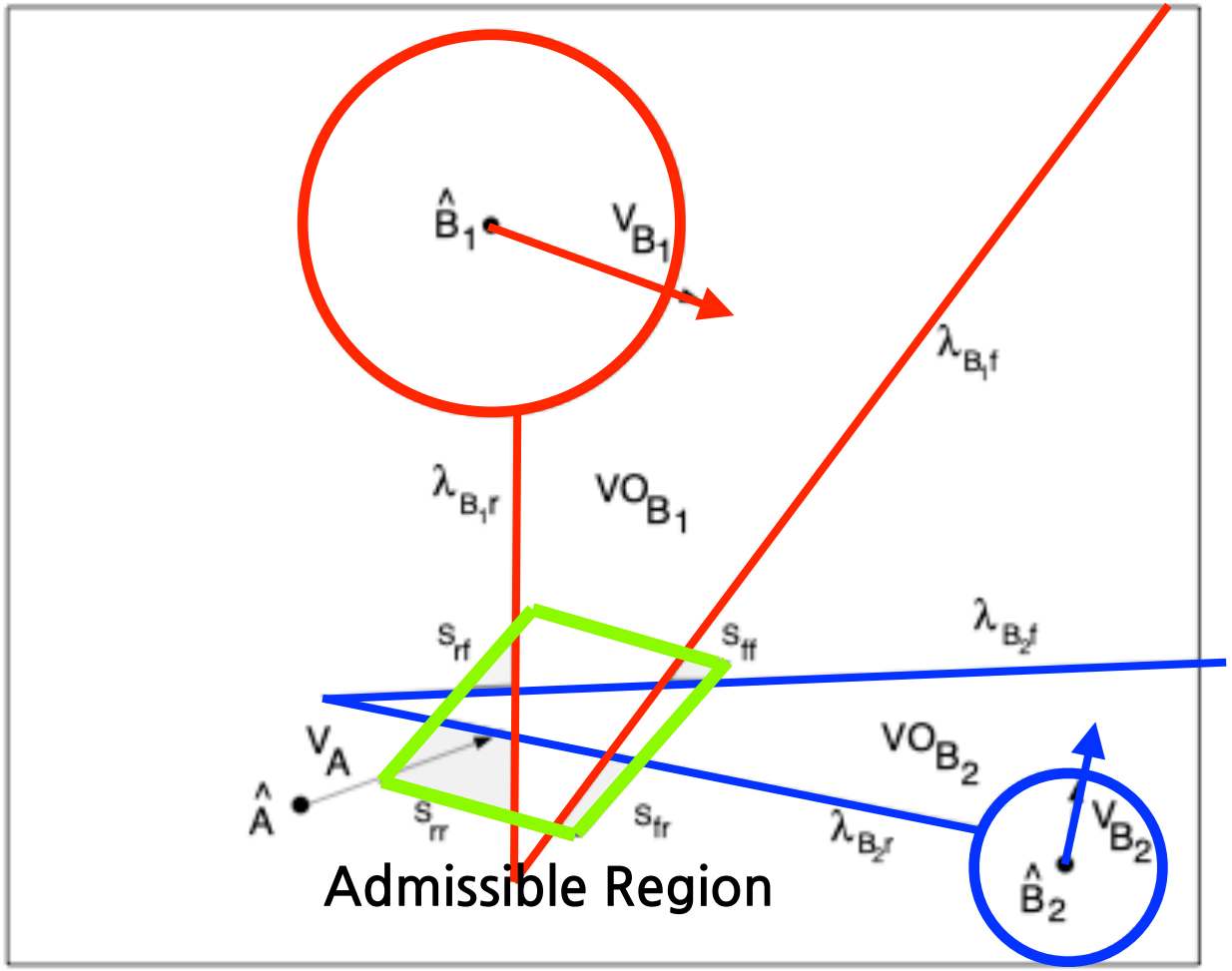


The action is selected within an admissible set.



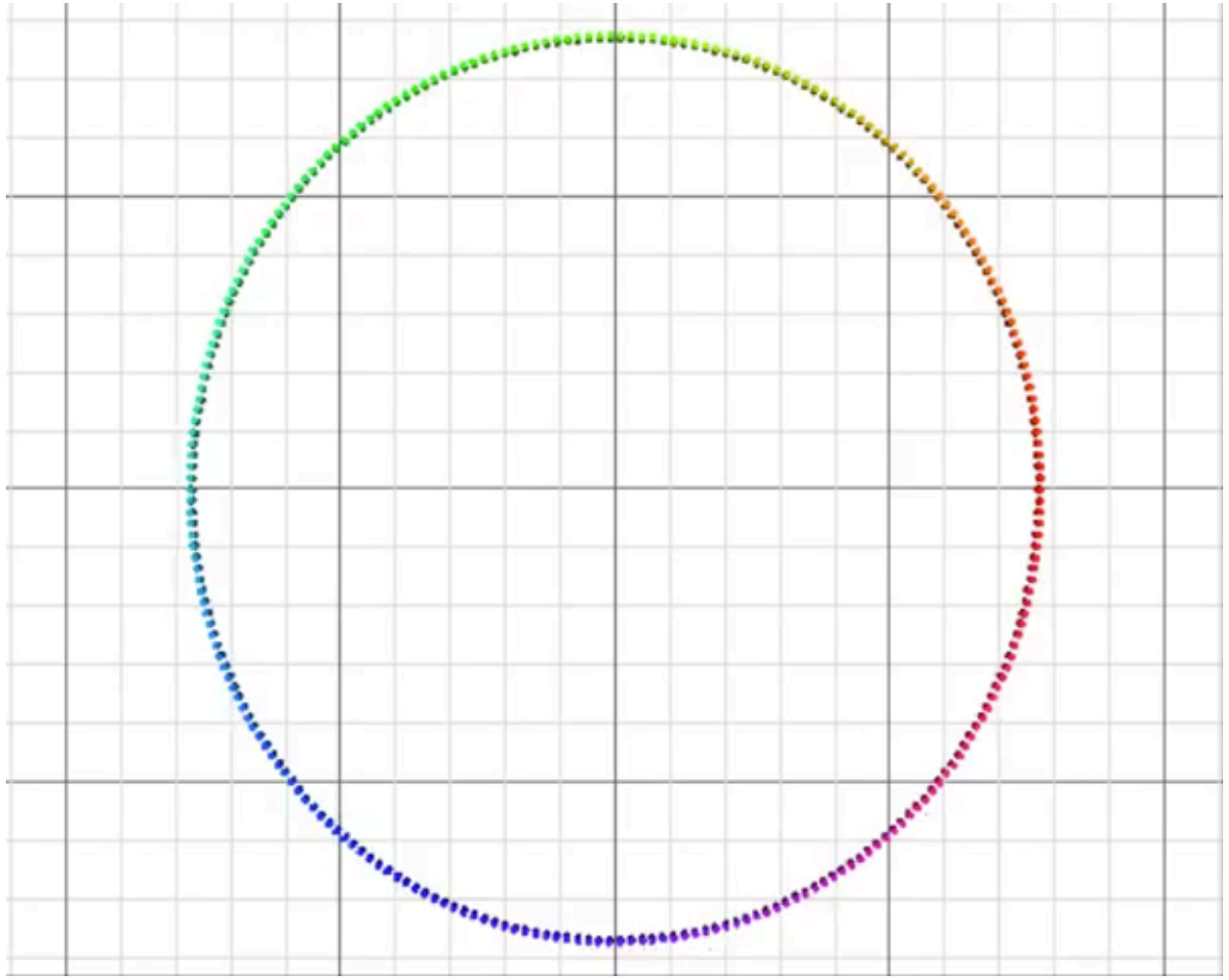
The reachable avoidance velocities (RAV)

Velocity Obstacles



VO with Multiple Moving Obstacles.

Velocity Obstacles



- Limitations
  - Positions and velocities of obstacles should be known.
  - Not straightforward to hand non-circle shaped obstacles (e.g., polygon).

ROBOT INTELLIGENCE LAB

Rapidly-Exploring Random Trees

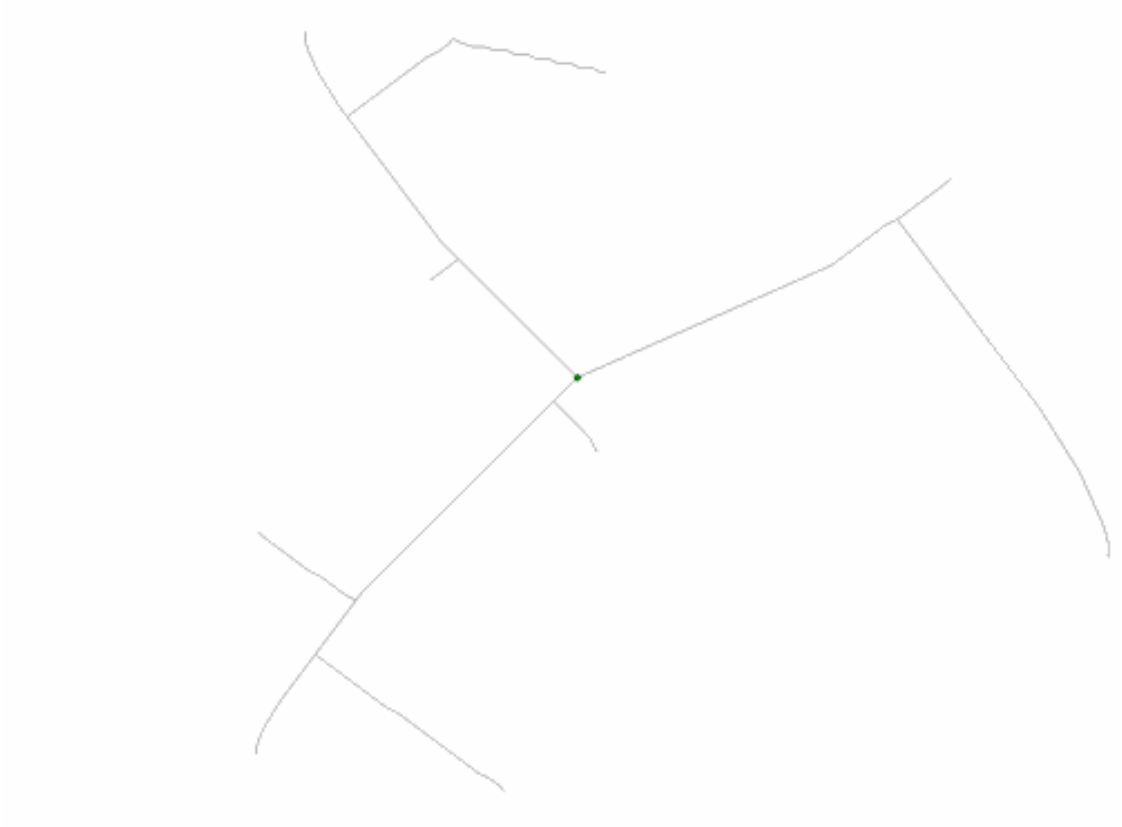
"Rapidly-Exploring Random Trees: Progress and Prospects," 2001

"Incremental Sampling-based Algorithms for Optimal Motion Planning," 2010

Rapidly-Exploring Random Trees

ROBOT INTELLIGENCE LAB

- This the famous **RRT** algorithm.



https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

Rapidly-Exploring Random Trees

- RRT algorithm
  - **Loop** until goal reached:
    - Sample **a random configuration** (or the goal configuration)
    - Find the **nearest vertex** from the **RRT graph**
    - Select new **configuration** by expanding the **graph** from the **selected vertex**.
    - Add the **configuration** and an edge to the **graph**.

Rapidly-Exploring Random Trees

- Advantages
  - Single parameter (ϵ-greedy)
  - Balance between greedy search and exploration
  - Converges to sampling distribution in the limit (completeness)
  - Simple and easy to implement
- Disadvantages
  - Metric sensitivity
  - Unknown rate of convergence
  - Cannot guarantee the **optimality**

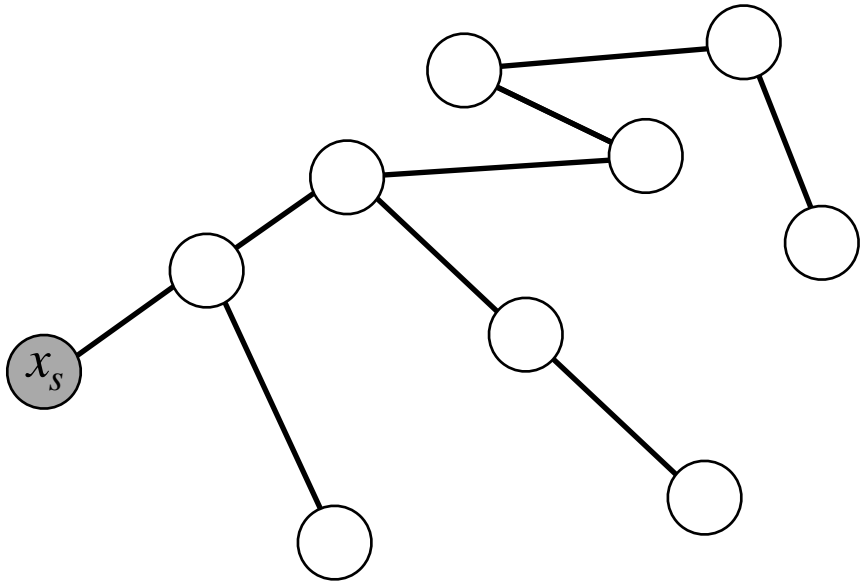
RRT*

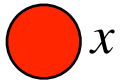
- "Incremental Sampling-based Algorithms for Optimal Motion Planning," 2010
  - RRT does not have any theoretical guarantees on the quality of the solution obtained by it.
  - This paper presents two approaches: Rapidly-exploring random graph (RRG) and the tree version of RRG (**RRT***).
  - Both methods converges to the **optimum** almost surely.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) >
           Cost(x_new) + c(Line(x_new, x_near)) then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```



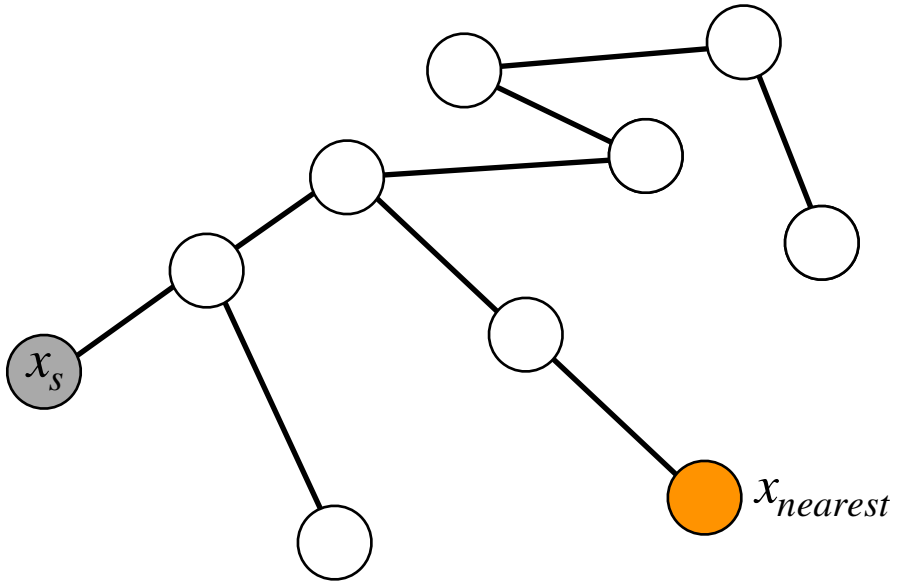


Randomly sample a configuration (or the goal configuration)

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$



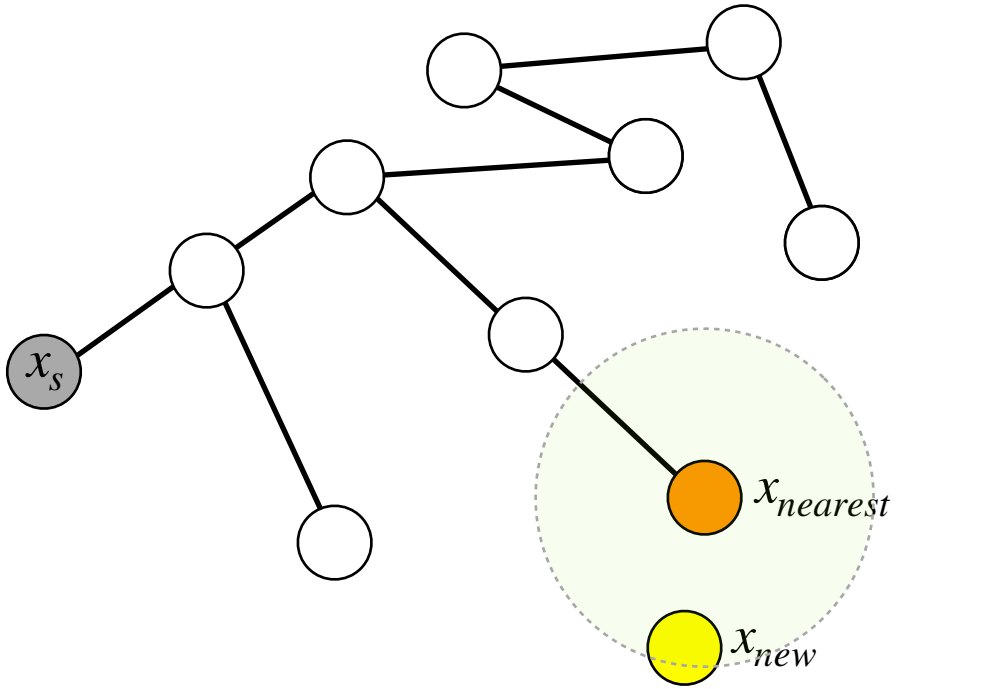
x

Find the nearest node in the graph.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$



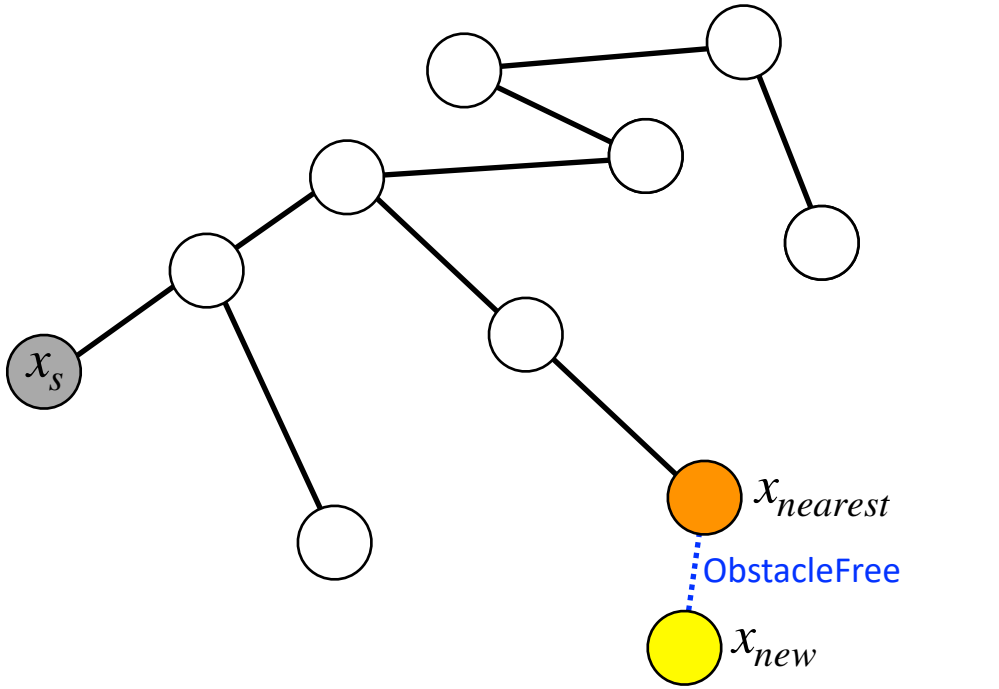
x

Sample a new configuration x_{new} closer to x than $x_{nearest}$.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$



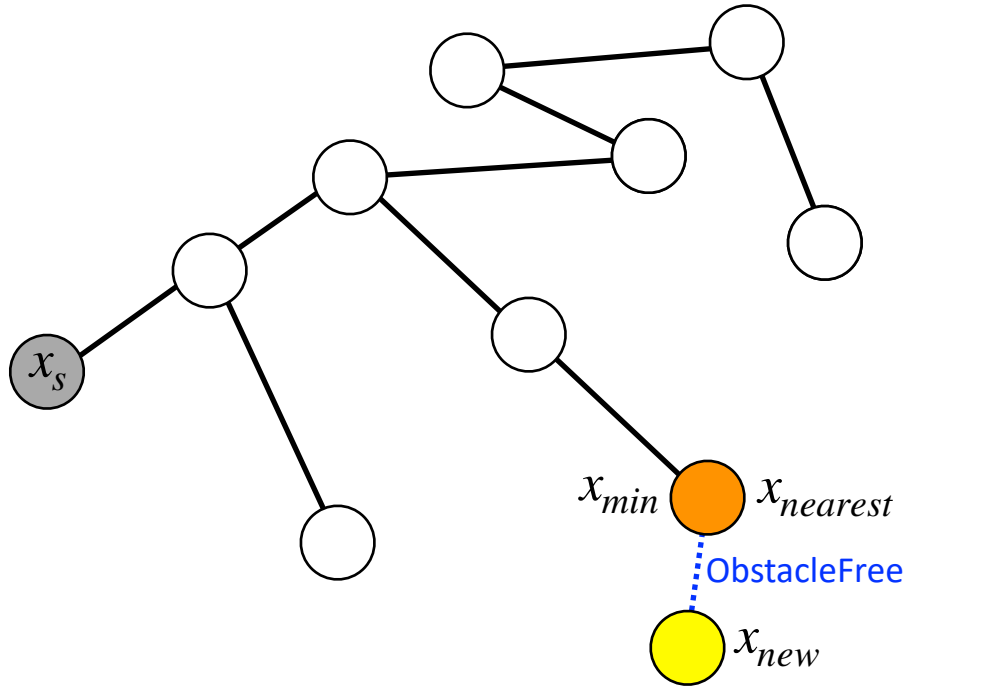
x

Check the feasibility of the connection.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$



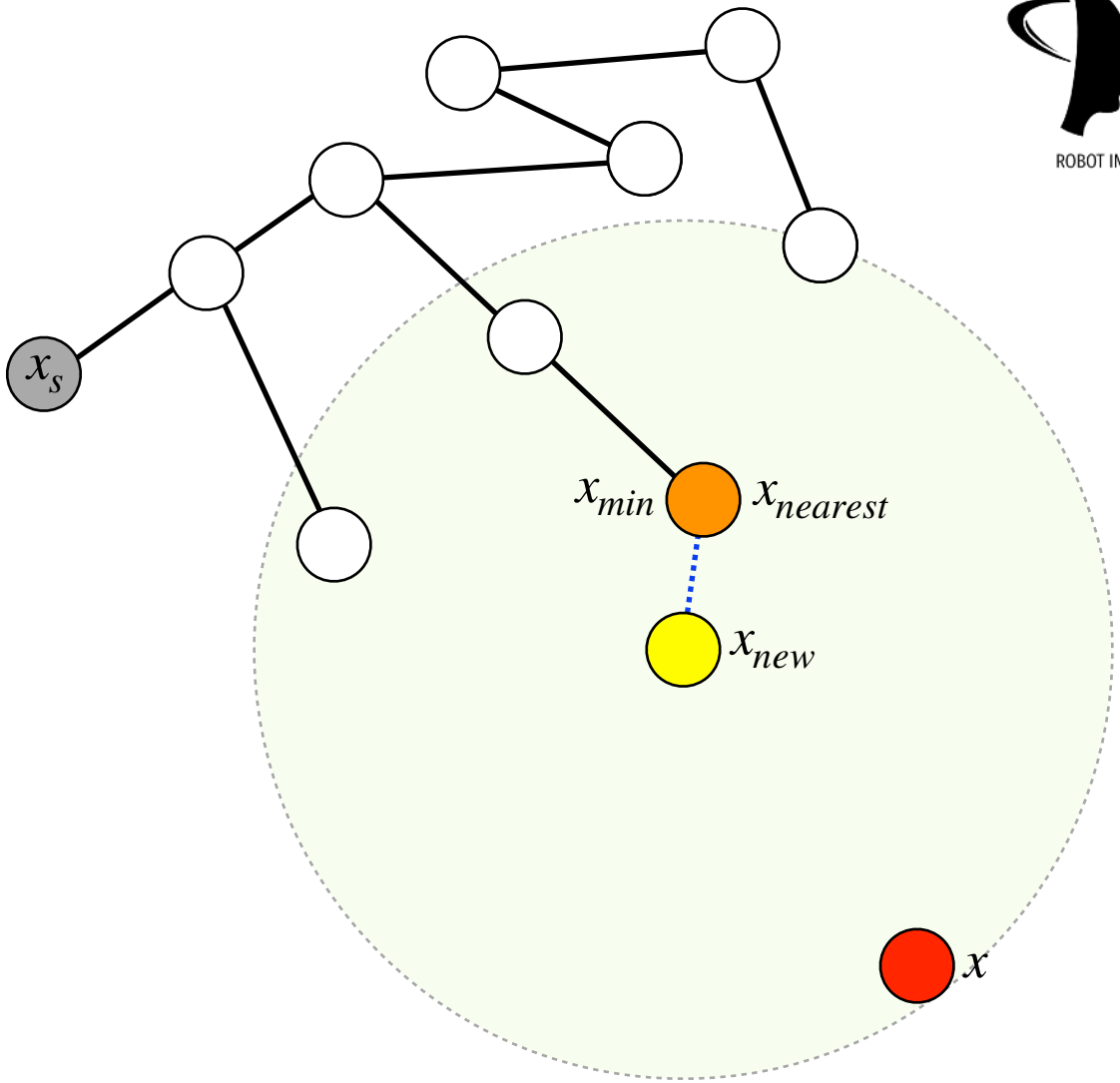
x

Assign x_{min} to $x_{nearest}$.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

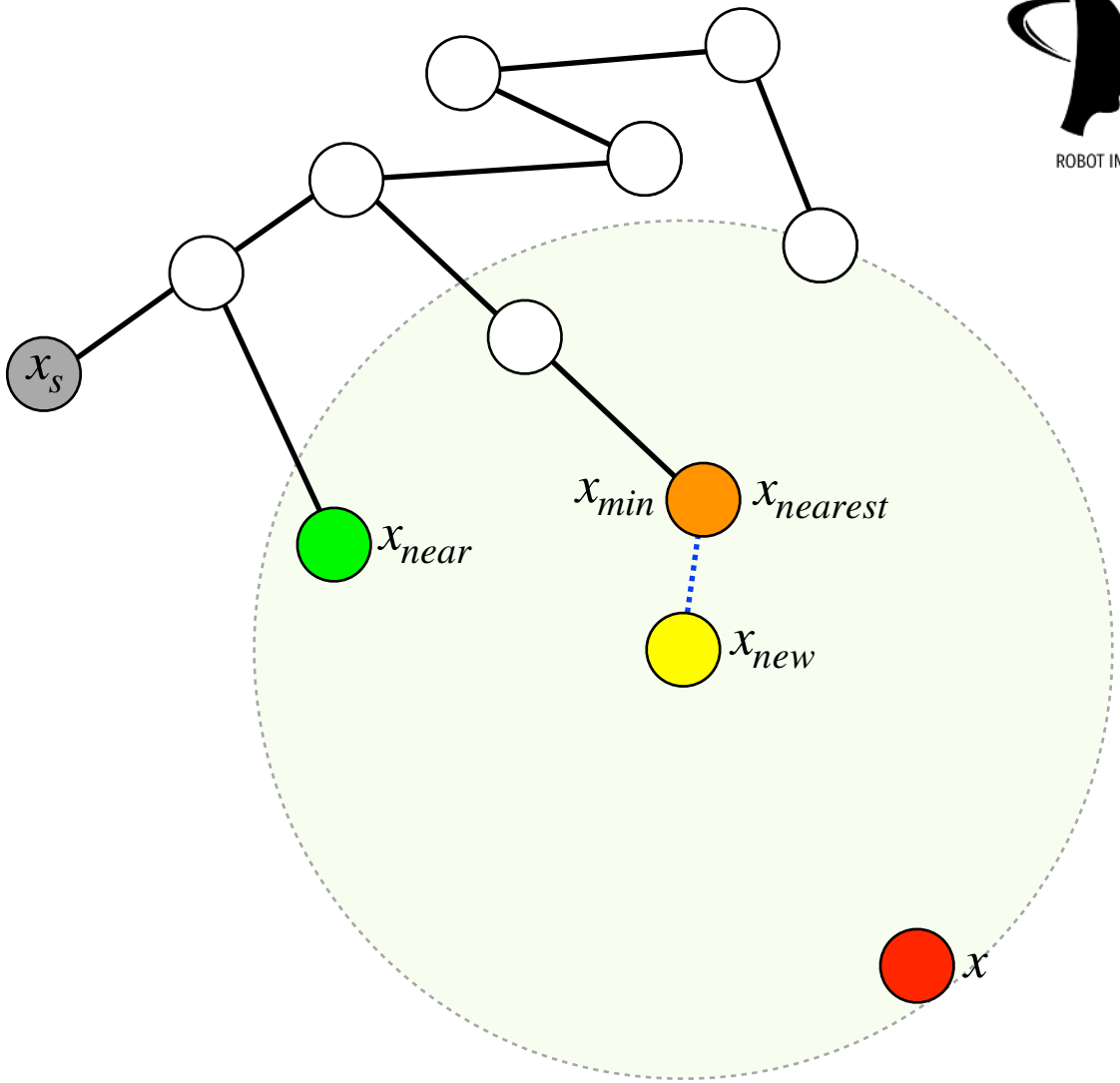


Select X_{near} a set of nodes centered around x_{new}

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

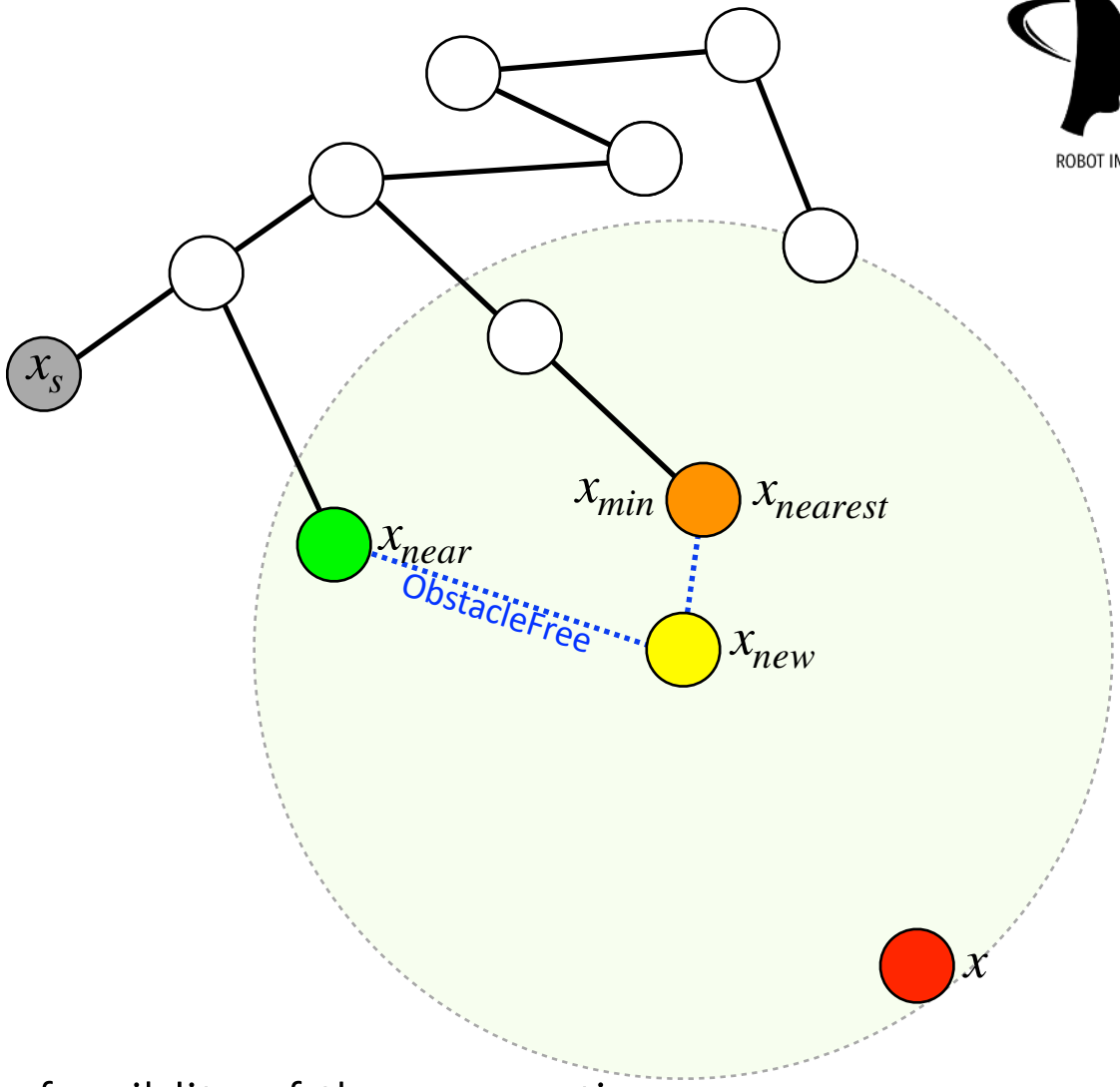


Select x_{near} in X_{near}.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

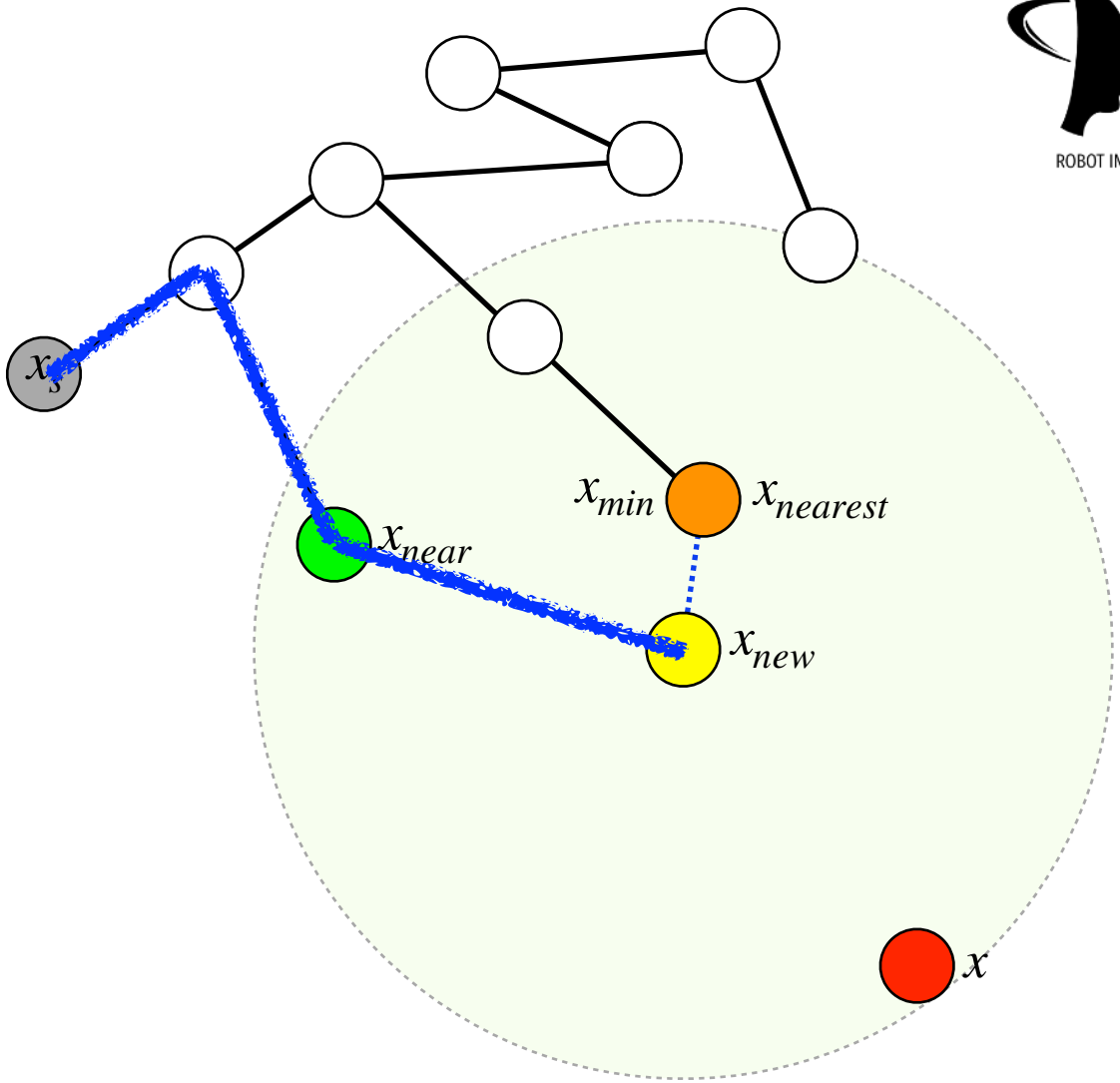


Check the feasiblity of the connection.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

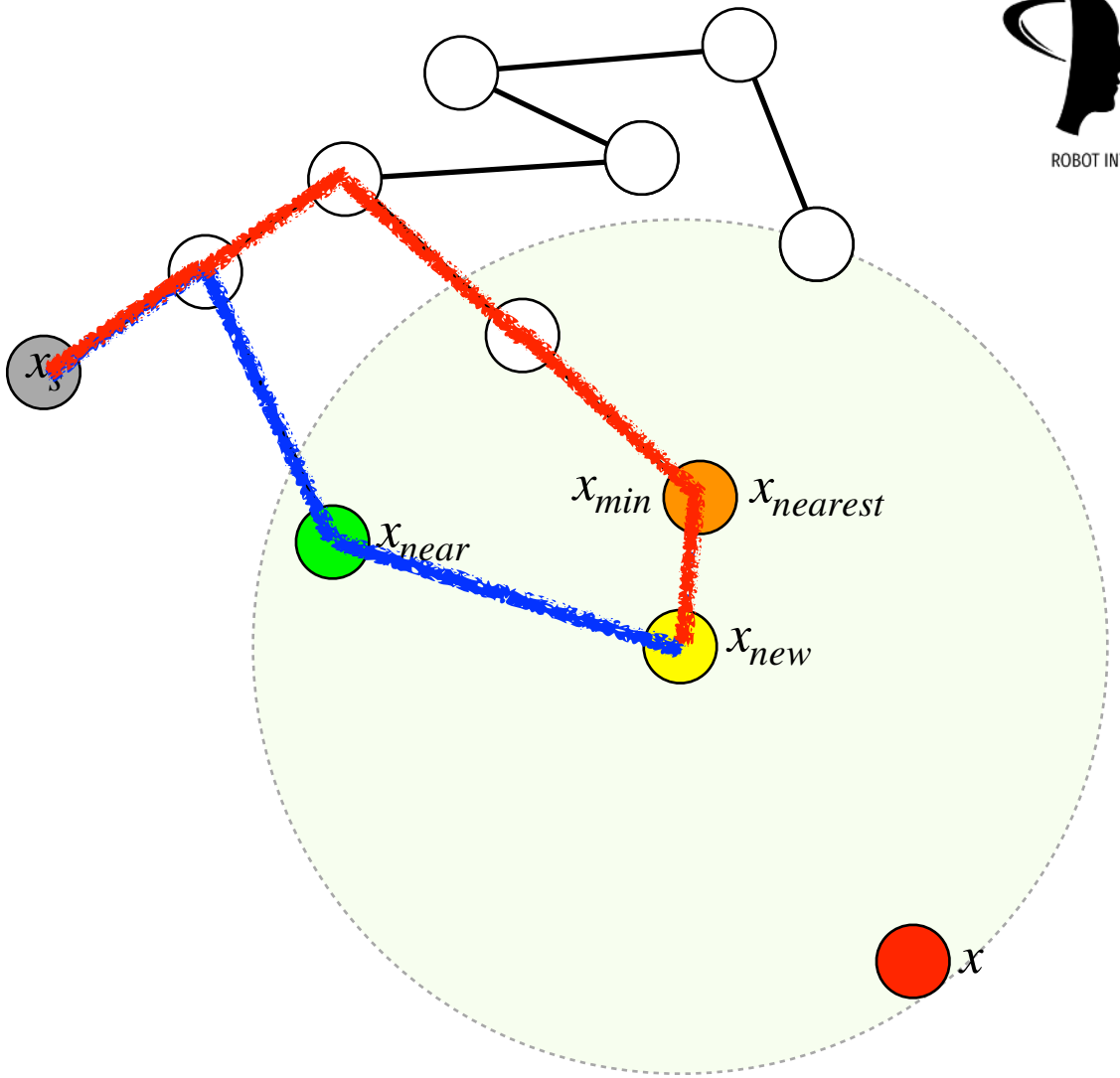


Compute the cost of x_{new} through x_{near}.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) >
           Cost(x_new) + c(Line(x_new, x_near)) then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```

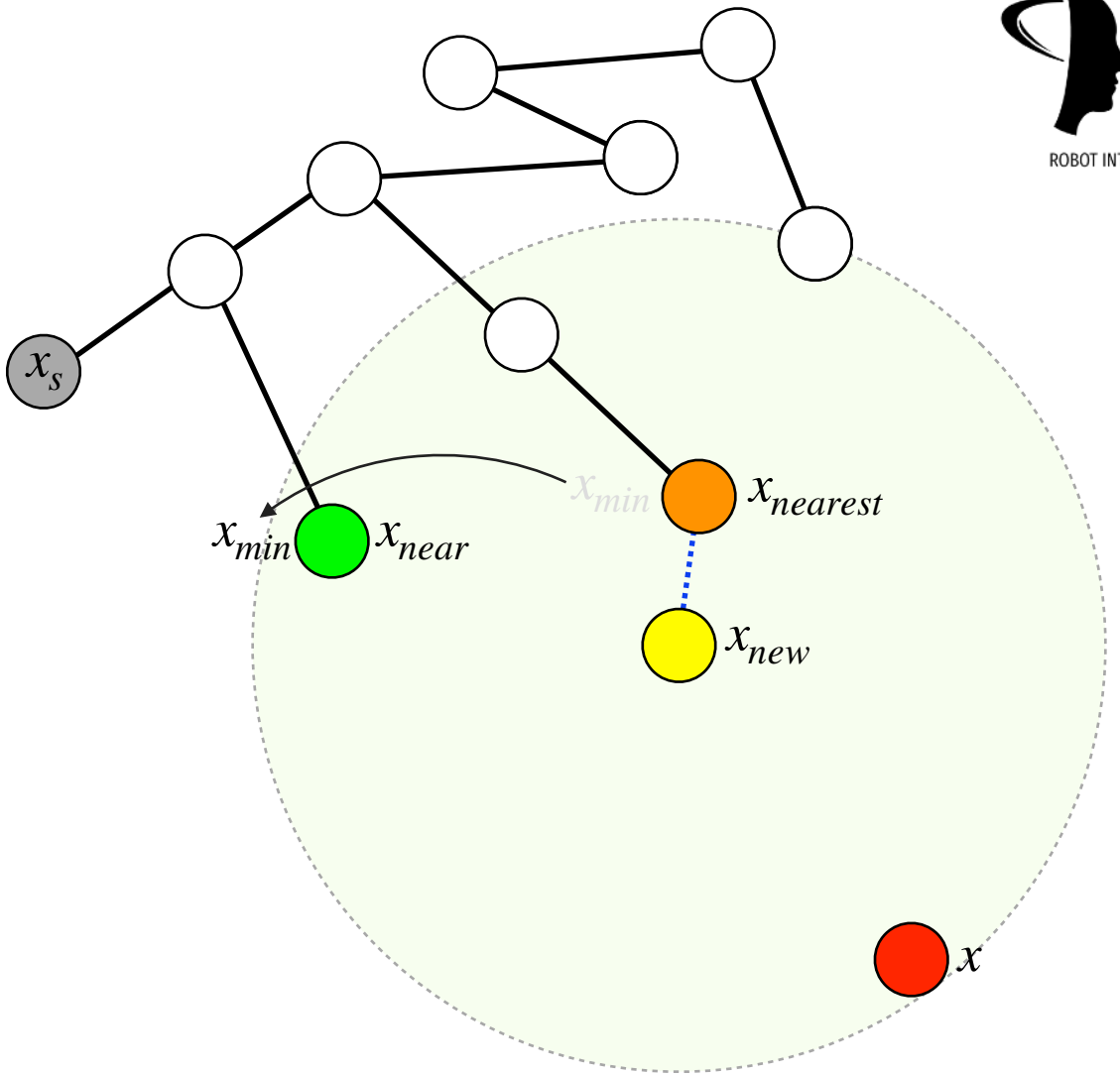


Compute the cost of x_{new} through $x_{nearest}$.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) >
           Cost(x_new) + c(Line(x_new, x_near)) then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```

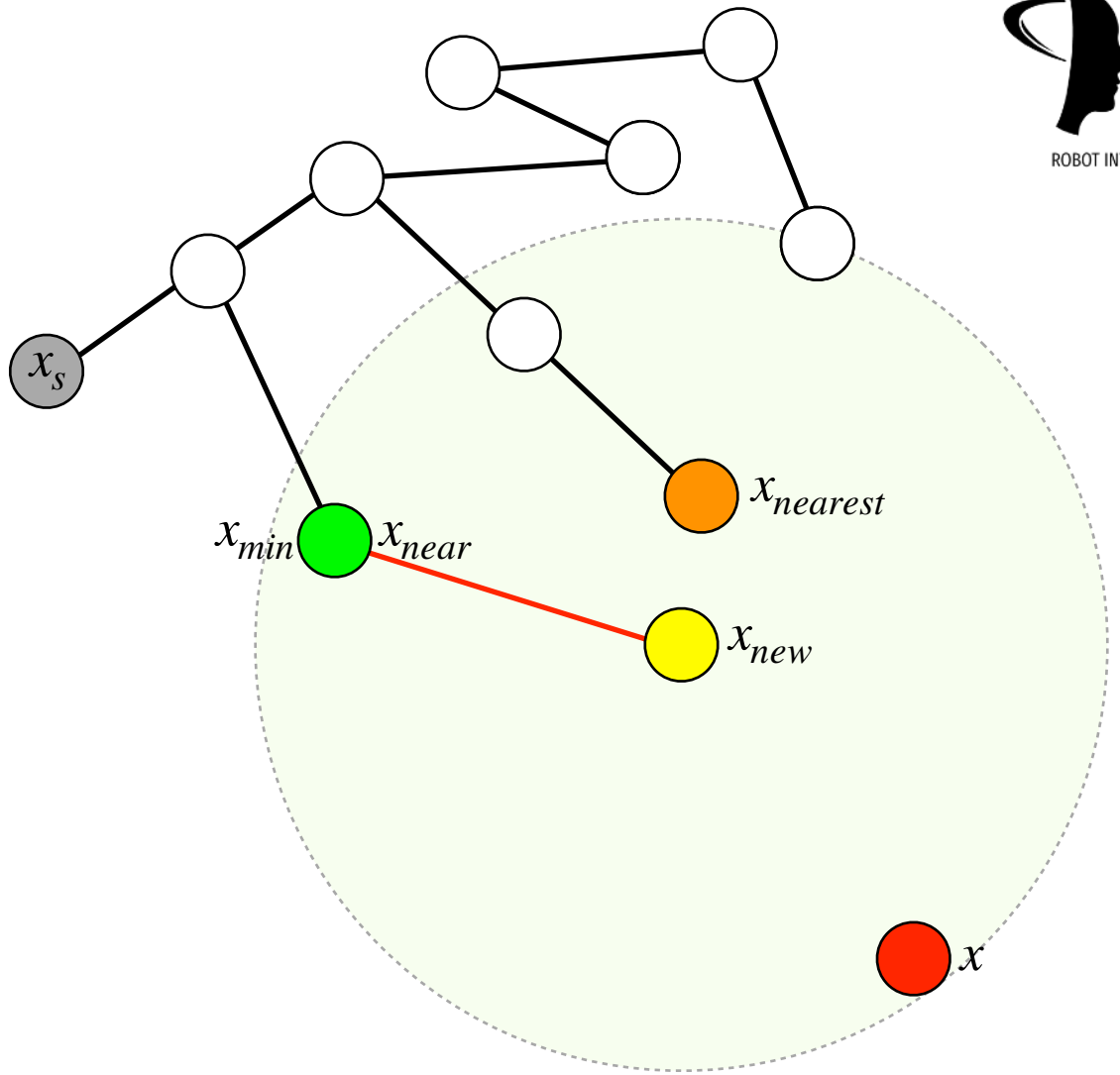


If the cost is smaller through x_{near}, assign x_{min} to x_{near}.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) >
           Cost(x_new) + c(Line(x_new, x_near)) then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```

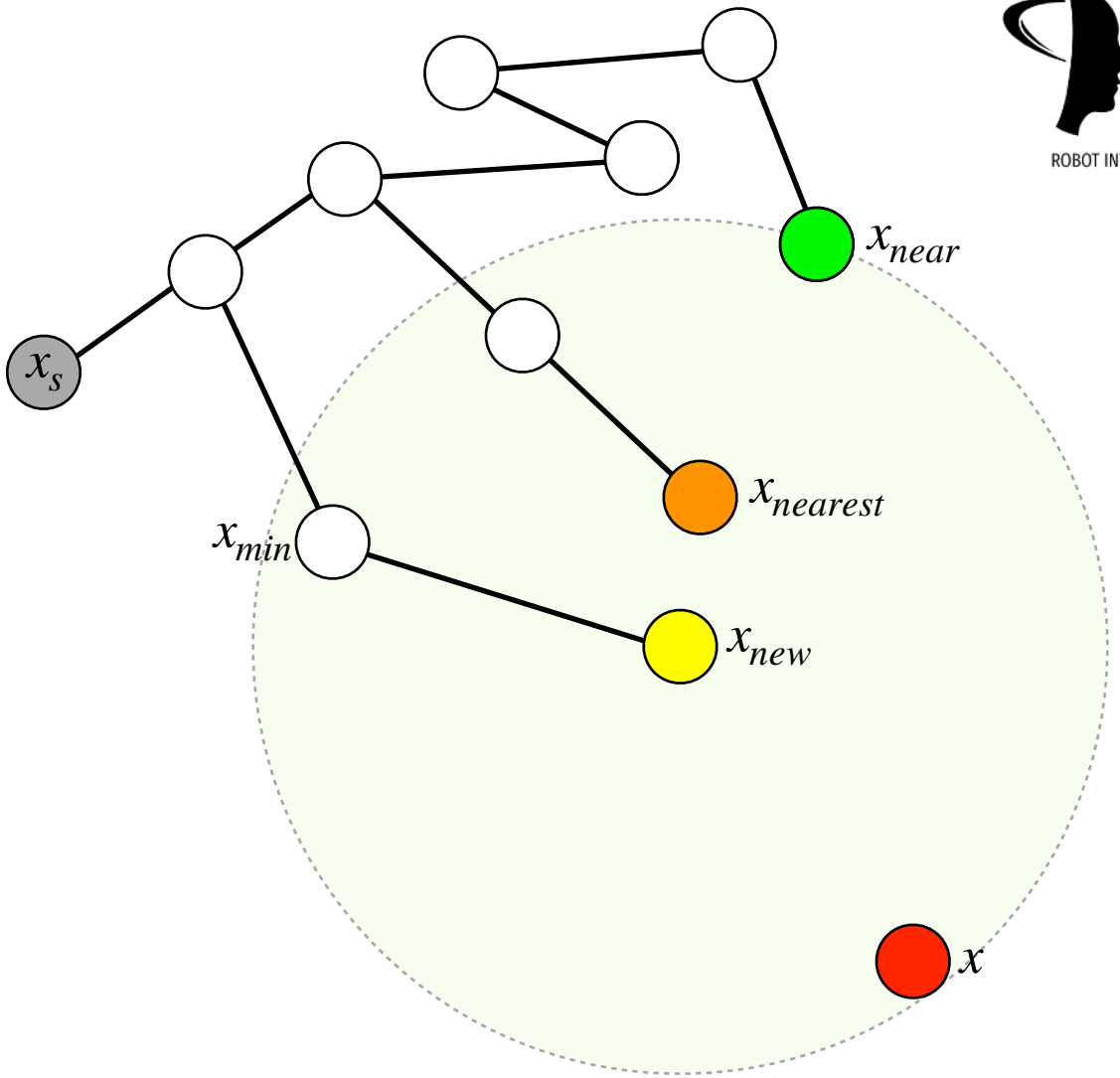


Make a connection between x_{min} and x_{new}.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) >
           Cost(x_new) + c(Line(x_new, x_near)) then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```

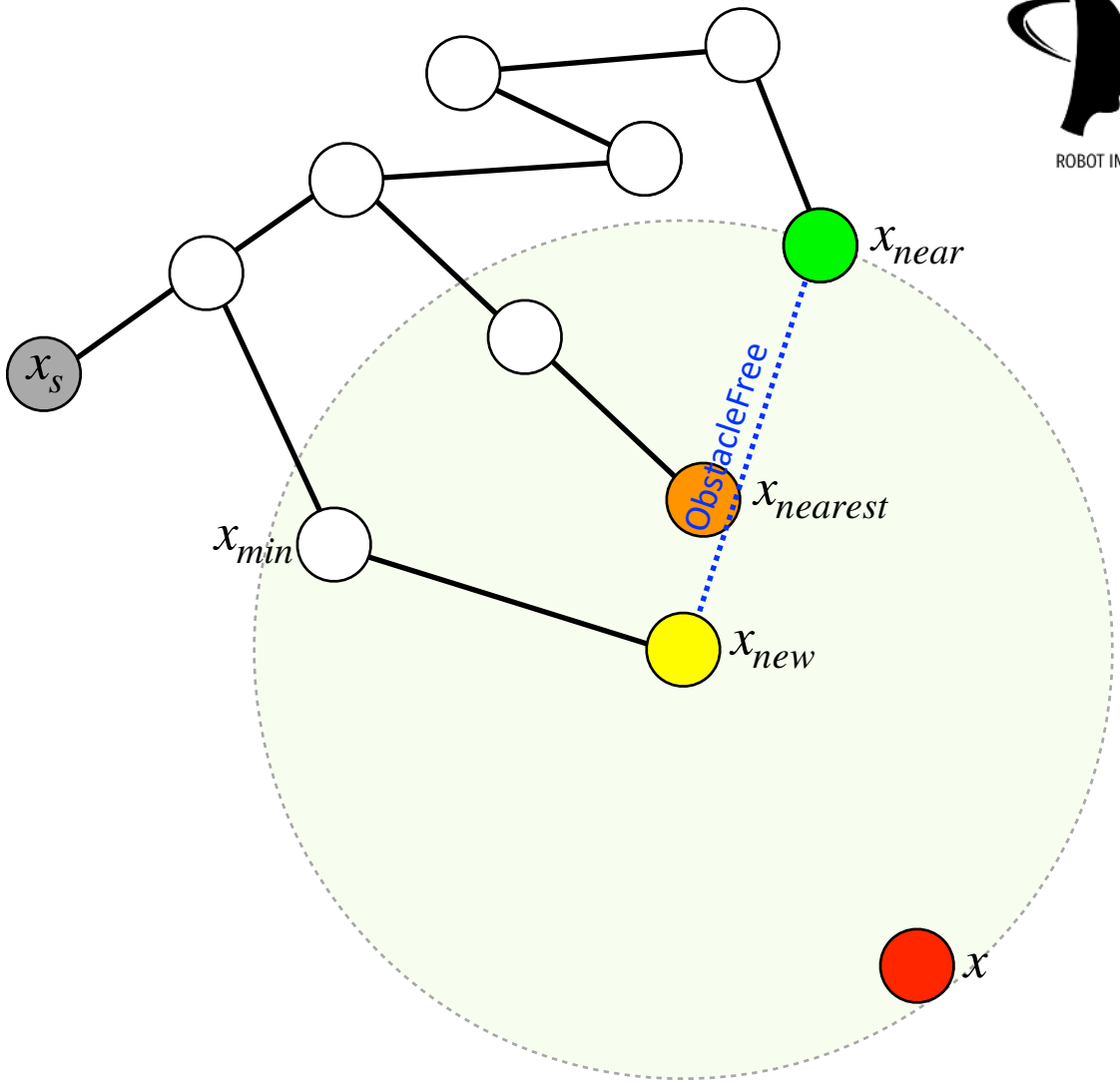


For all x_{near} excluding x_{min}

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) >
           Cost(x_new) + c(Line(x_new, x_near)) then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```

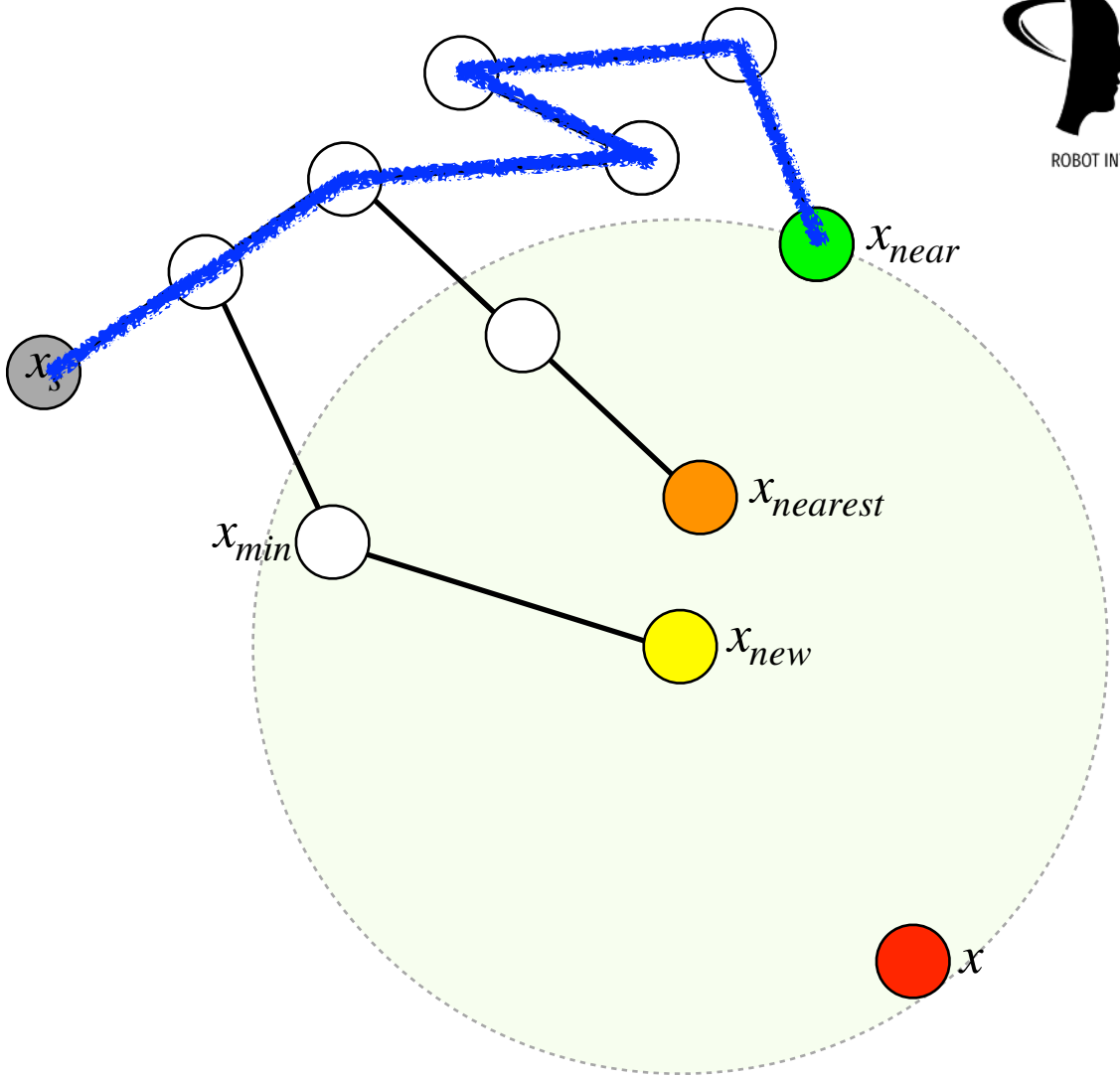


If one can connect x_{near} and x_{new}

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) >$ $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

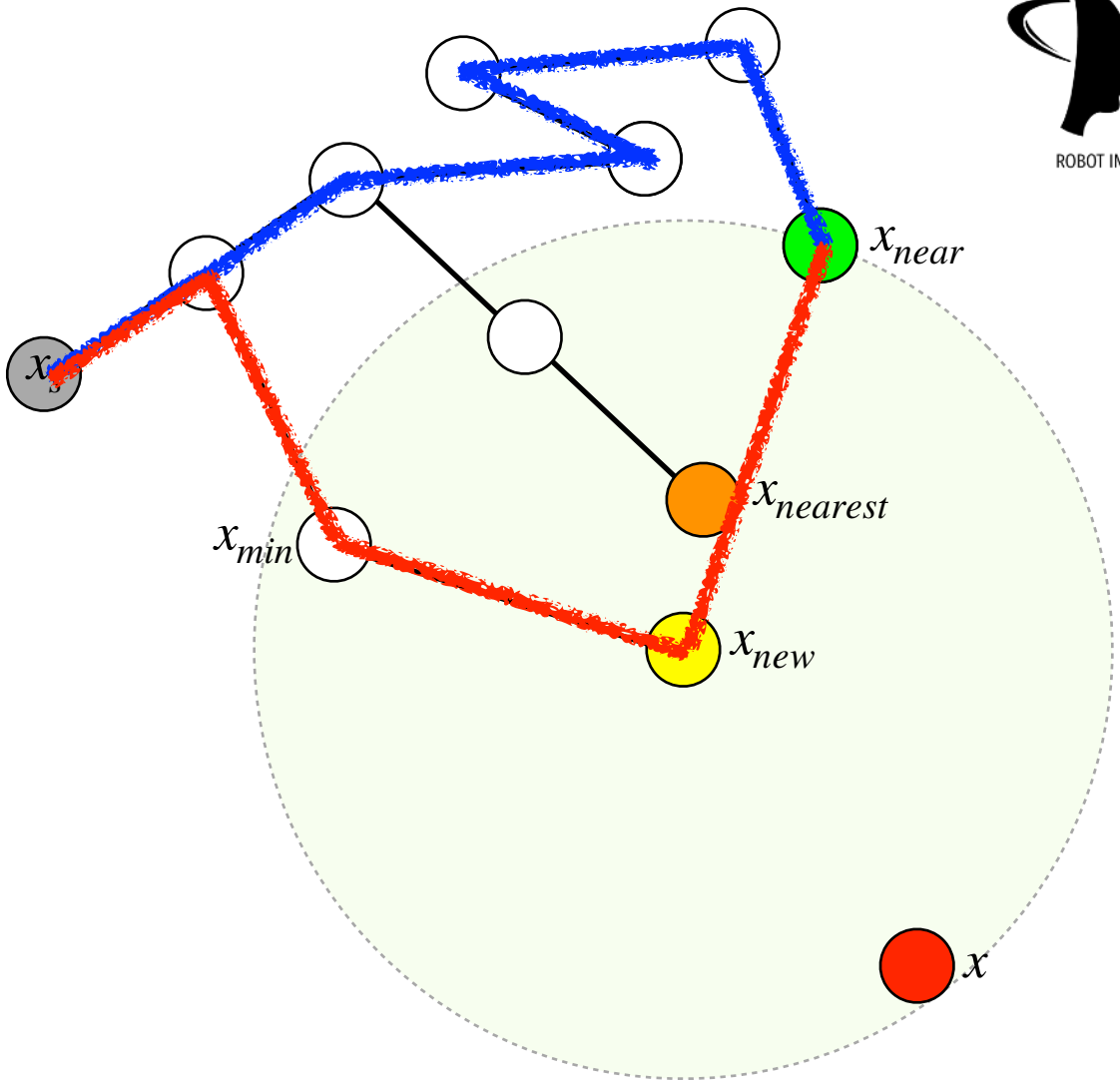


Compute the original cost of x_{near}.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

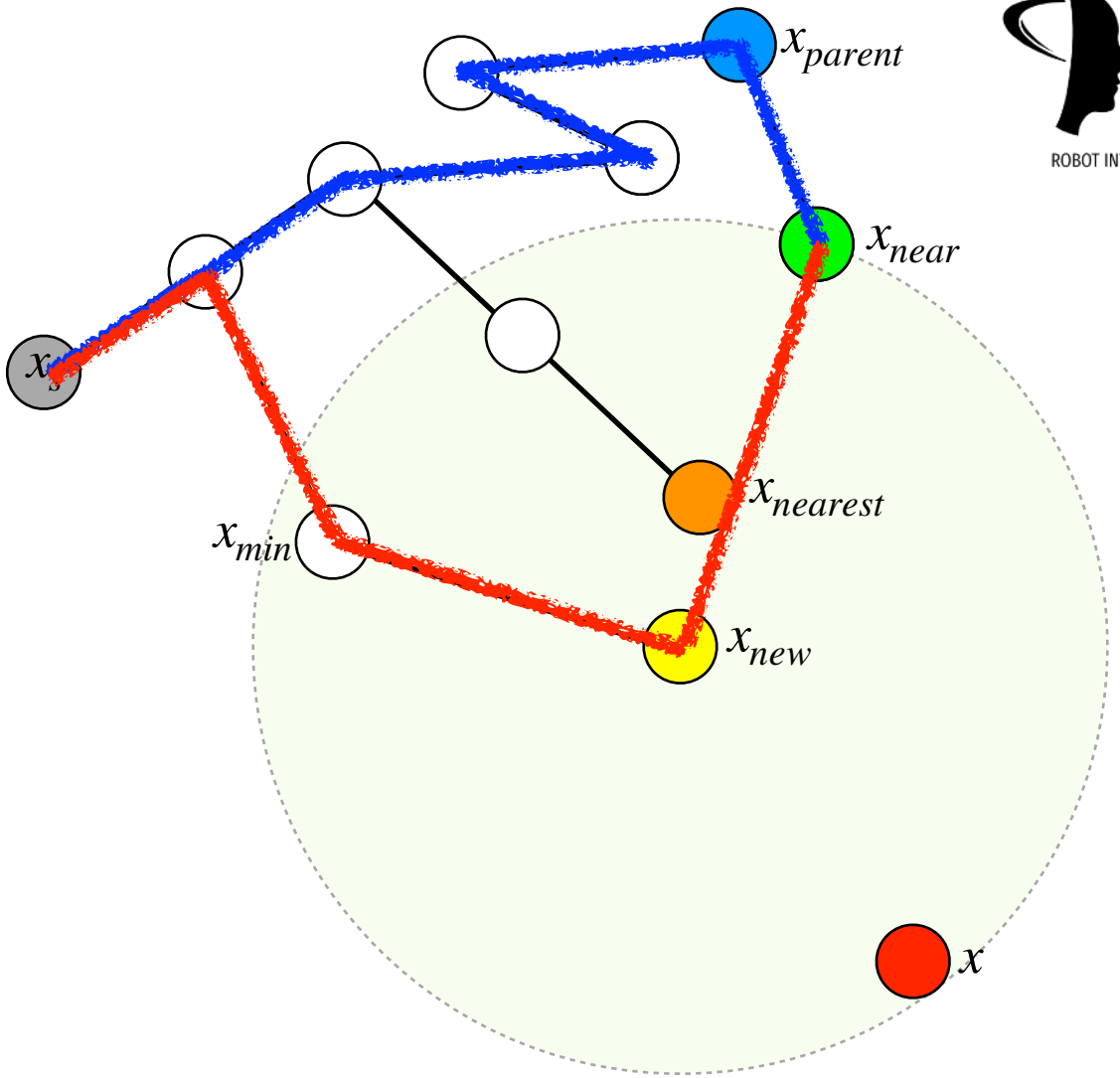


Compute the cost of x_{near} through x_{new}.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

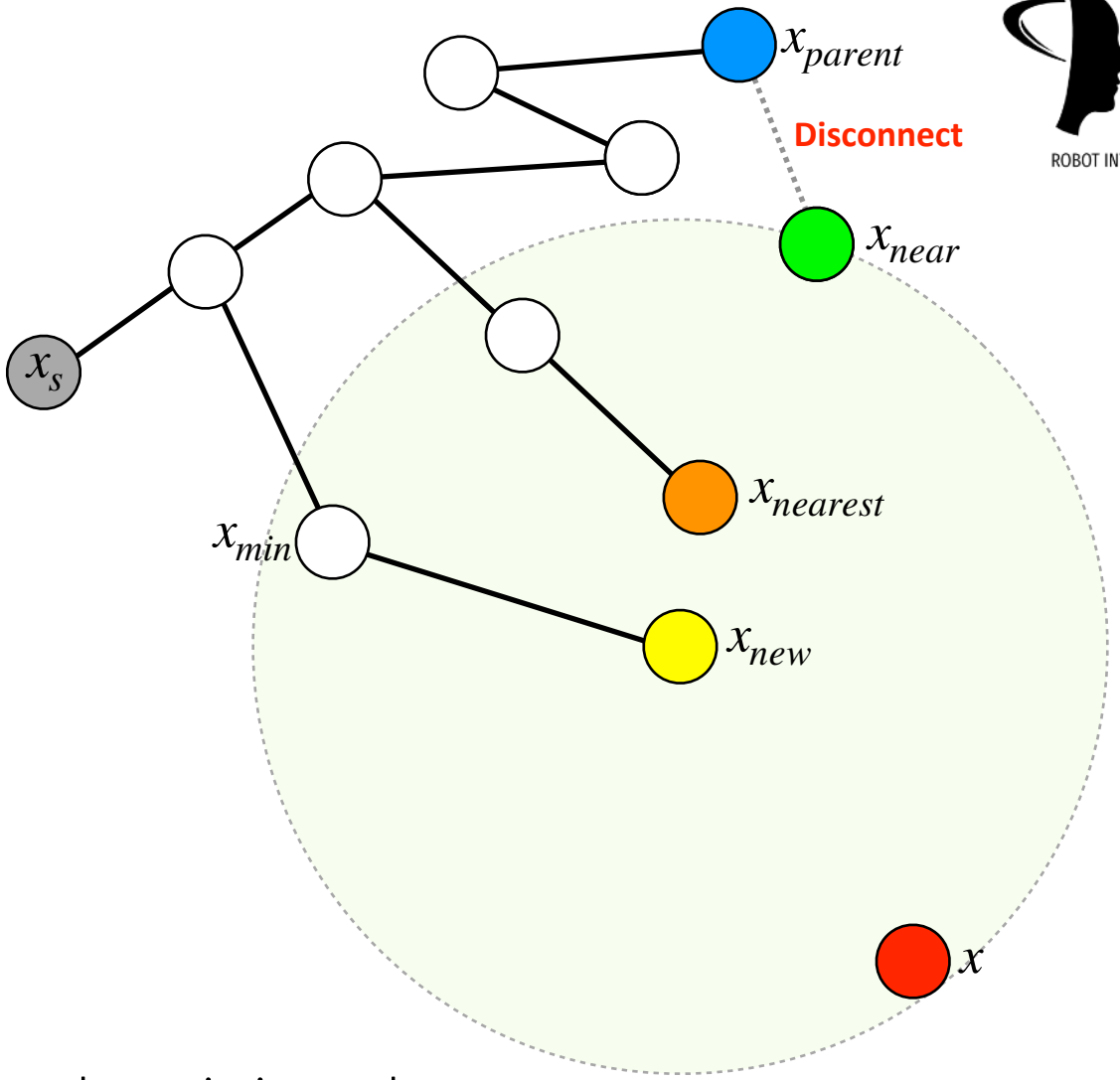


If the cost of x_{near} through x_{new} is smaller, then **rewire**.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

1 $V' \leftarrow V$; $E' \leftarrow E$;
2 $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
3 $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
4 **if** $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
5 $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
6 $x_{\min} \leftarrow x_{\text{nearest}}$;
7 $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
8 **for** *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
9 **if** $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
10 $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
11 **if** $c' < \texttt{Cost}(x_{\text{new}})$ **then**
12 $x_{\min} \leftarrow x_{\text{near}}$;
13 $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
14 **for** *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
15 **if** $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
16 $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
17 $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
  $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
18 **return** $G' = (V', E')$

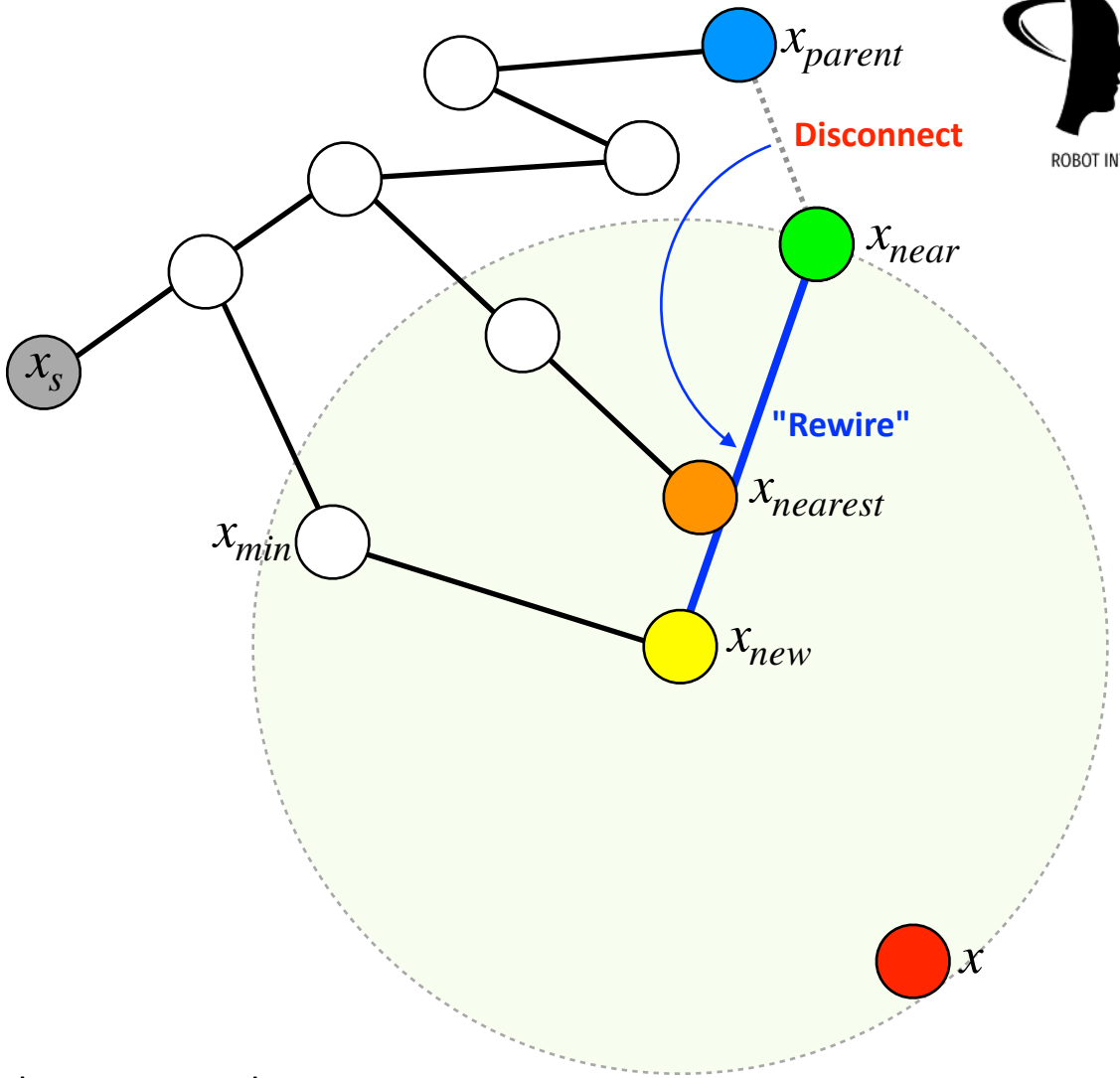


Disconnect the existing edge.

RRT*

Algorithm 4: $\texttt{Extend}_{RRT^*}$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) >
           Cost(x_new) + c(Line(x_new, x_near)) then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```



Connect the new edge.

RRT*

ROBOT INTELLIGENCE LAB

Algorithm 4: $\texttt{Extend}_{RRT^*}$

1 $V' \leftarrow V$; $E' \leftarrow E$;
2 $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
3 $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
4 **if** $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
5 $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
6 $x_{\min} \leftarrow x_{\text{nearest}}$;
7 $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
8 **for** *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
9 **if** $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
10 $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
11 **if** $c' < \texttt{Cost}(x_{\text{new}})$ **then**
12 $x_{\min} \leftarrow x_{\text{near}}$;
13 $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
14 **for** *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
15 **if** $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) >$ $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
16 $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
17 $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
 $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
18 **return** $G' = (V', E')$

Same as RRT

Cost-aware routing

Rewiring

ROBOT INTELLIGENCE LAB

Gaussian Processes in Navigation

"Real-Time Navigation in Crowded Dynamic Environments Using Gaussian Process Motion Control," 2014

"Gaussian Random Paths," 2016

GP Navigation

ROBOT INTELLIGENCE LAB

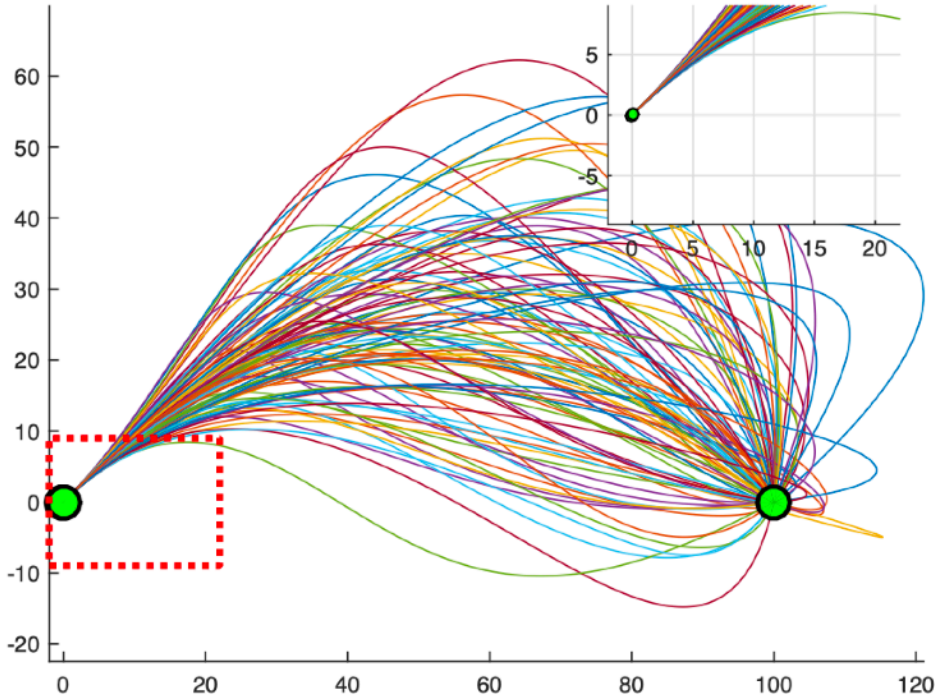
- "Real-Time Navigation in Crowded Dynamic Environments Using Gaussian Process Motion Control," 2014
  - It first predicts the future pedestrian position using Gaussian processes.
  - Then, Gaussian process regression is used to find the control signal from consecutive pedestrian positions.

GP Navigation

Gaussian Random Path

ROBOT INTELLIGENCE LAB

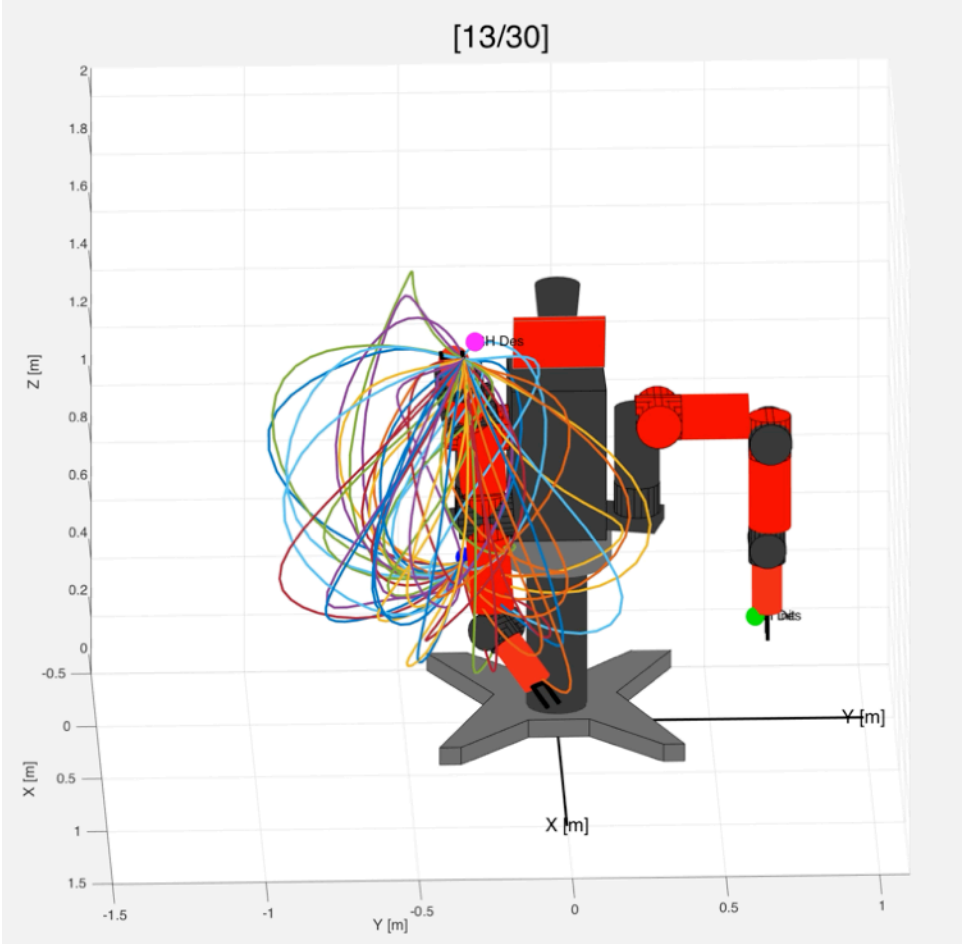
- "Gaussian Random Paths," 2016
  - GRP defines a distribution over a smooth trajectories and one can easily sample the trajectories.



Gaussian Random Path

ROBOT INTELLIGENCE LAB

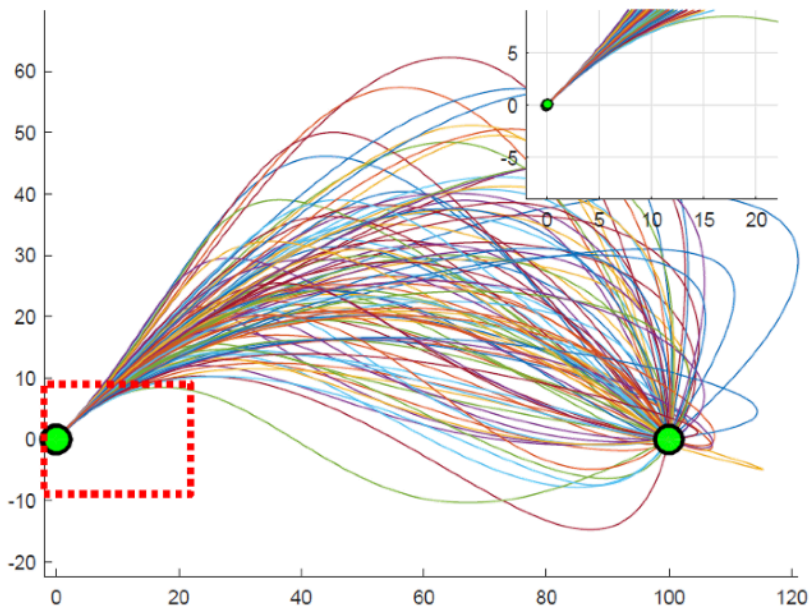
- How can we sample diverse paths interpolating given anchoring points?



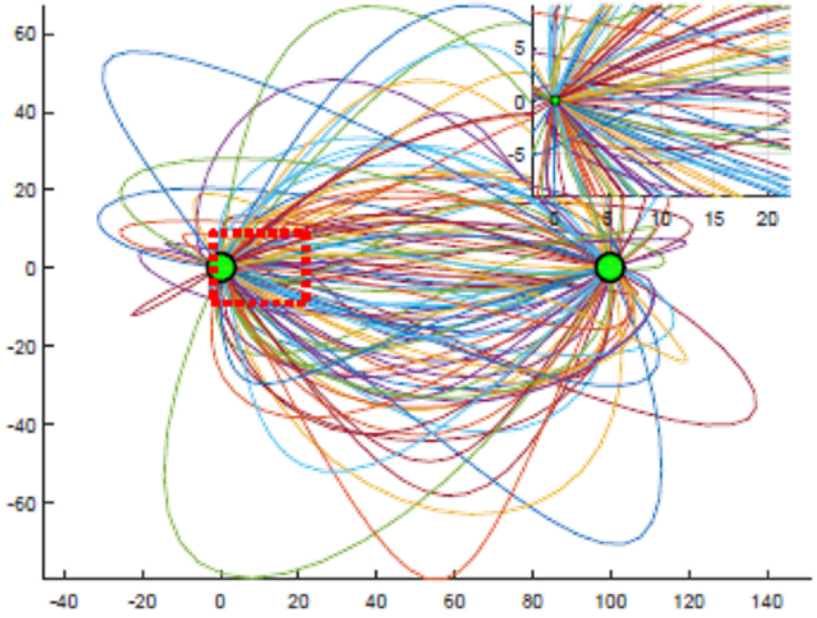
Gaussian Random Path

ROBOT INTELLIGENCE LAB

Sampled paths with ϵ run-up

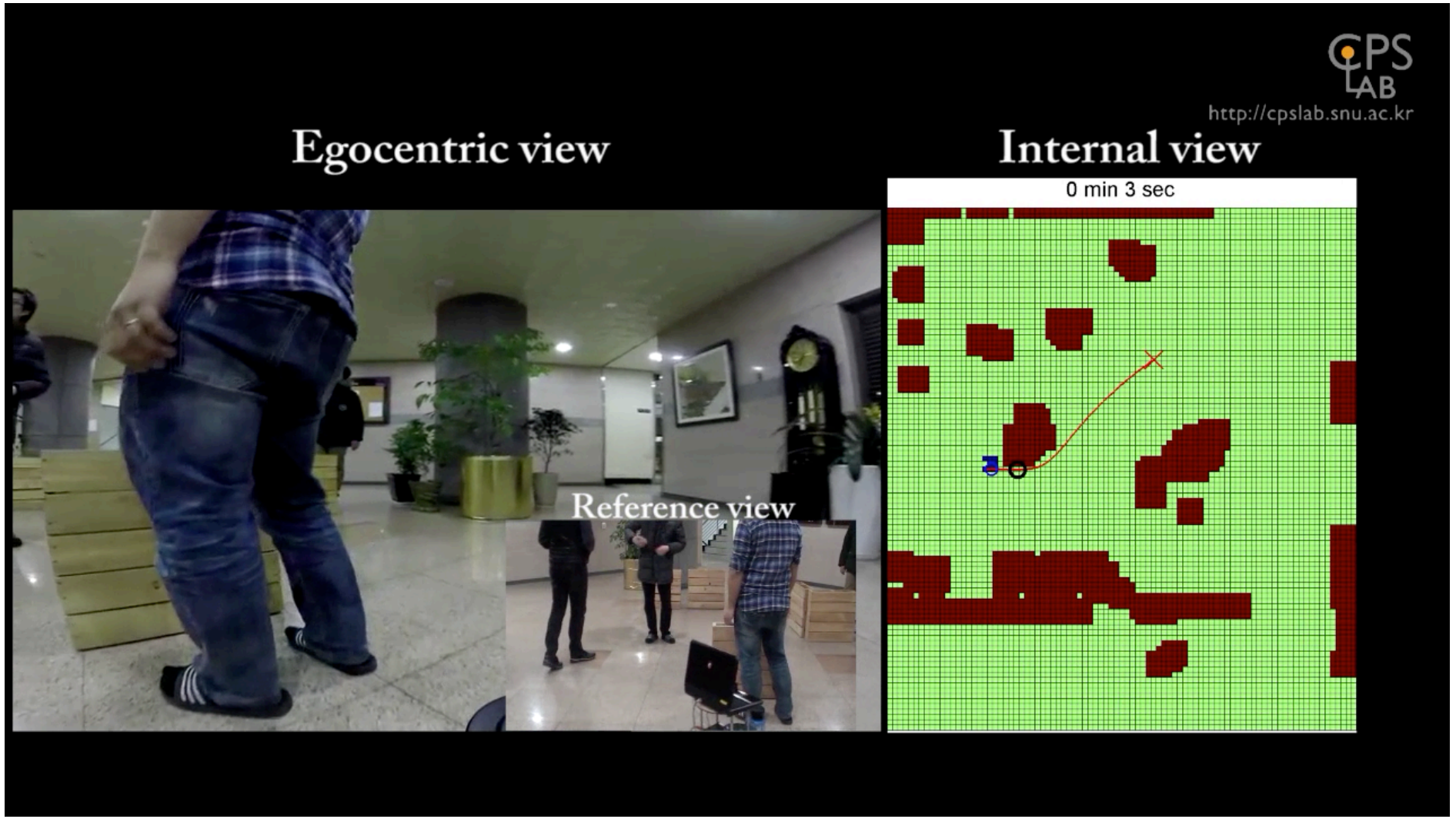


Sampled paths without ϵ run-up



Gaussian Random Path

ROBOT INTELLIGENCE LAB



Gaussian Random Path

ROBOT INTELLIGENCE LAB

ROBOT INTELLIGENCE LAB

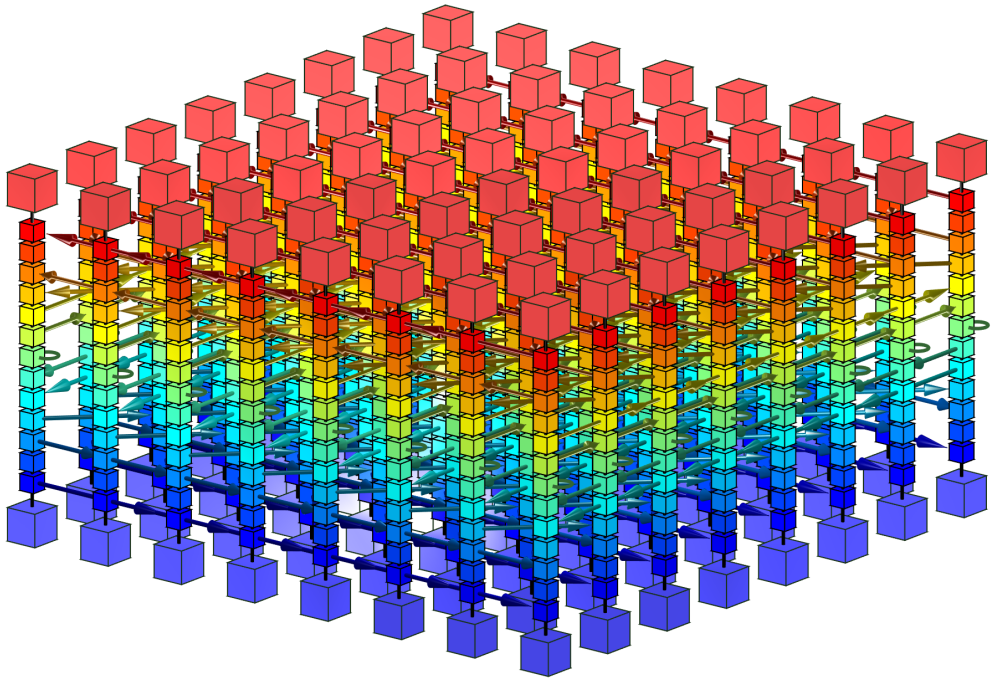
Occupancy Flow

"Robust Modeling and Prediction in Dynamic Environments Using Recurrent Flow Networks," 2016

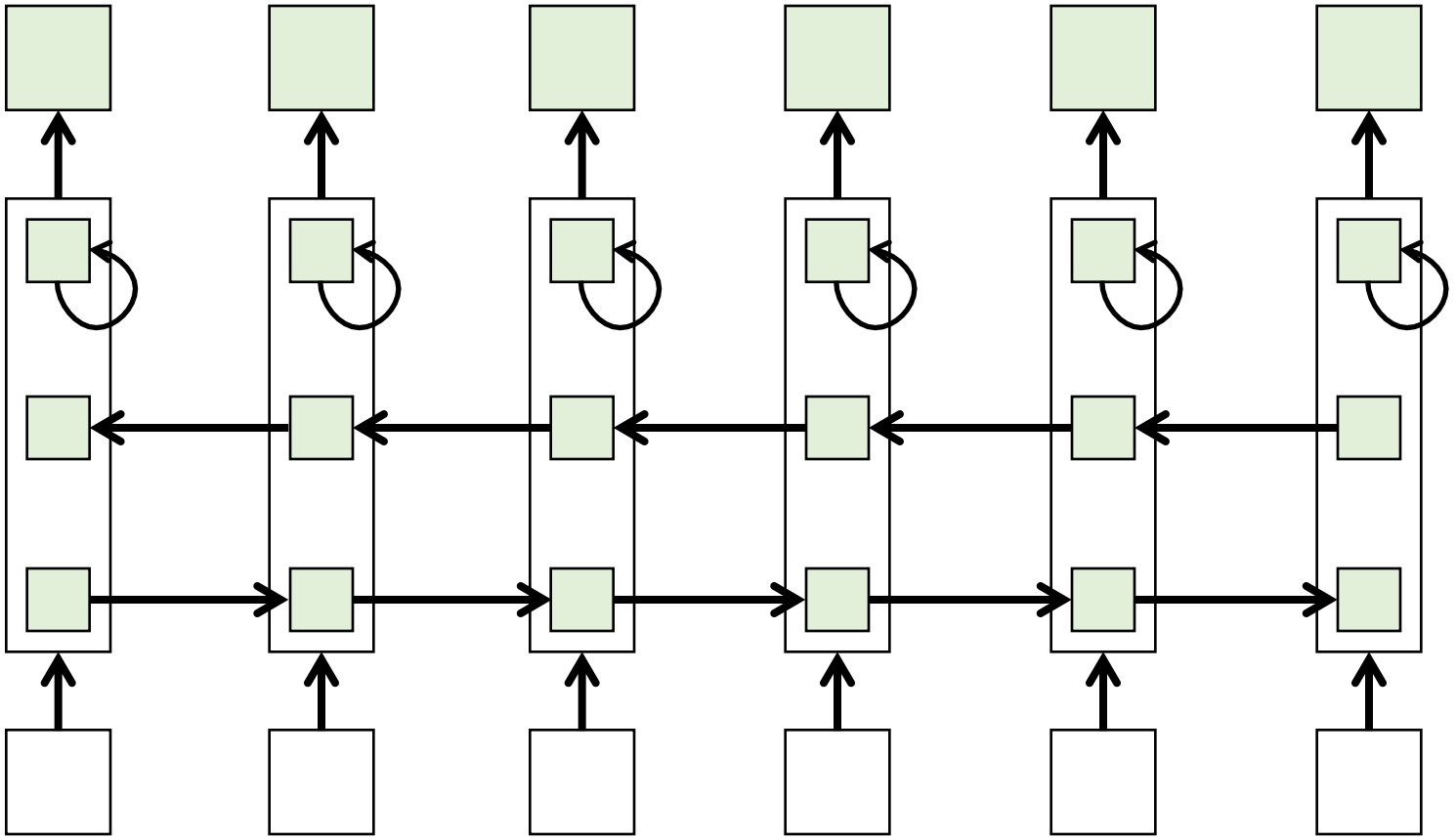
Occupancy Flow

ROBOT INTELLIGENCE LAB

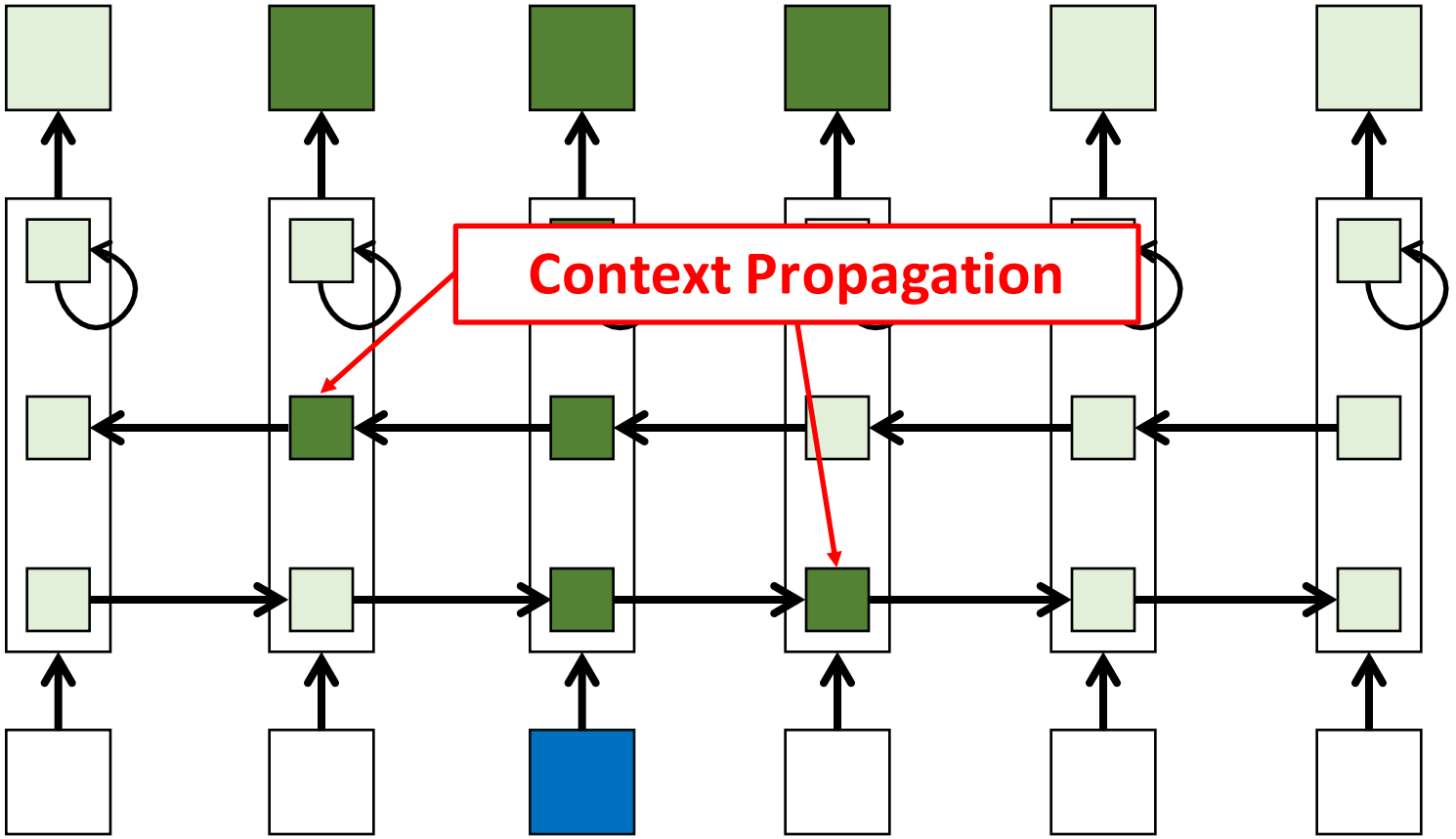
- "Robust Modeling and Prediction in Dynamic Environments Using Recurrent Flow Networks," 2016
  - A recurrent flow network is presented that can robustly estimate the **occupancy flow** without the necessity of inferring the objectiveness.



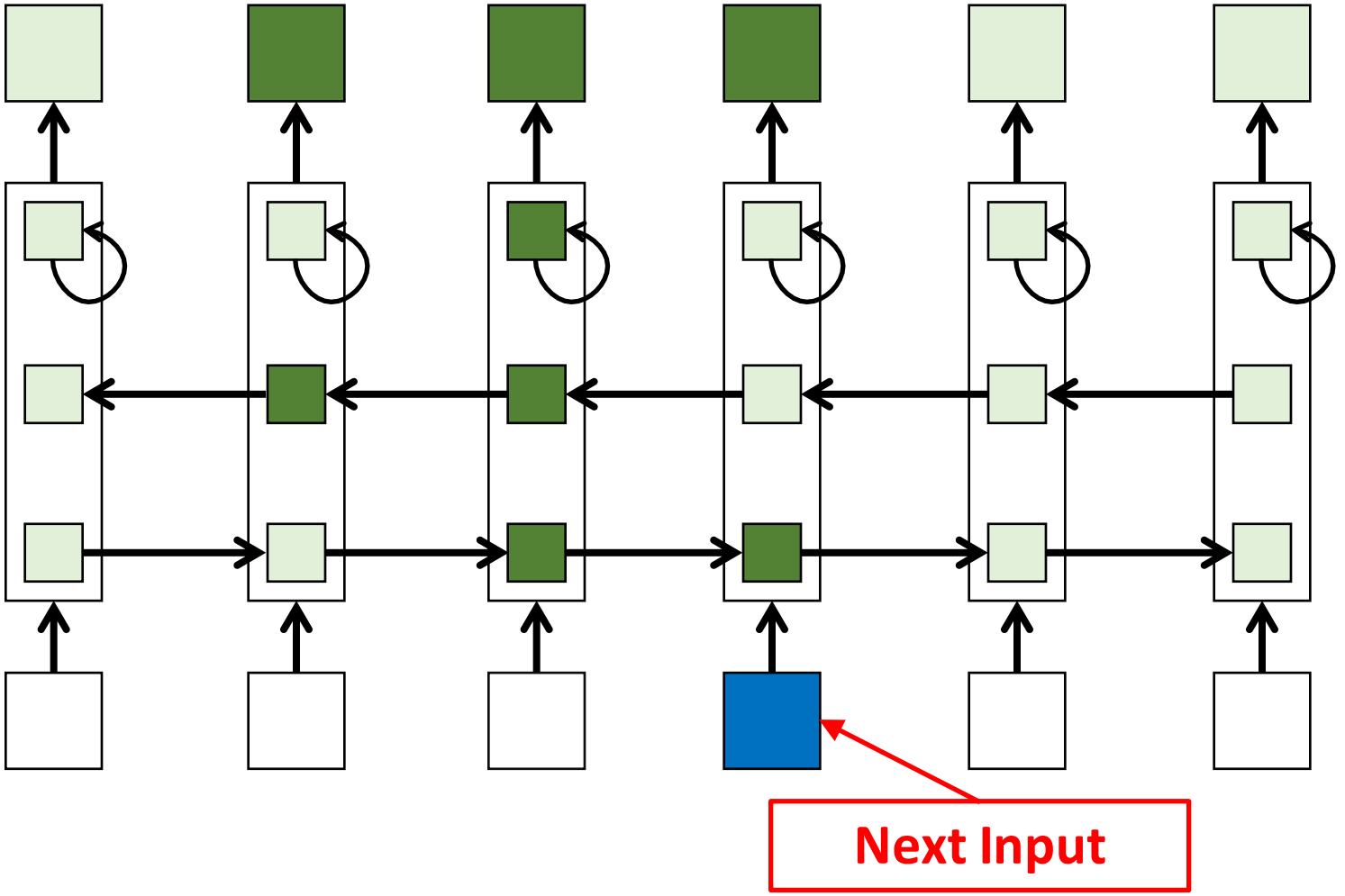
Occupancy Flow



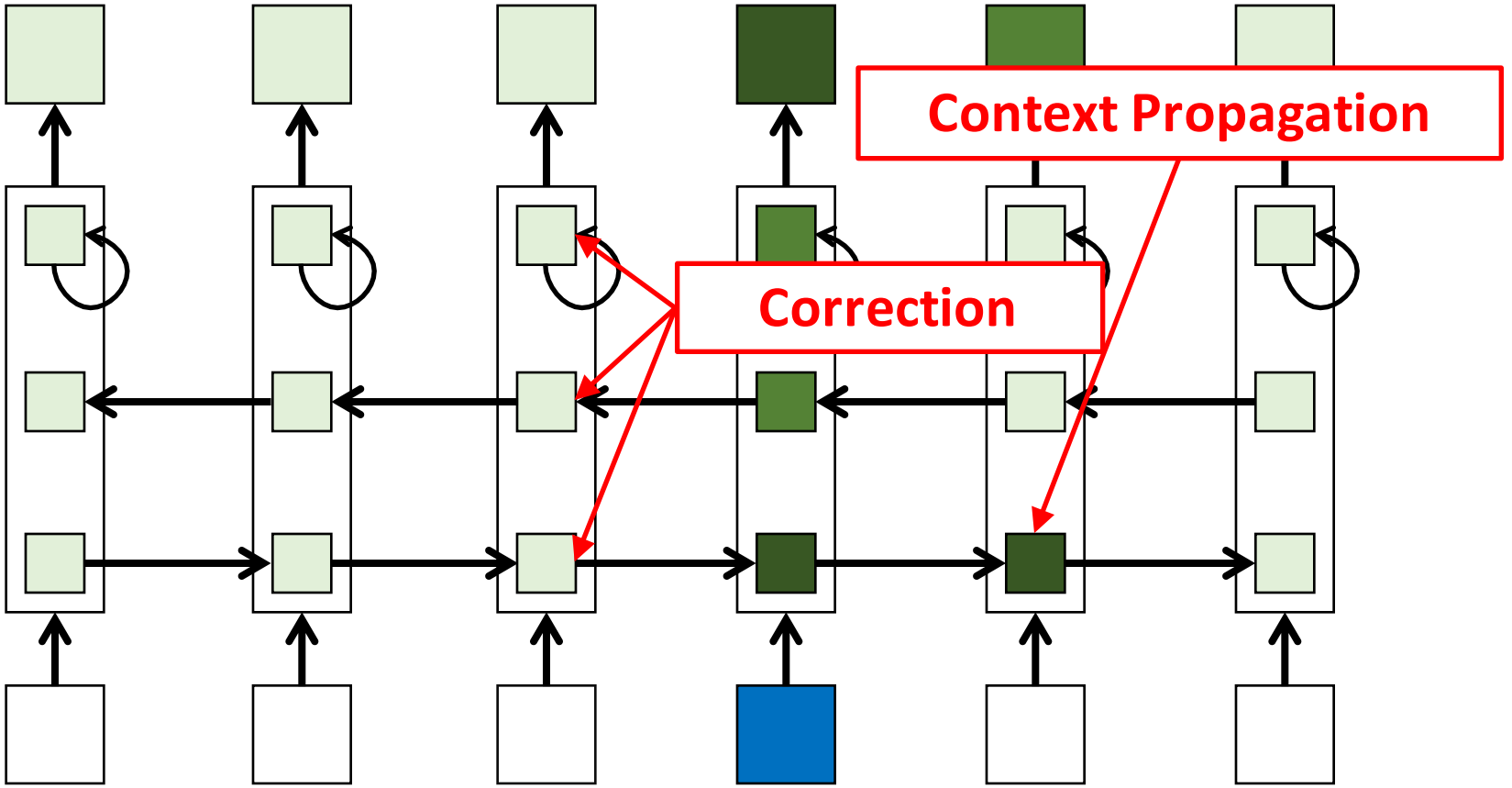
Occupancy Flow



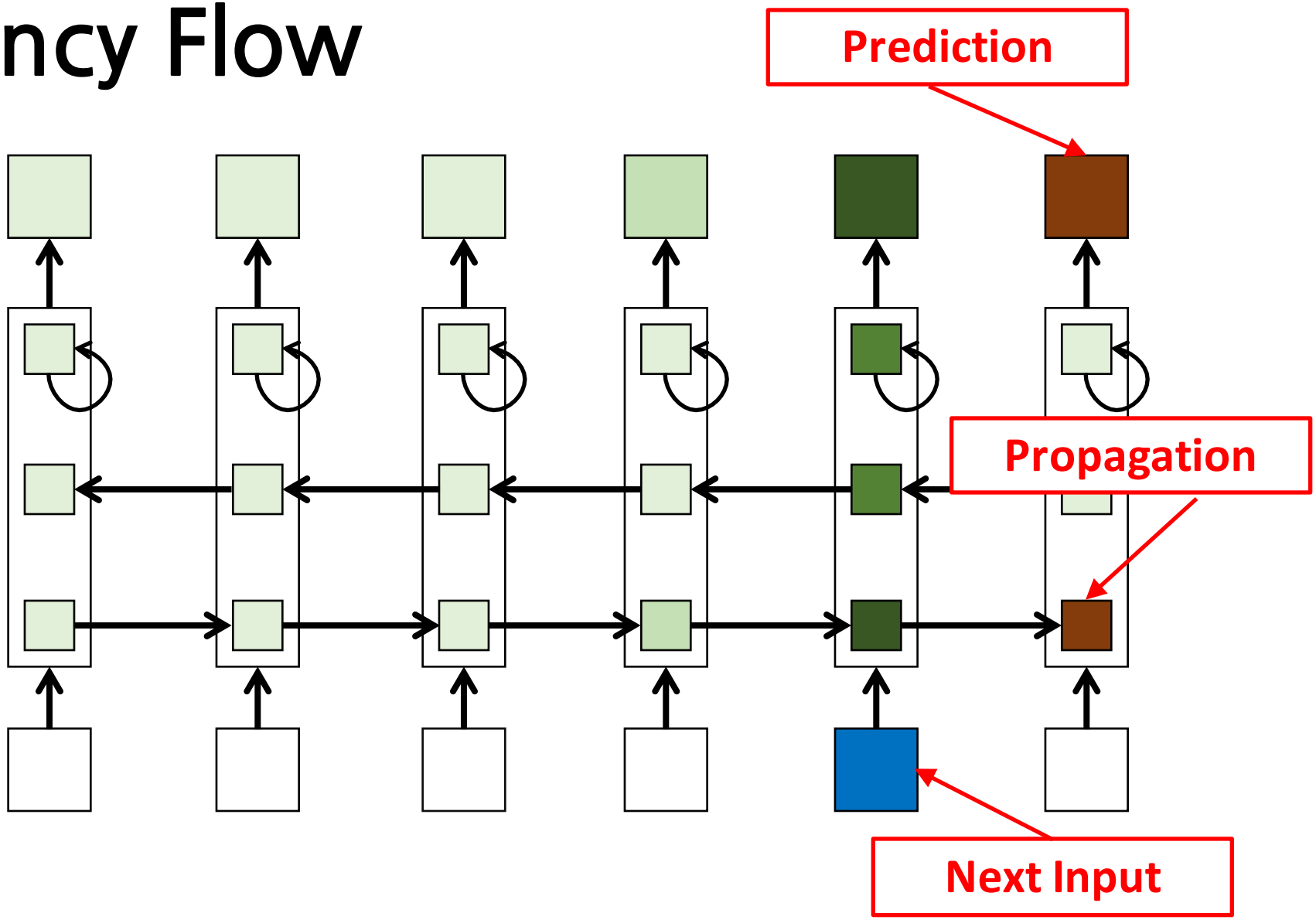
Occupancy Flow



Occupancy Flow

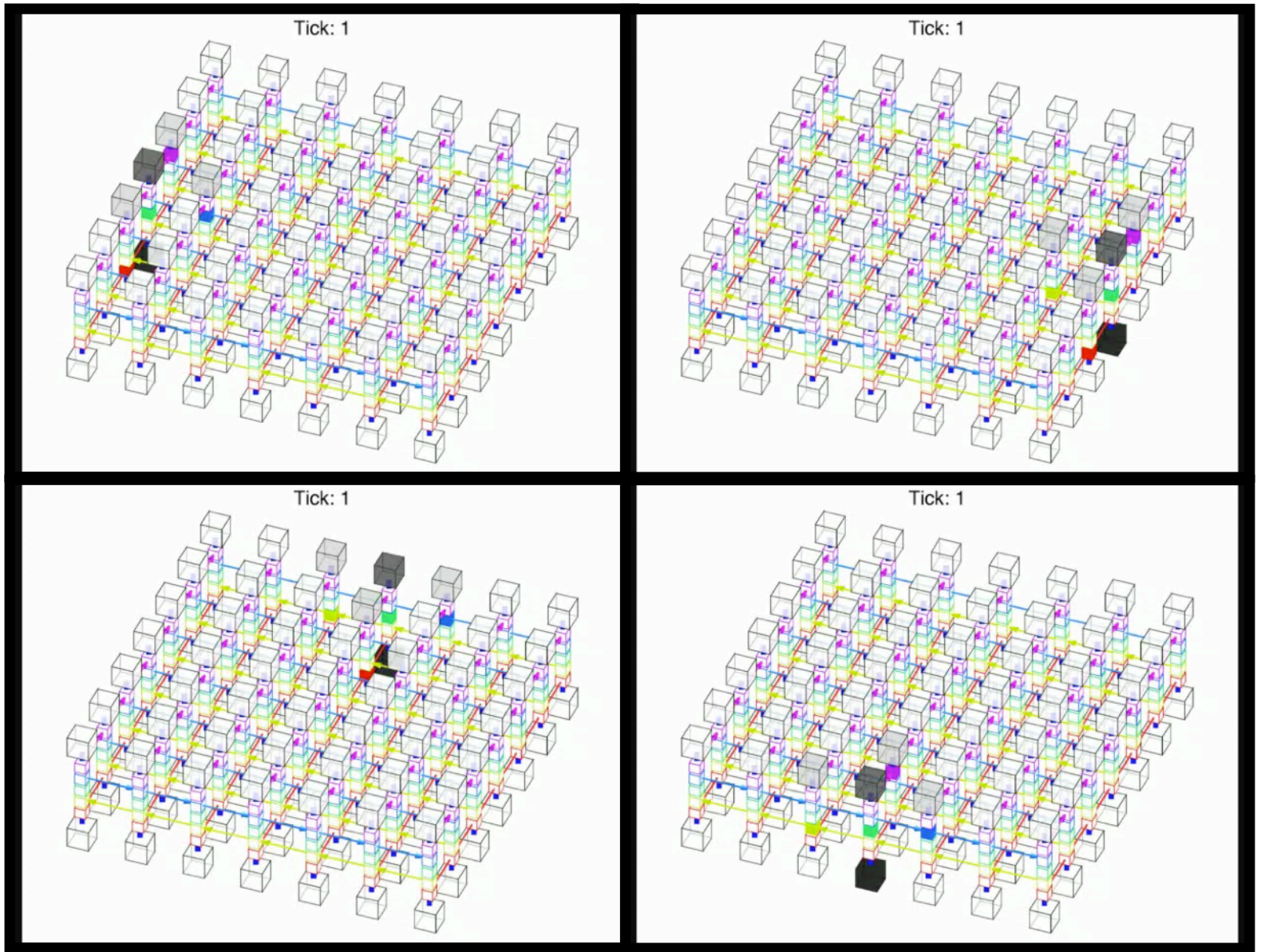


Occupancy Flow



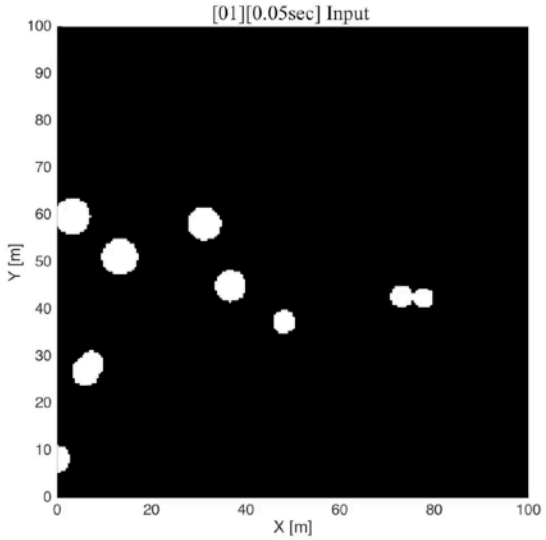
ROBOT INTELLIGENCE LAB

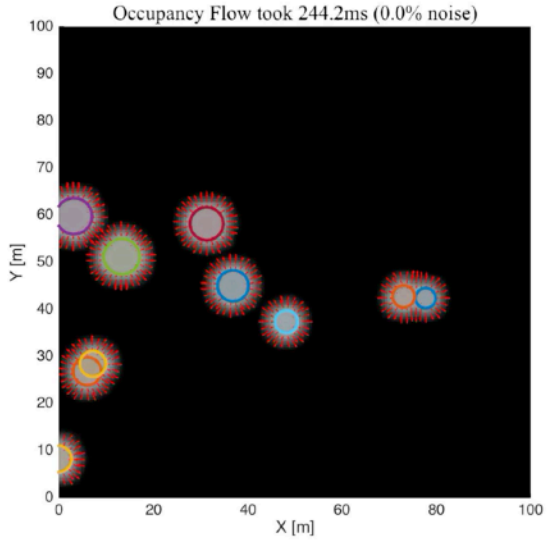
Occupancy Flow

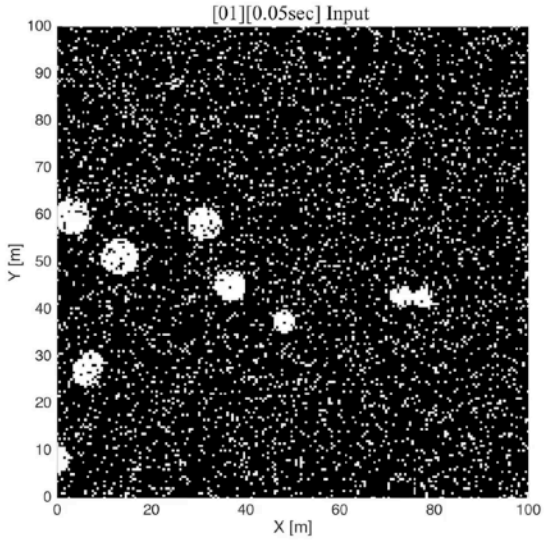


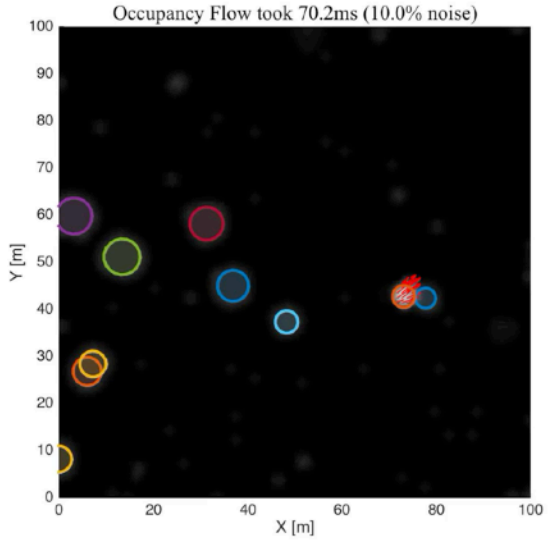
ROBOT INTELLIGENCE LAB

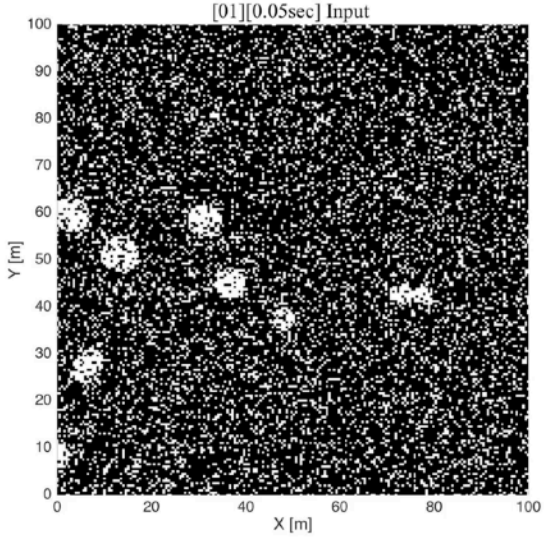
Occupancy Flow

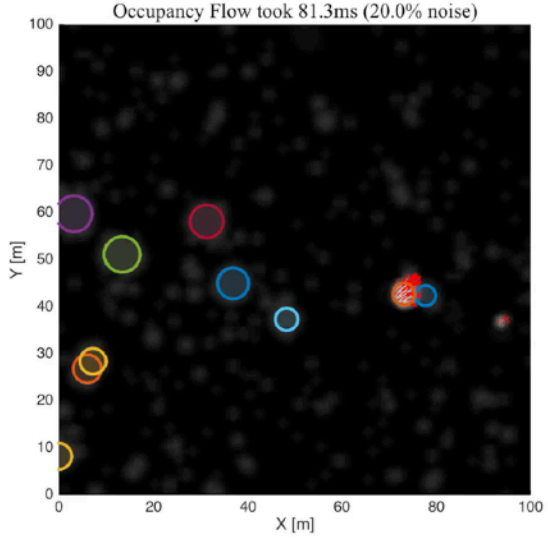


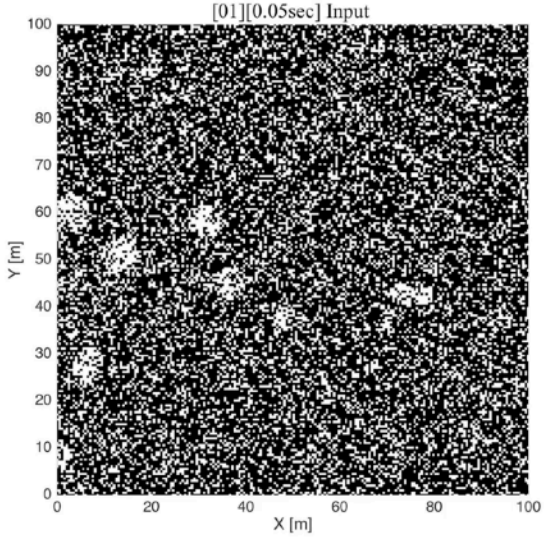


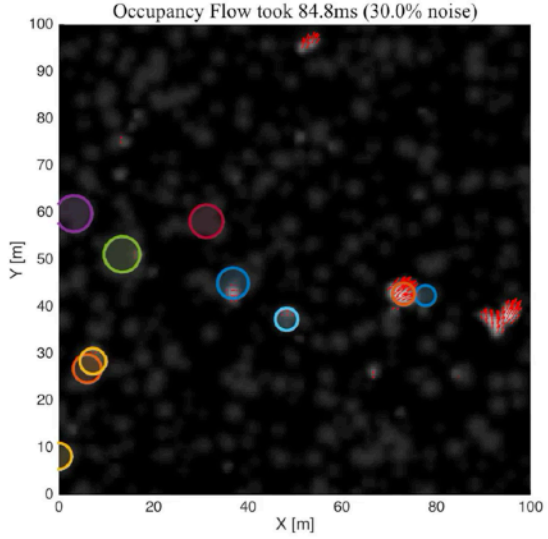




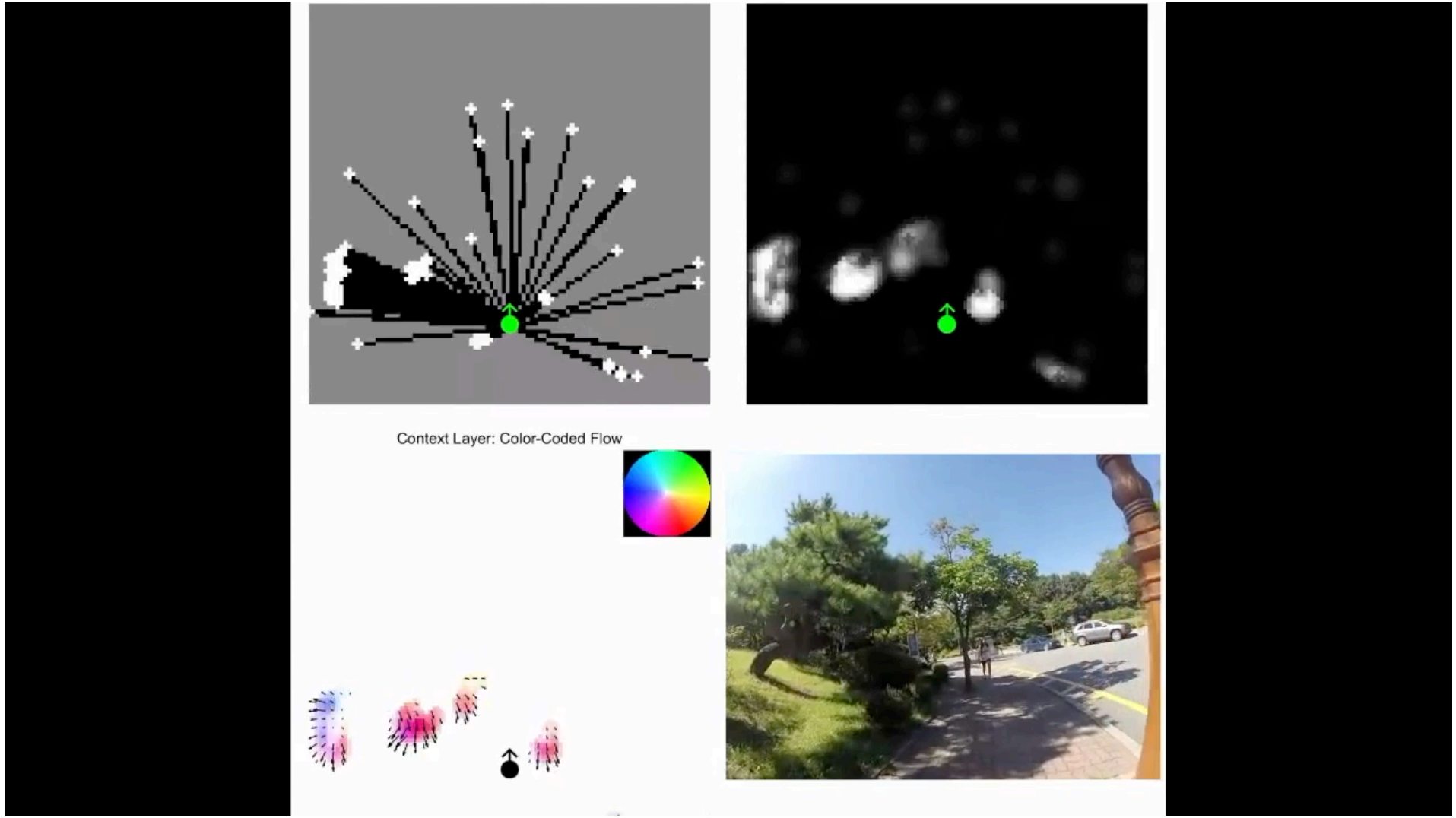






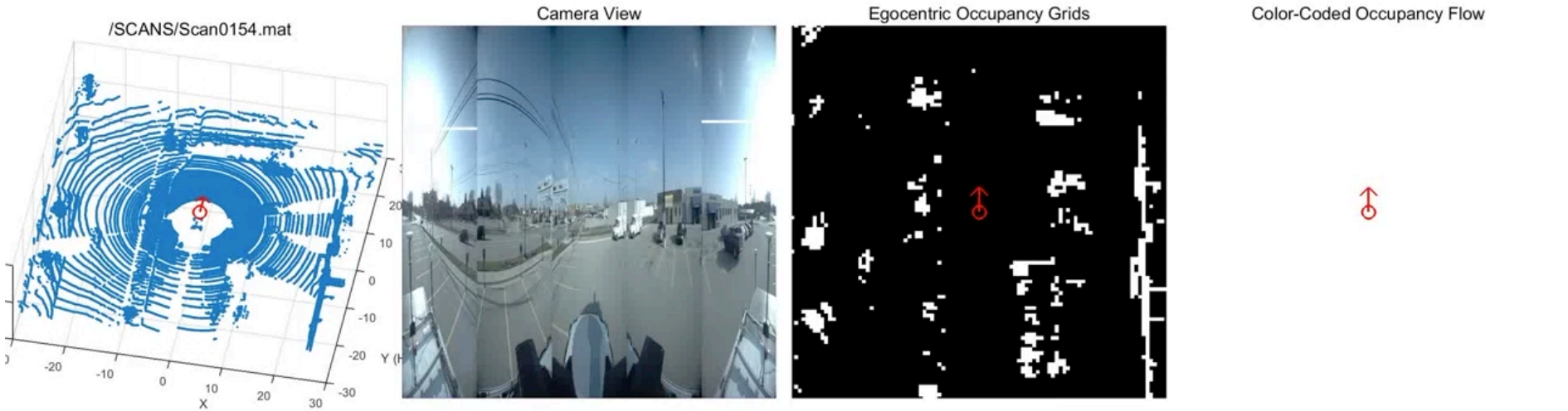


Occupancy Flow



Occupancy Flow

ROBOT INTELLIGENCE LAB



Thank You