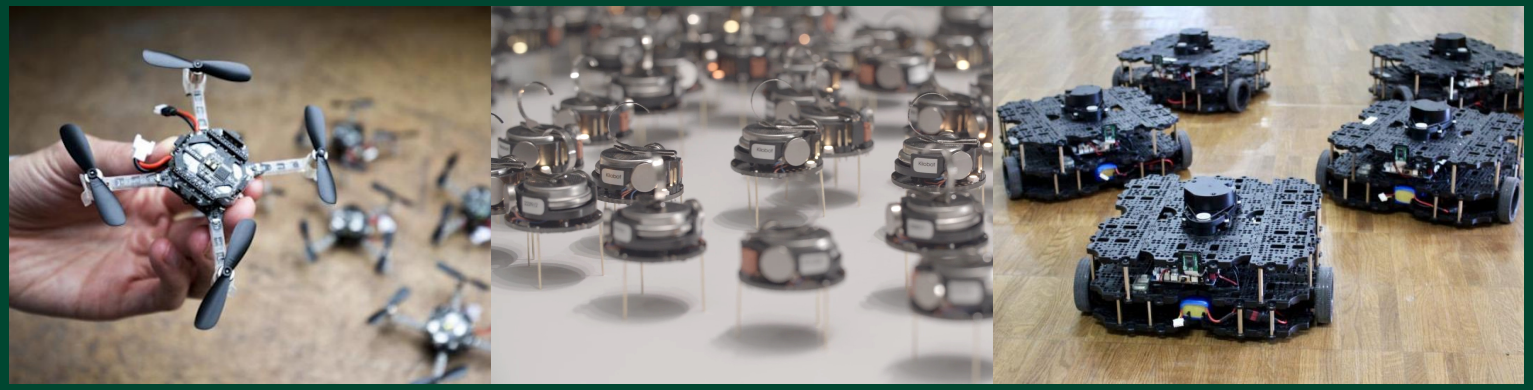
ECE693H, Spring 2025:
Multi-robot System Design

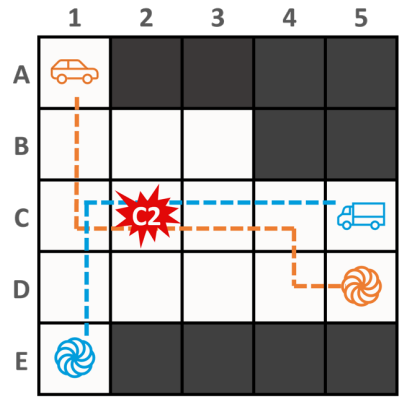
“Multi-robot Systems 2”

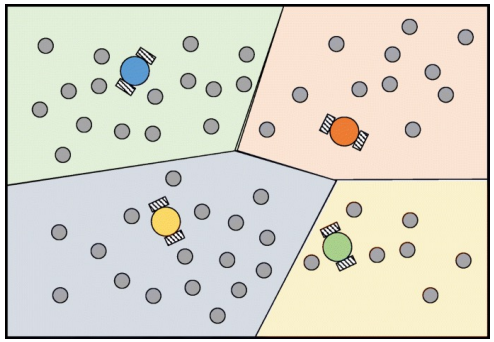


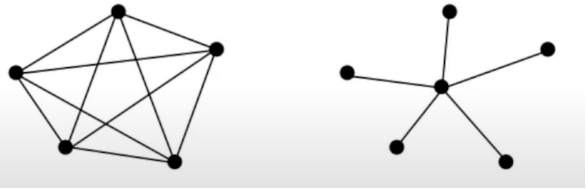
Dr. Daniel Drew

COLLEGE OF ENGINEERING
UNIVERSITY of HAWAI‘I at MĀNOA

MRS Survey Recap







A group (>1) of robots with a common objective

Questions?

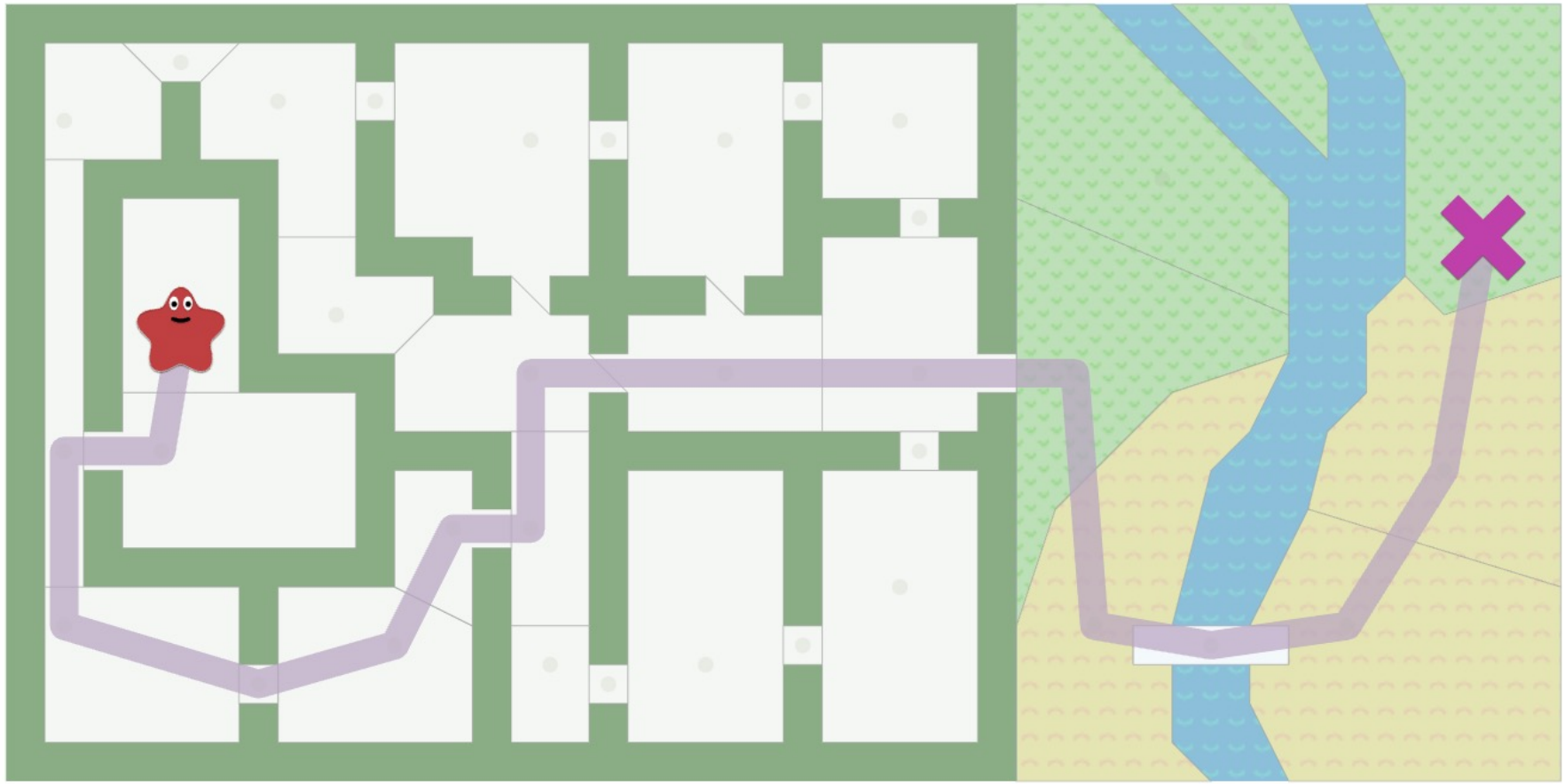
Class Overview

1. Lecture (~20 minutes)
2. Astound the Class (~40 minutes)

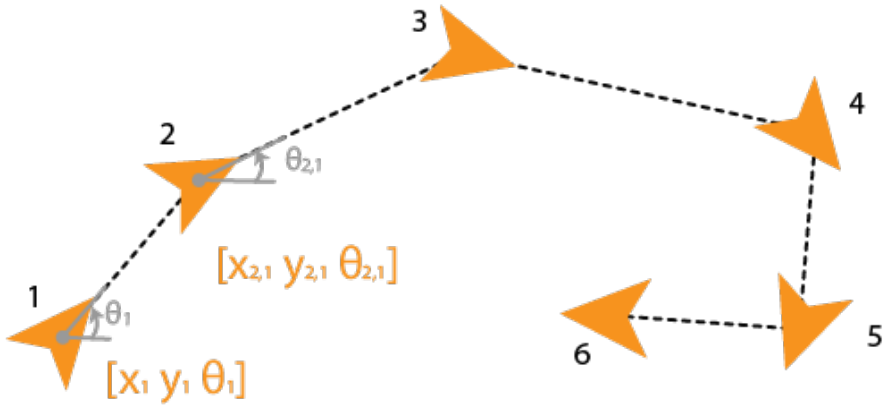
Multi-robot Systems: Lecture 2 of 4

Lecture 1: MRS Survey
Lecture 2: Motion Planning
Lecture 3: Task Assignment (+Motion Planning if needed)
Lecture 3: Communication and Control Paradigms

The Motion Planning Problem



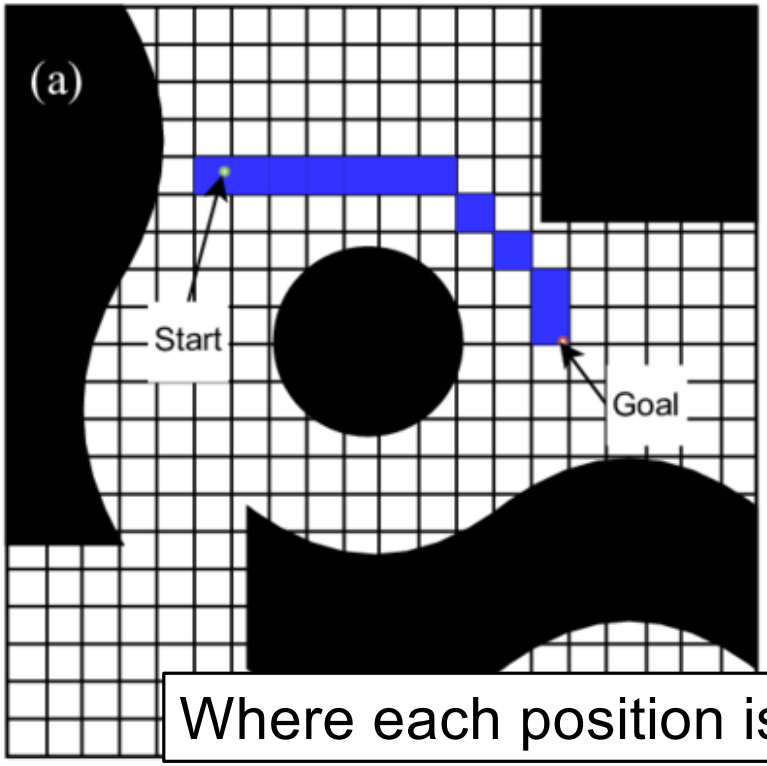
The Motion Planning Problem



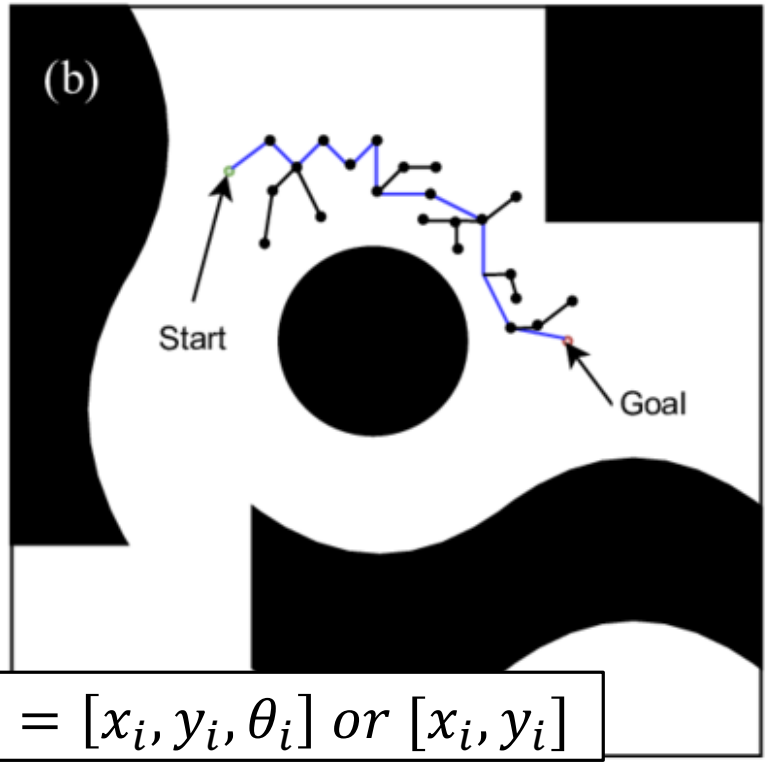
What set of poses best gets the robot from A to B?

Review of Single-Robot Motion Planning

Search-based



Sample-based



Where each position is $p_i = [x_i, y_i, \theta_i]$ or $[x_i, y_i]$

Multi-robot Motion Planning: Taxonomies

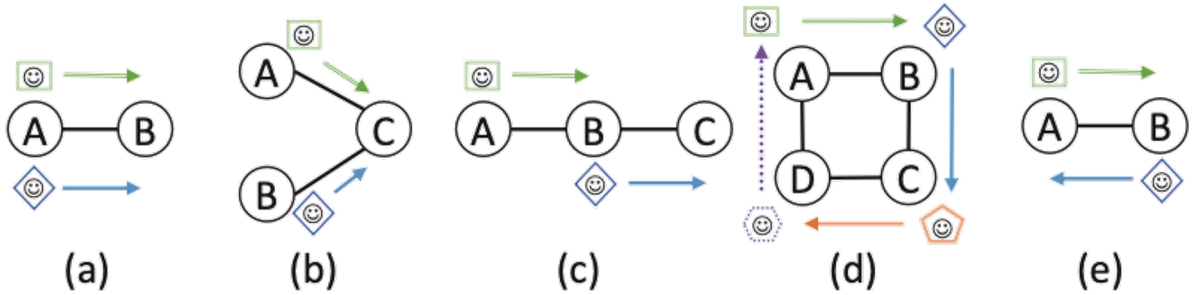
Search-based versus Sample-based

Same dichotomy!

Coupled versus Decoupled

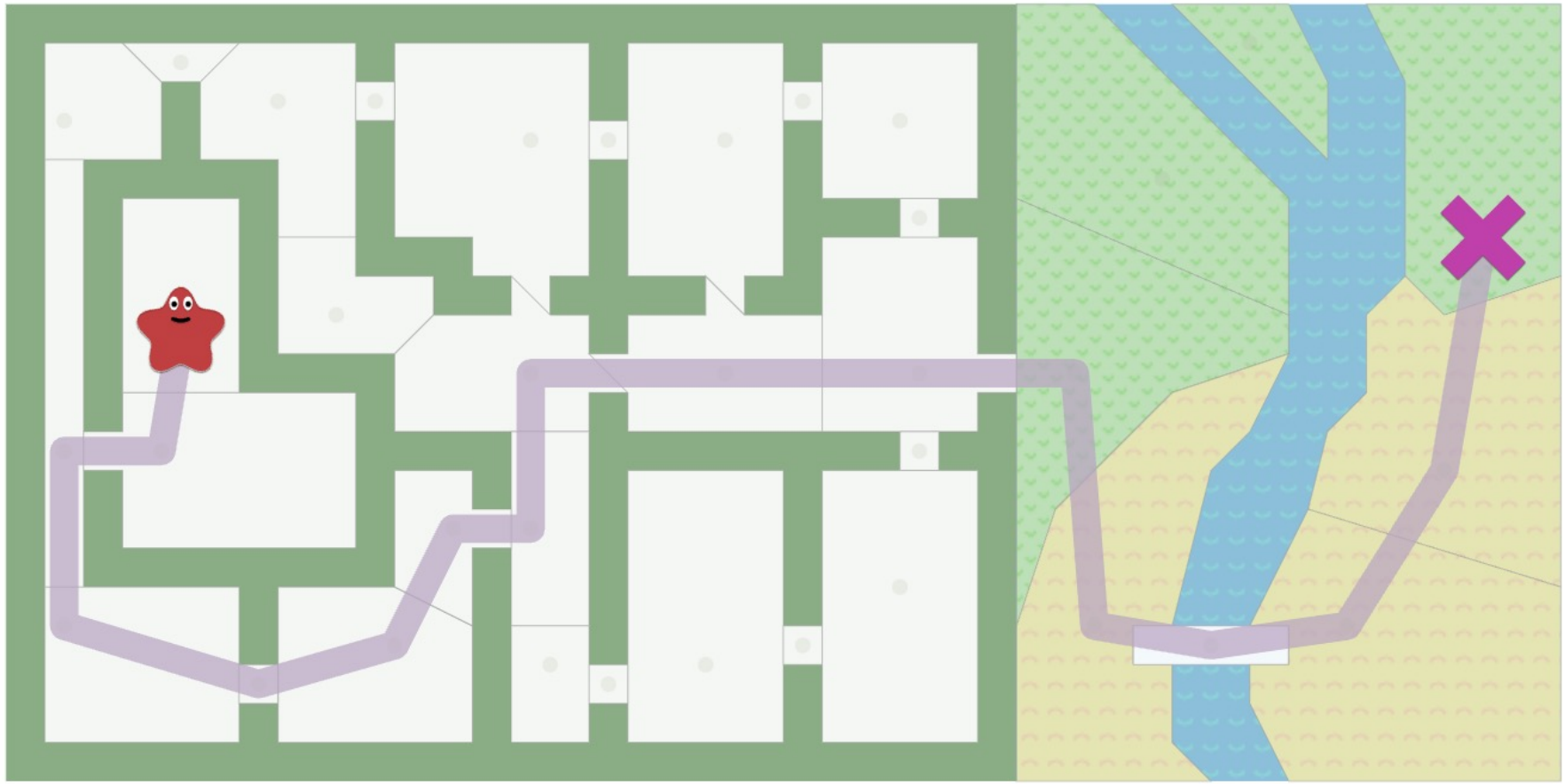
Plan in the full (“global”) joint configuration space versus plan in individual (”local”) configuration spaces

Conflicts

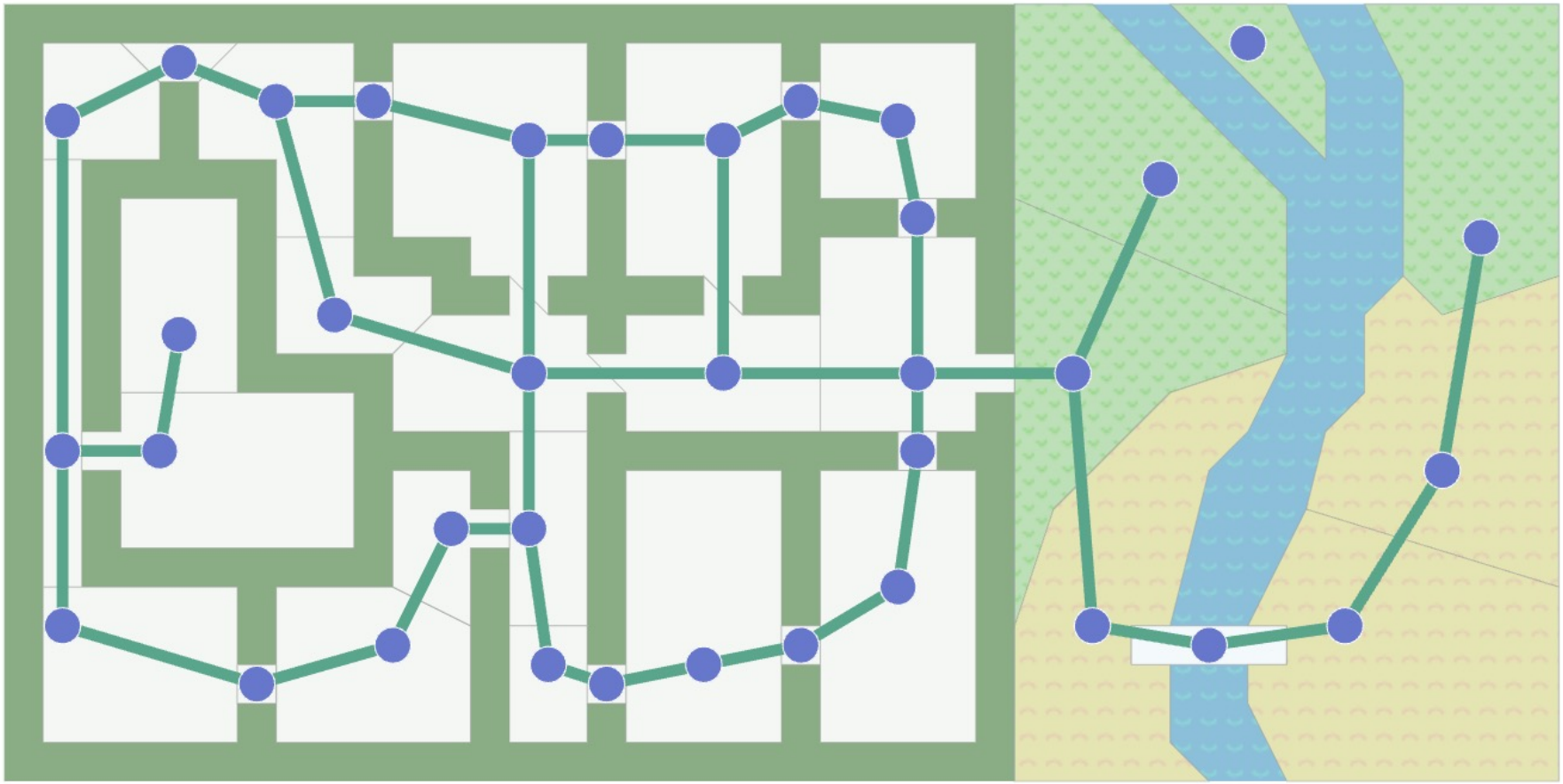


(a) is an edge conflict, (b) a vertex conflict, (c) a following conflict, (d) a cycle conflict, and (e) a swapping conflict.

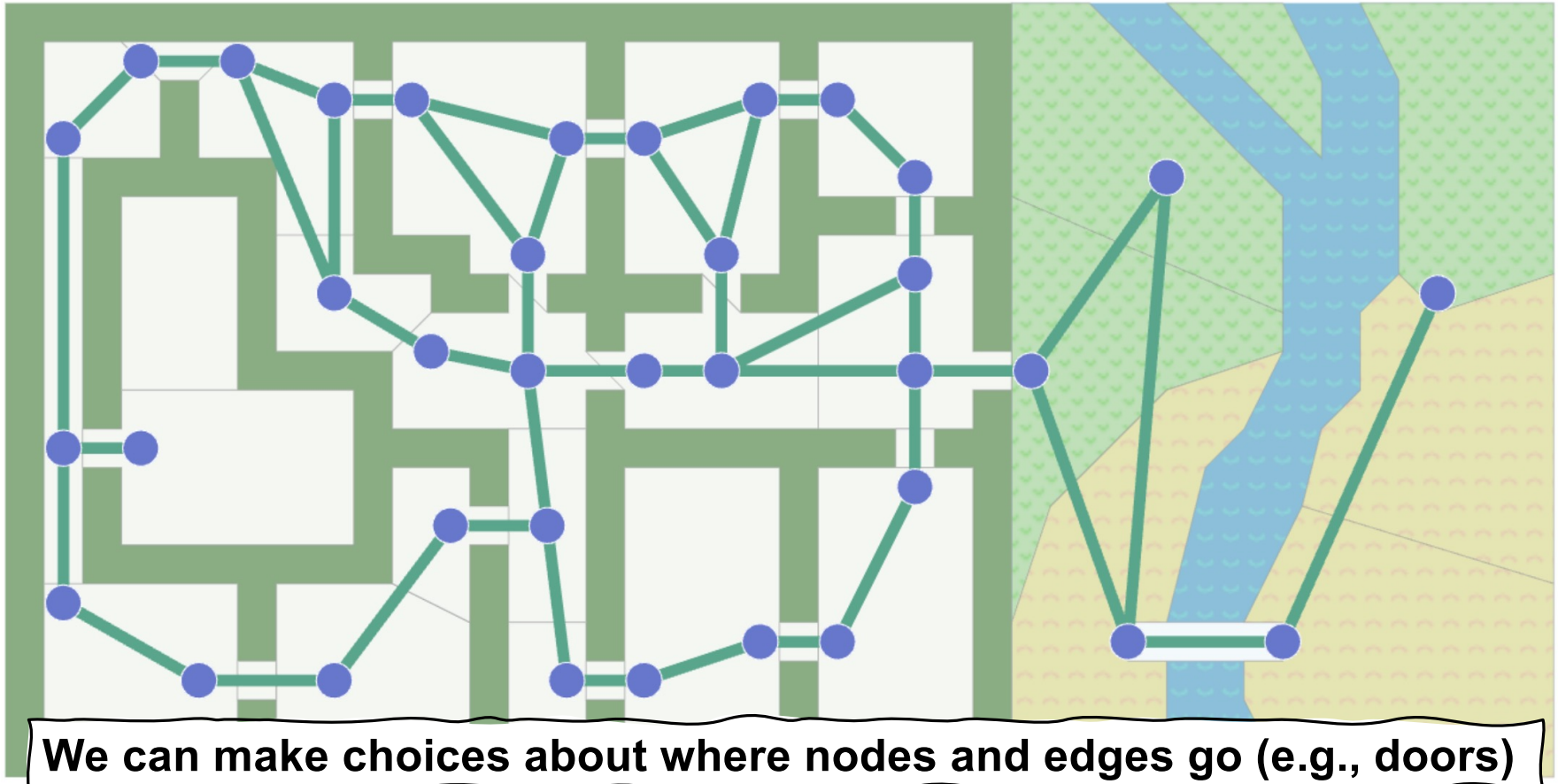
Single Robot Planning: A*



Single Robot Planning: A*

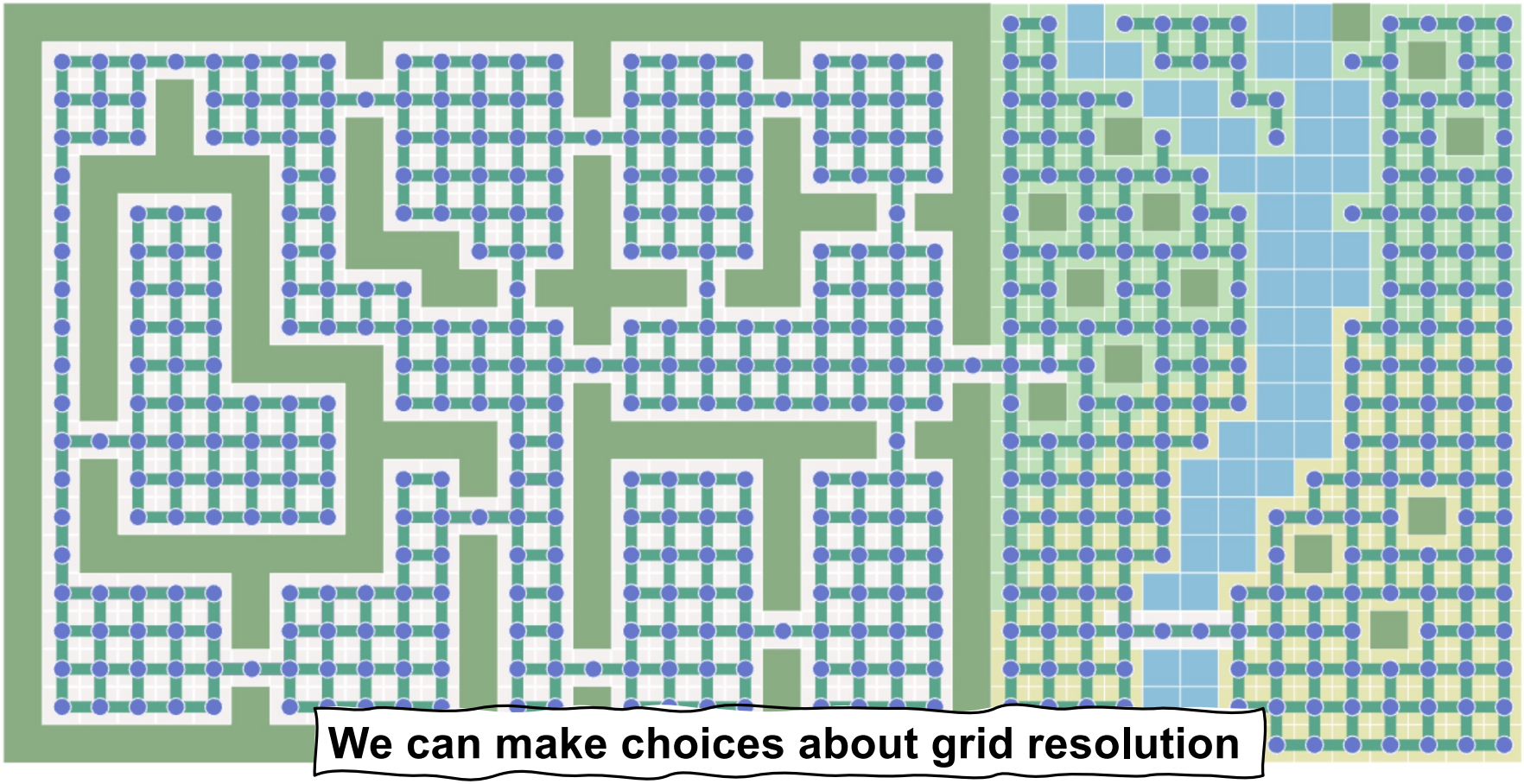


Single Robot Planning: A*



We can make choices about where nodes and edges go (e.g., doors)

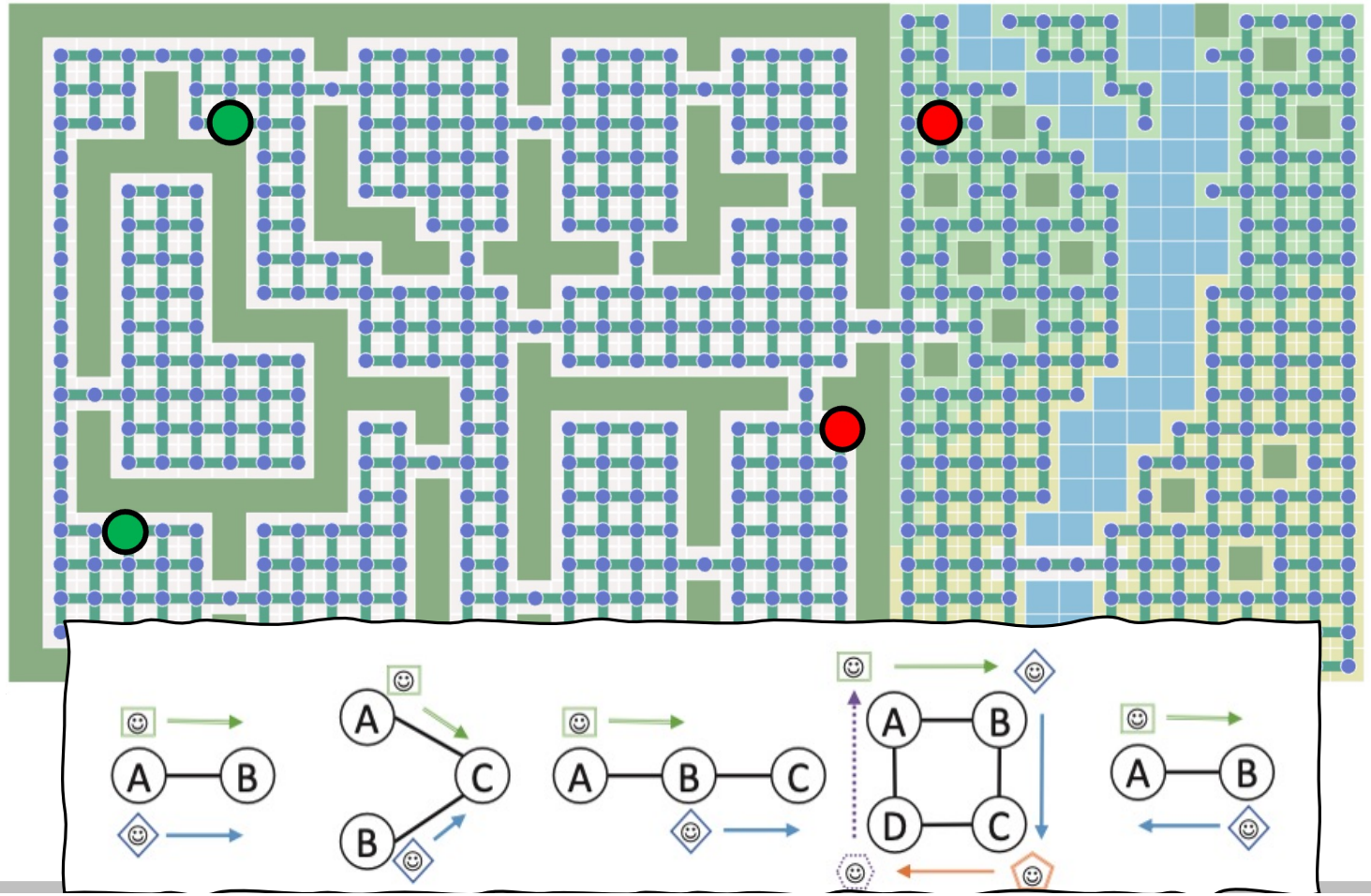
Single Robot Planning: A*



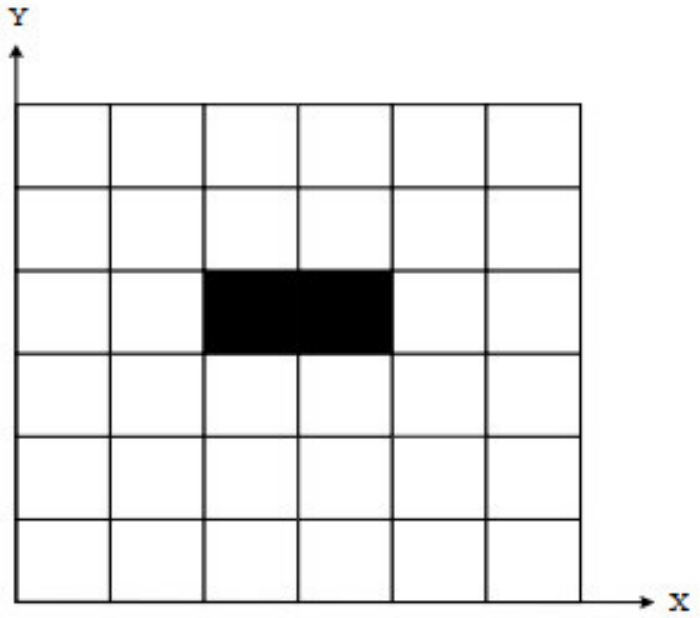
Single Robot Planning: A*

https://www.redblobgames.com/pathfinding/a-star/introduction.html

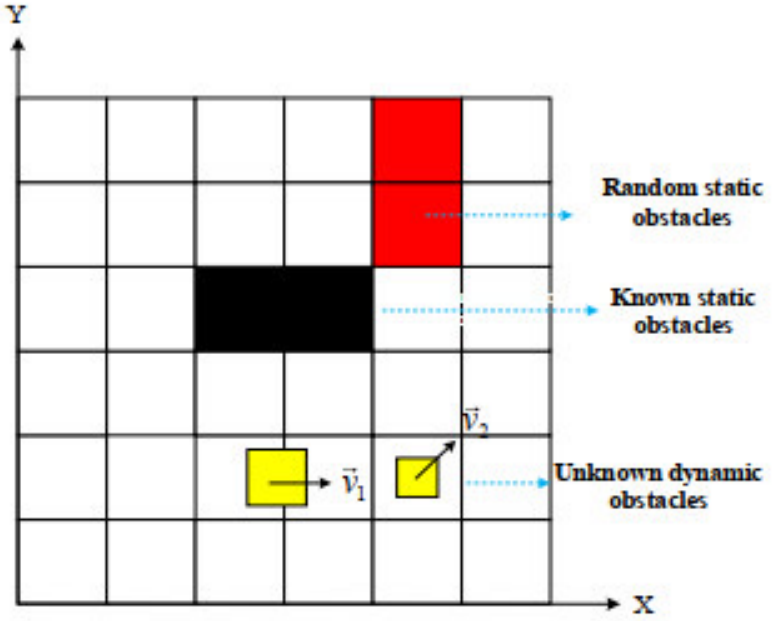
Search-based Methods for MRS



A* with Replanning: Local Repair (LRA*)

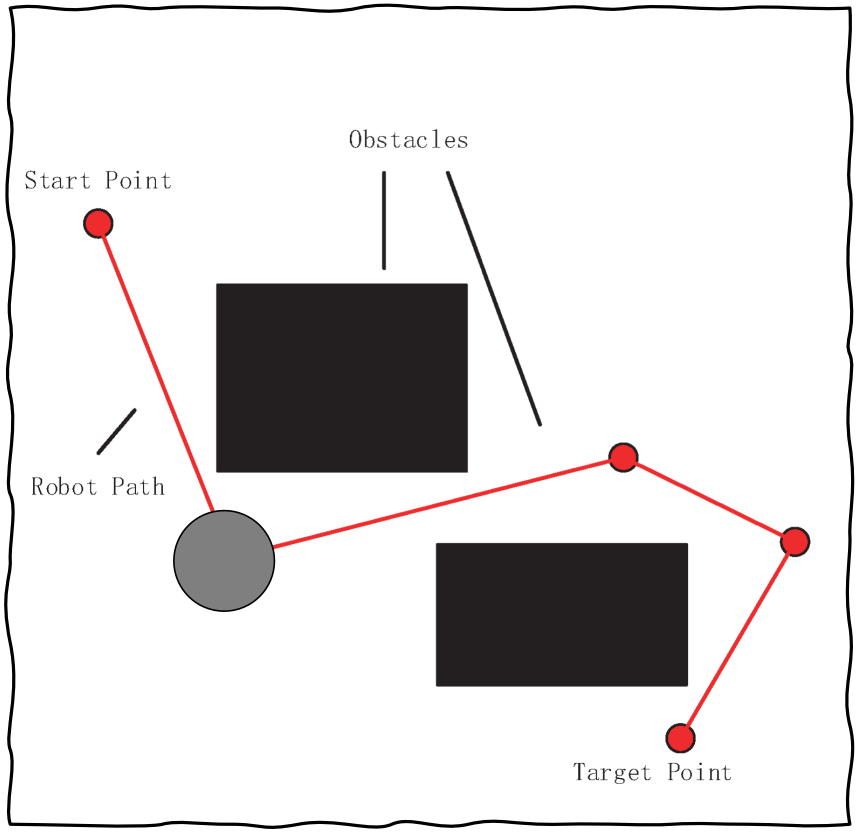


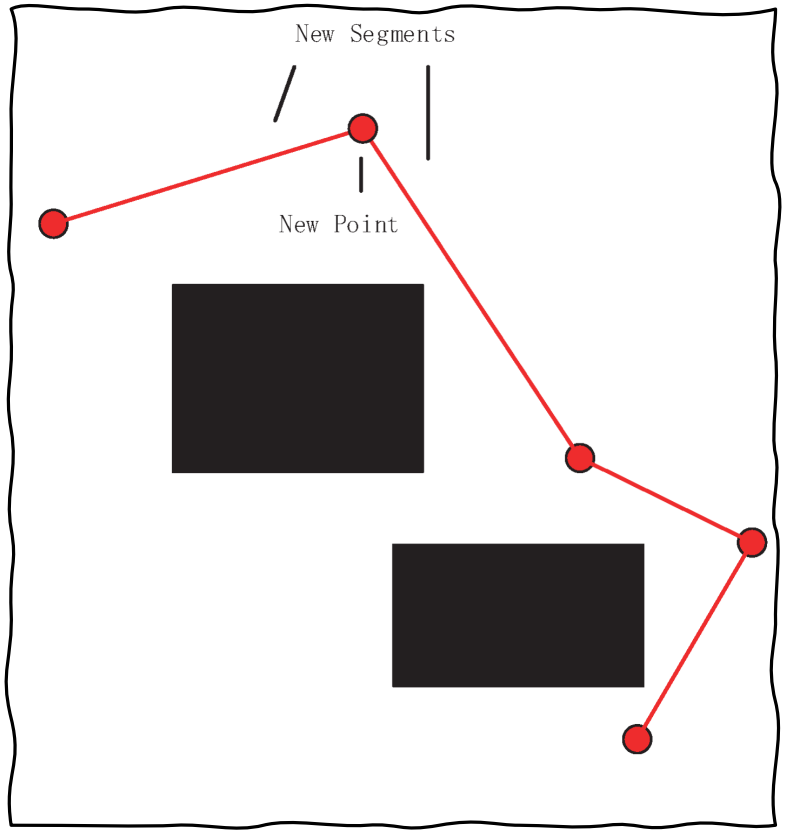
(a) Traditional global static environment



(b) Global static + local dynamic environment in this paper

A* with Replanning: Local Repair (LRA*)





Don't *need* to replan the entire route

A* with Replanning: Local Repair (LRA*)

Local replanning without additional rules leads to **collision cycles.**

One workaround: **“agitation”:** after a cycle, add random noise to the graph edges.

Cooperative A* (CA*)

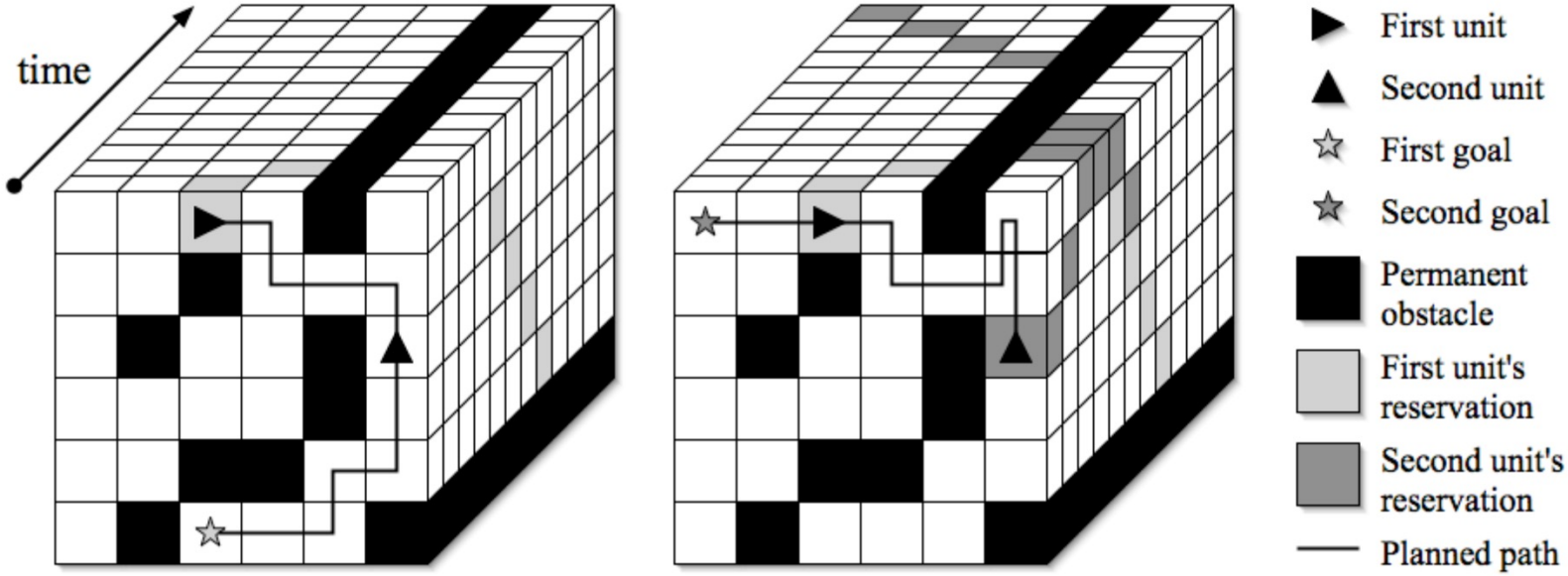


Figure 1 Two units pathfinding cooperatively. (A) The first unit searches for a path and marks it into the reservation table. (B) The second unit searches for a path, taking account of existing reservations, and also marks it into the reservation table.

$[x_i, y_i] \rightarrow [x_i, y_i, t_i]$

Add command $wait$

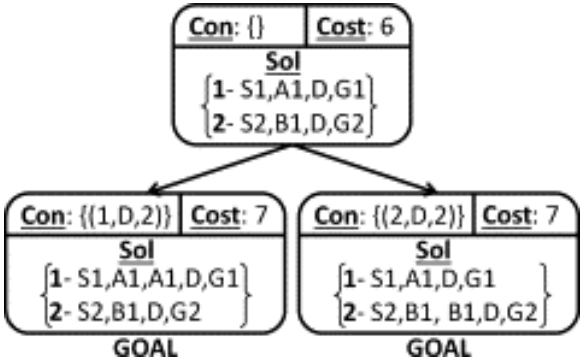
Challenge: curse of dimensionality

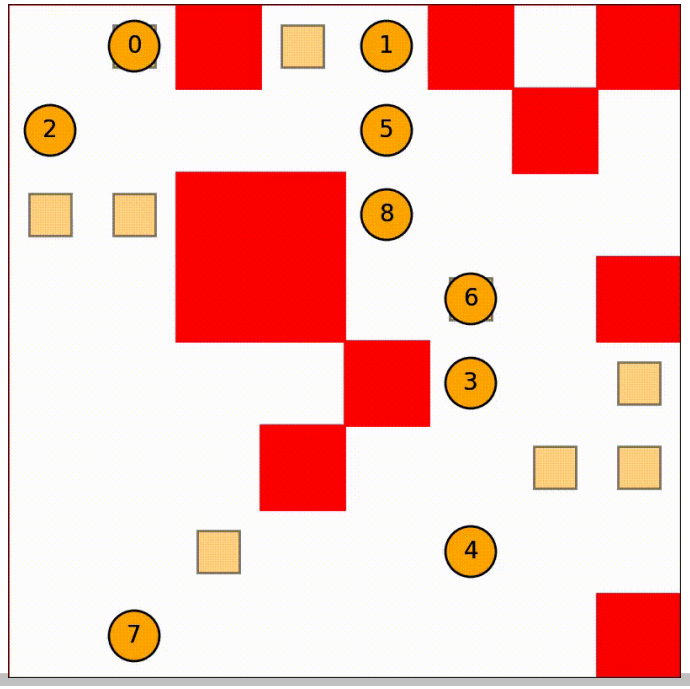
Conflict Based Search (CBS)

Input: MAPF instance
1 *Root.constraints* $= \emptyset$
2 *Root.solution* = find individual paths by the low level()
3 *Root.cost* = *SIC*(*Root.solution*)
4 insert *Root* to OPEN
5 **while** OPEN *not empty* **do**
6 $P \leftarrow$ best node from OPEN // *lowest solution cost*
7 Validate the paths in P until a conflict occurs.
8 **if** P *has no conflict* **then**
9 **return** P.*solution* // P *is goal*
10 C $\leftarrow$ first conflict (a_i, a_j, v, t) in P
11 [redacted]
18
19 **foreach** *agent* a_i *in* C **do**
20 $A \leftarrow$ new node
21 $A.constraints \leftarrow P.constraints + (a_i, v, t)$
22 $A.solution \leftarrow P.solution$
23 Update $A.solution$ by invoking low level(a_i)
24 $A.cost = SIC(A.solution)$
25 **if** $A.cost < \infty$ // *A solution was found* **then**
26 Insert A to OPEN

Conflict (a_i, a_j, v, t) (agent 1 and 2 occupy vertex at time)

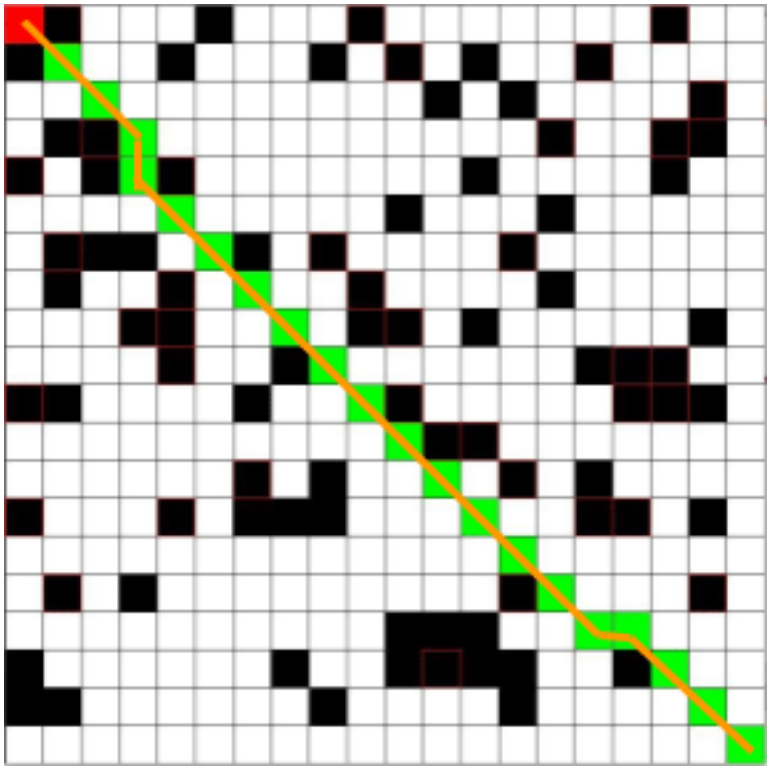
Constraint (a_i, v, t) (agent can't occupy vertex at time)






A* PATHFINDING
68 0 68
58 10 68
48 20 68
38 30 68
34 40 74
38 50 88
58 24 82
24 44 68
28 54 82
54 28 82
44 24 68
34 20 54
24 24 48
14 28 42
10 38 48
14 48 62
24 58 82
58 38 96
40 34 74
30 30 60
20 34 54
10 38 48
10 52 62
20 62 82
EPISODE 01

D* and D* Lite



Two separate lists maintained:

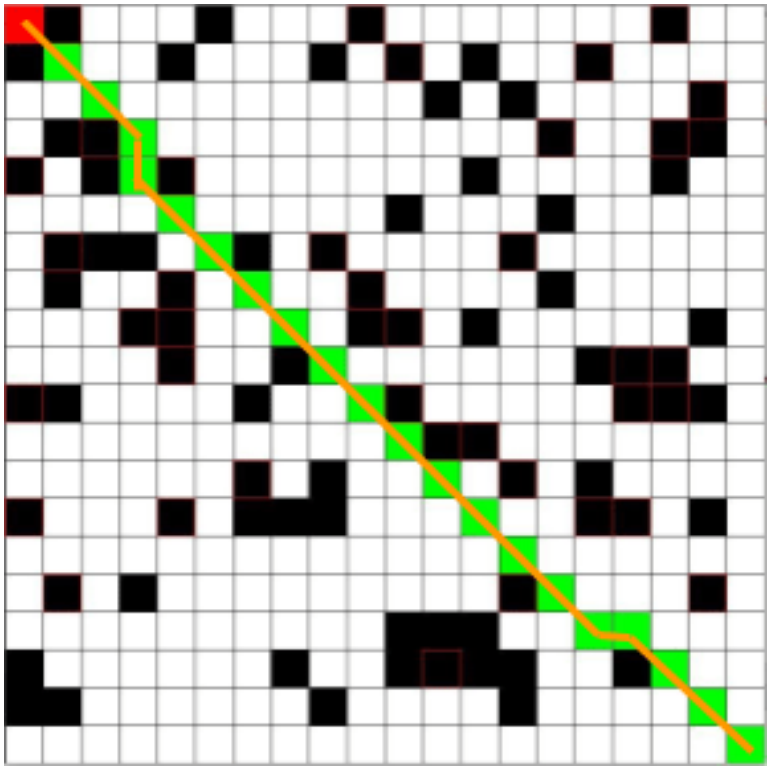
"**G**" score = Cost to connect the starting point to the current point

"**RHS**" score = Connection cost from current node to next node, inf. for obstacle; aka one-step lookahead cost

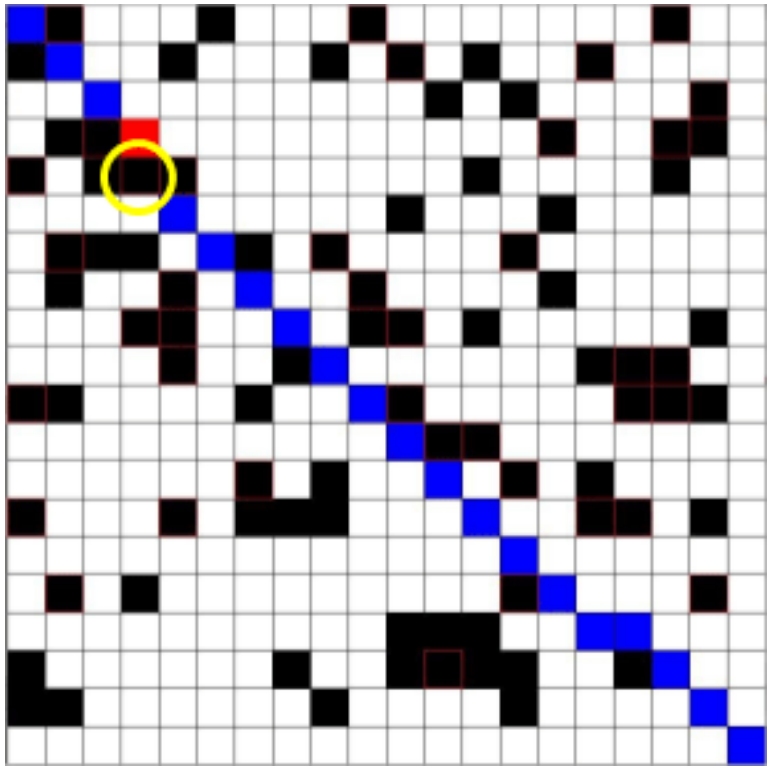
→when G(s) /=/ RHS(s), queue for update.

```
Terminology:
S = current node
S' = predecessor node
C (s', s) = edge cost for s' ? s
G(s) = current cost to arrive at that node
G(s) = G(s') + C (S', S)
RHS(s) = min value((G(S') + C (S', S))
```

D* and D* Lite

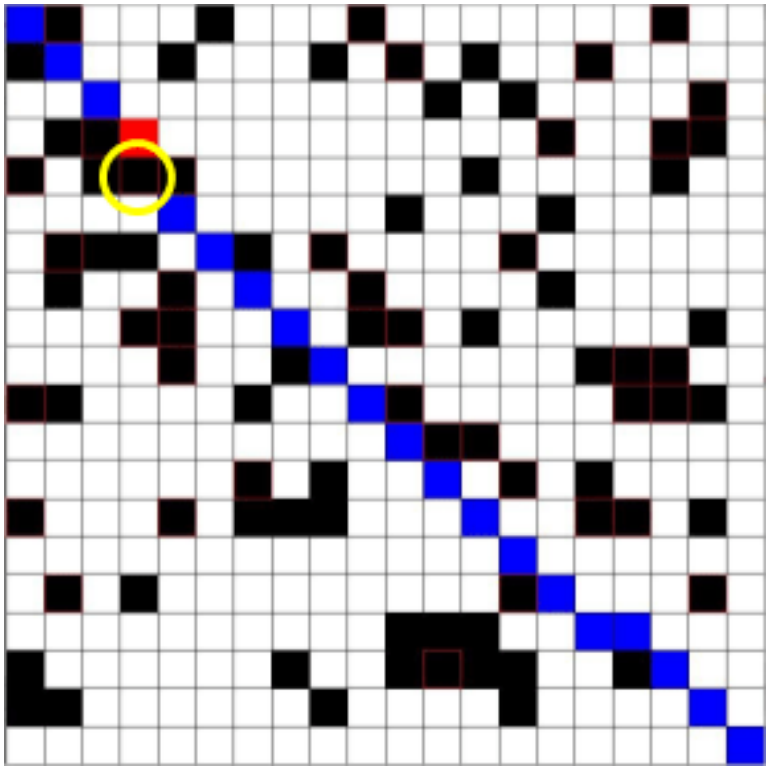


Initial plan

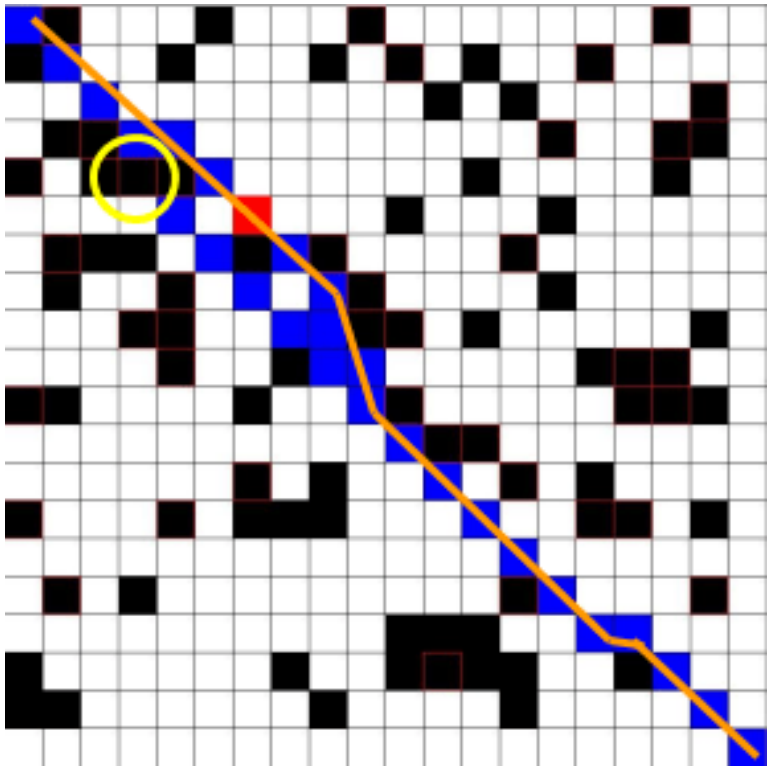


Obstacle encountered

D* and D* Lite

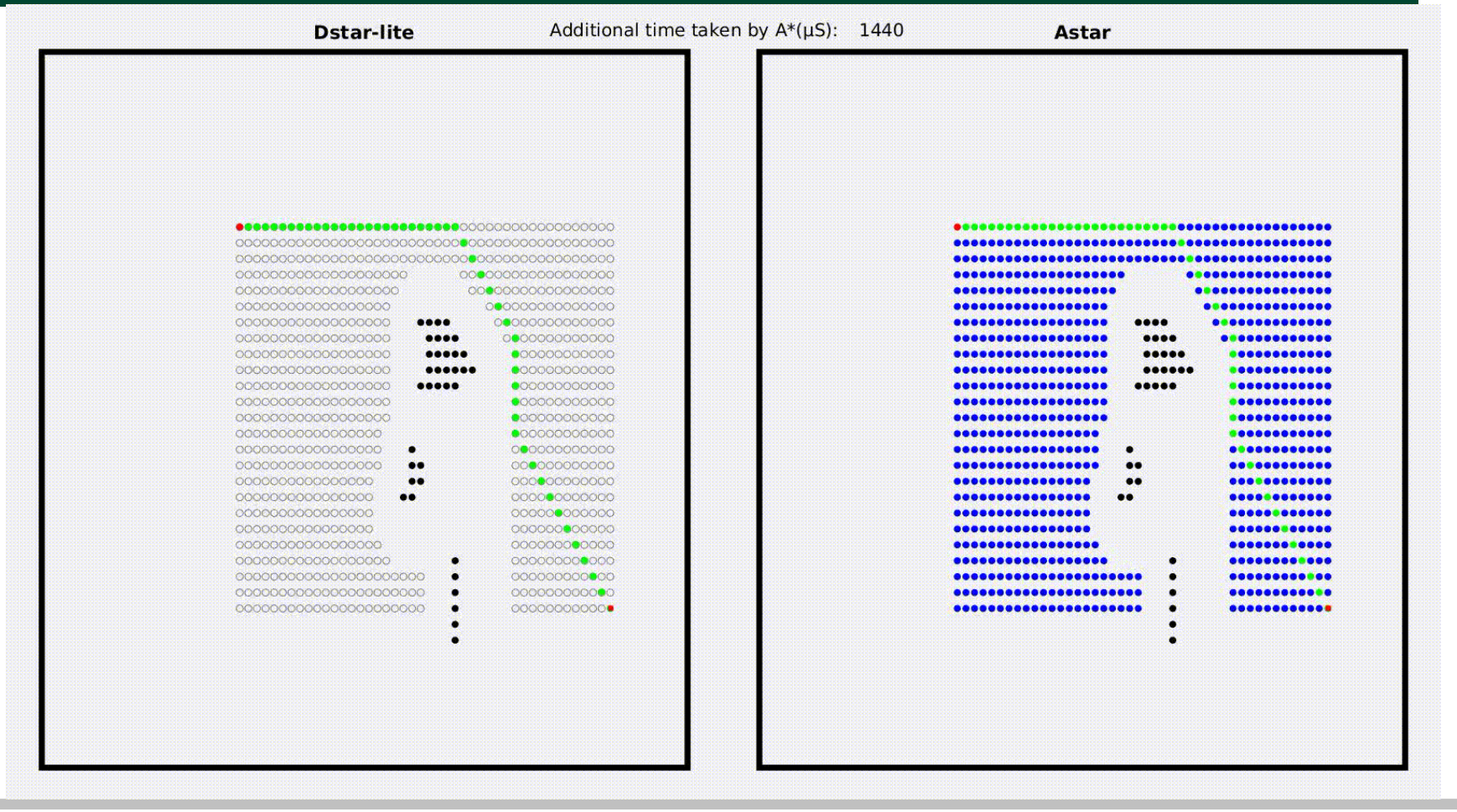


Obstacle encountered



Plan updated

D* and D* Lite



Contrasting These Approaches

A* with replanning:

Initial planning: A*
Replanning: Re-run A*
Efficiency: Poor
Optimality: High

D* Lite:

Initial planning: A*-ish in reverse
Replanning: Update lookahead cost RHS(s)
Efficiency: High
Optimality: Variable

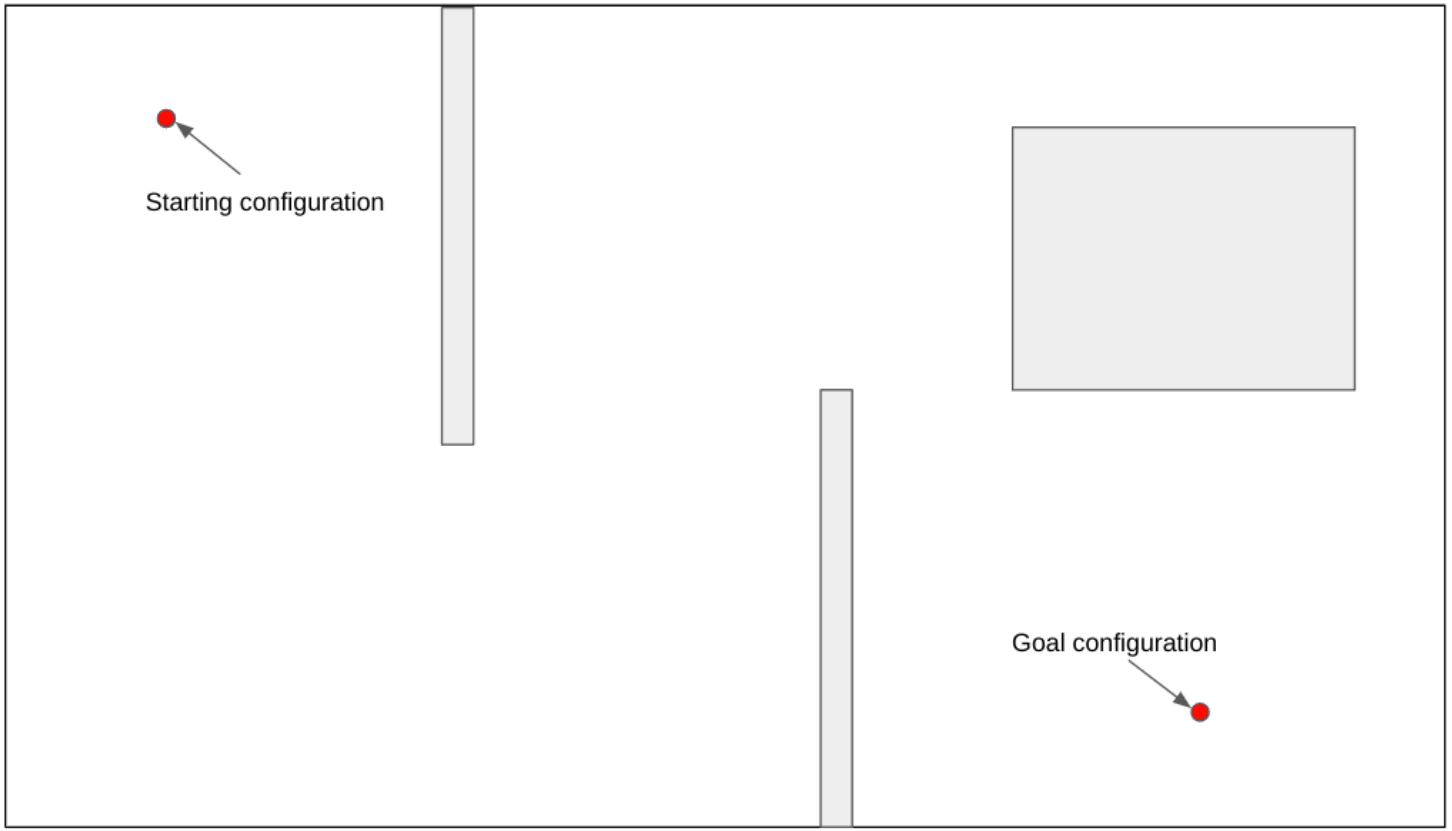
D* Lite for mostly dynamic environments, A* for mostly static

Single Robot Planning: Sampling-based Methods

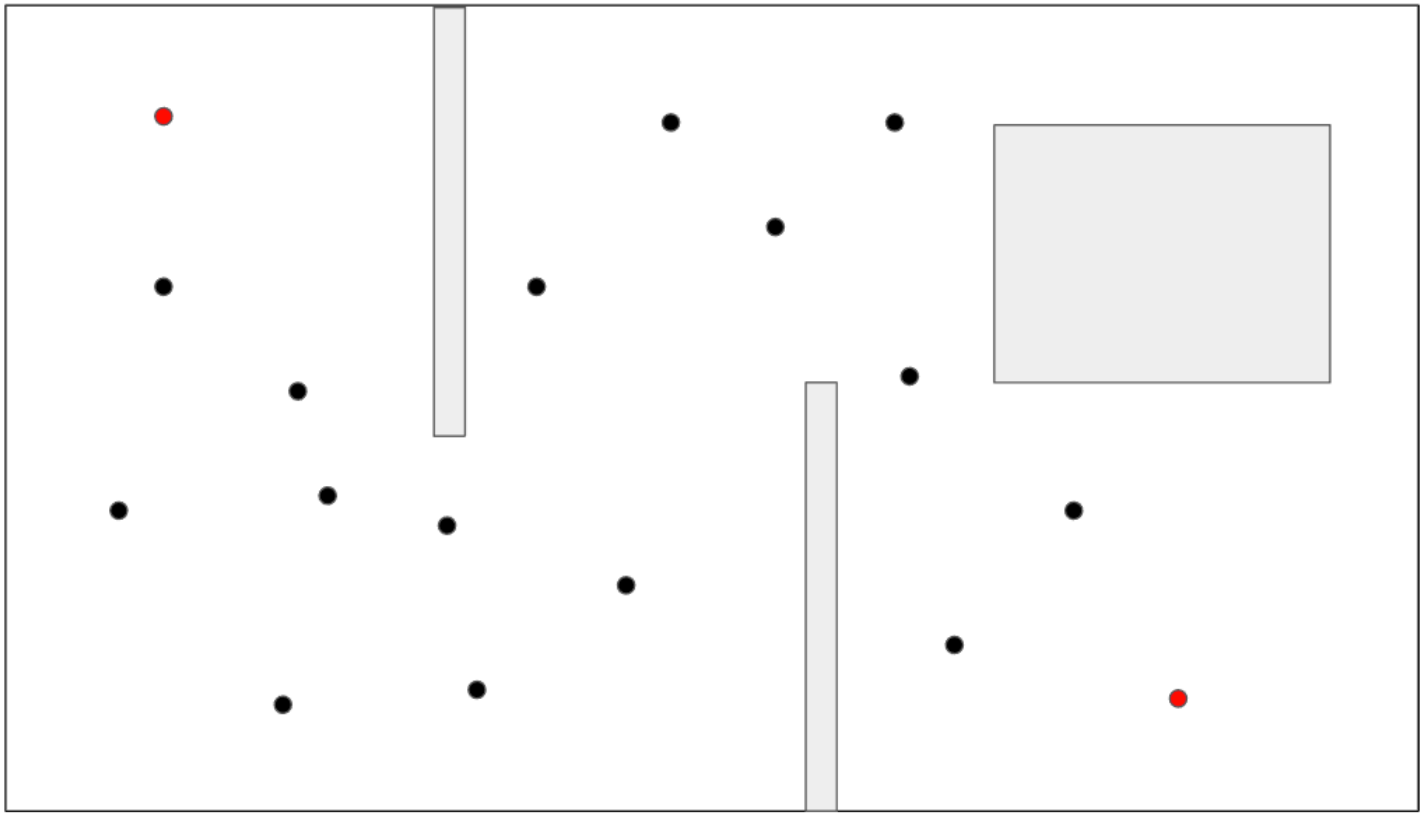
What do we do if we have no grid, or the required grid dimensions make computation infeasible, or we have many degrees of freedom?

Succinctly: Sampling-based methods are good **for large-scale, continuous environments with high-dimensional robots**

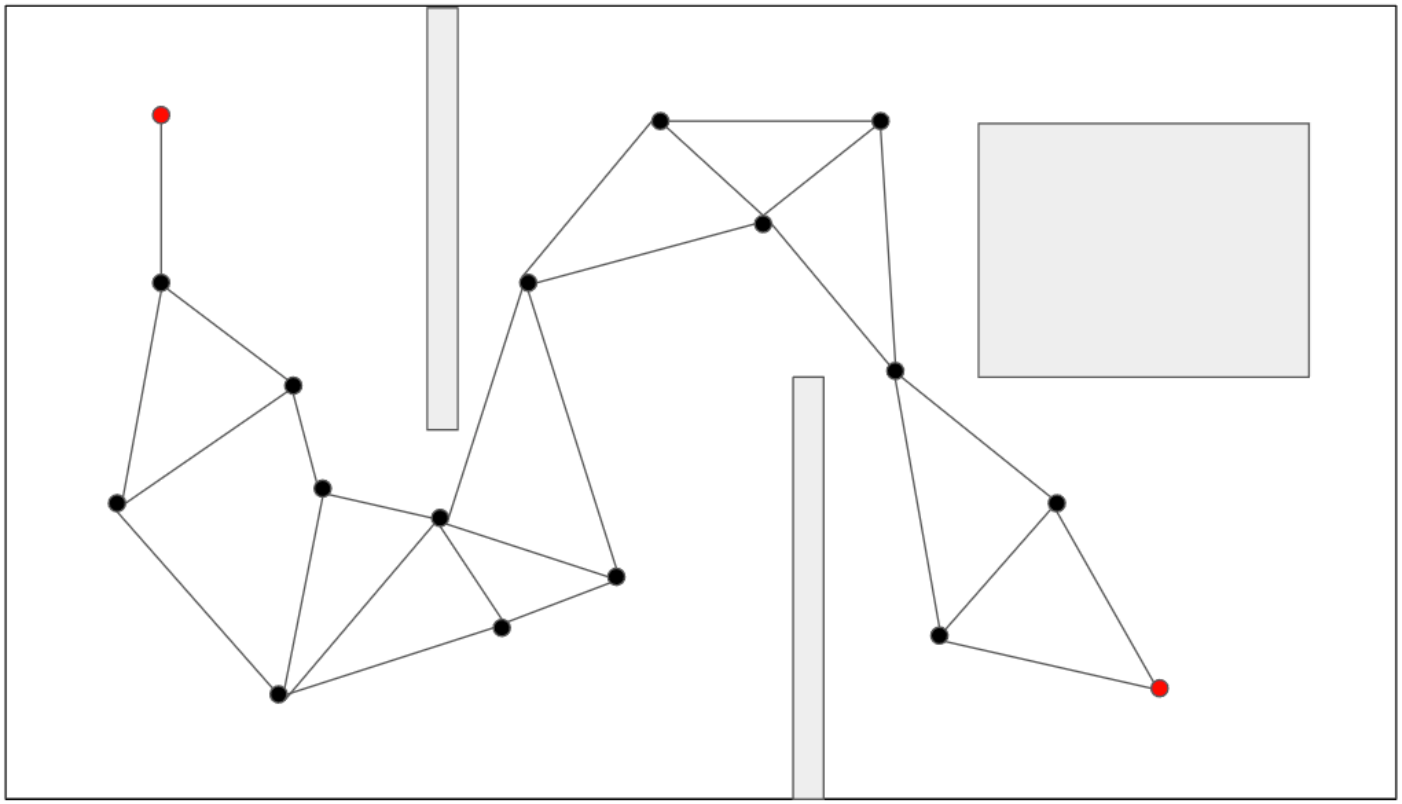
Single Robot Planning: Probabilistic Roadmap (PRM)



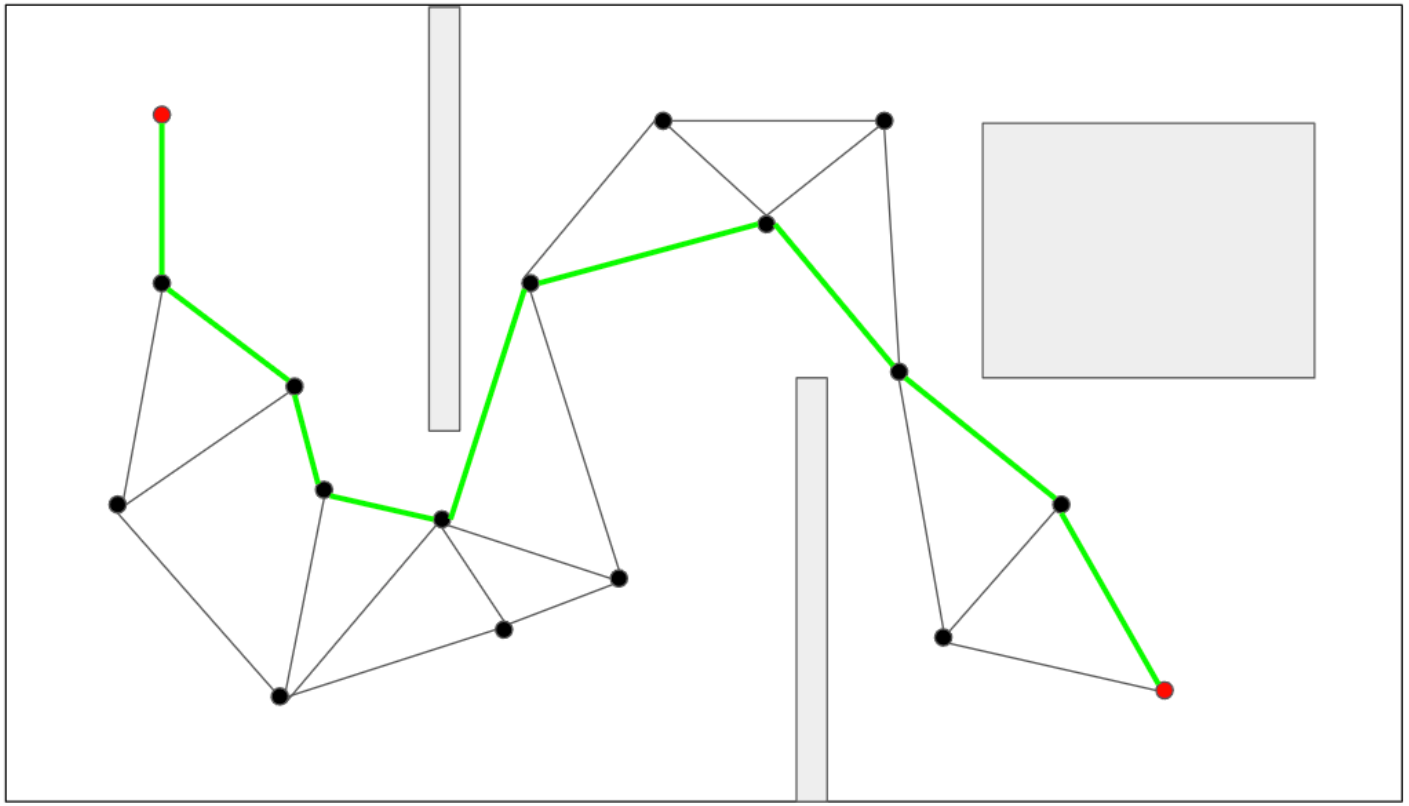
Single Robot Planning: PRM



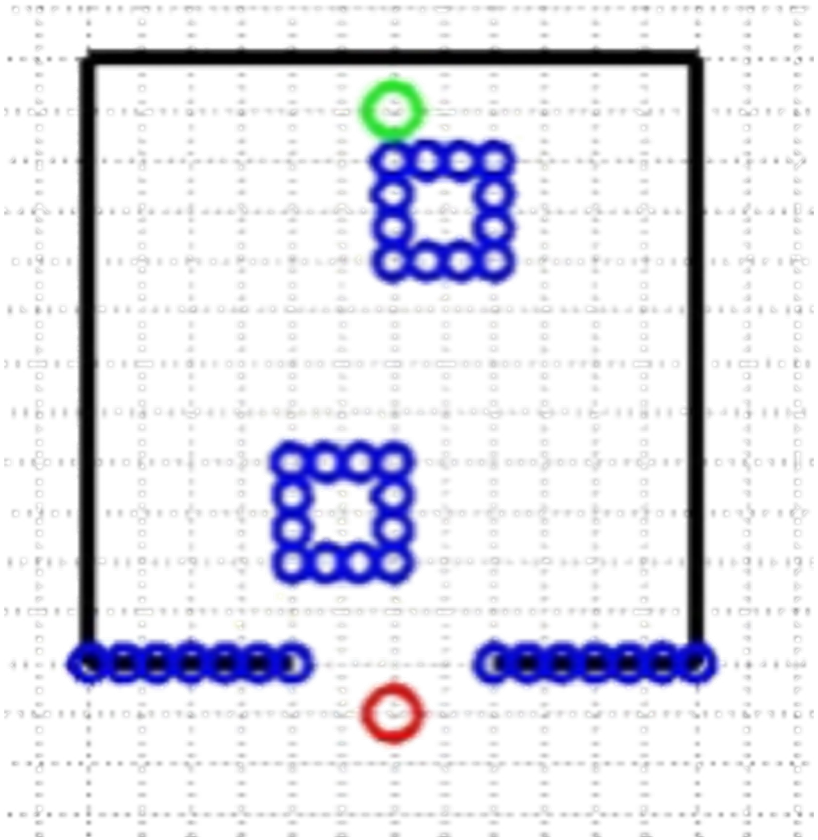
Single Robot Planning: PRM



Single Robot Planning: PRM



Single Robot Planning: RRT

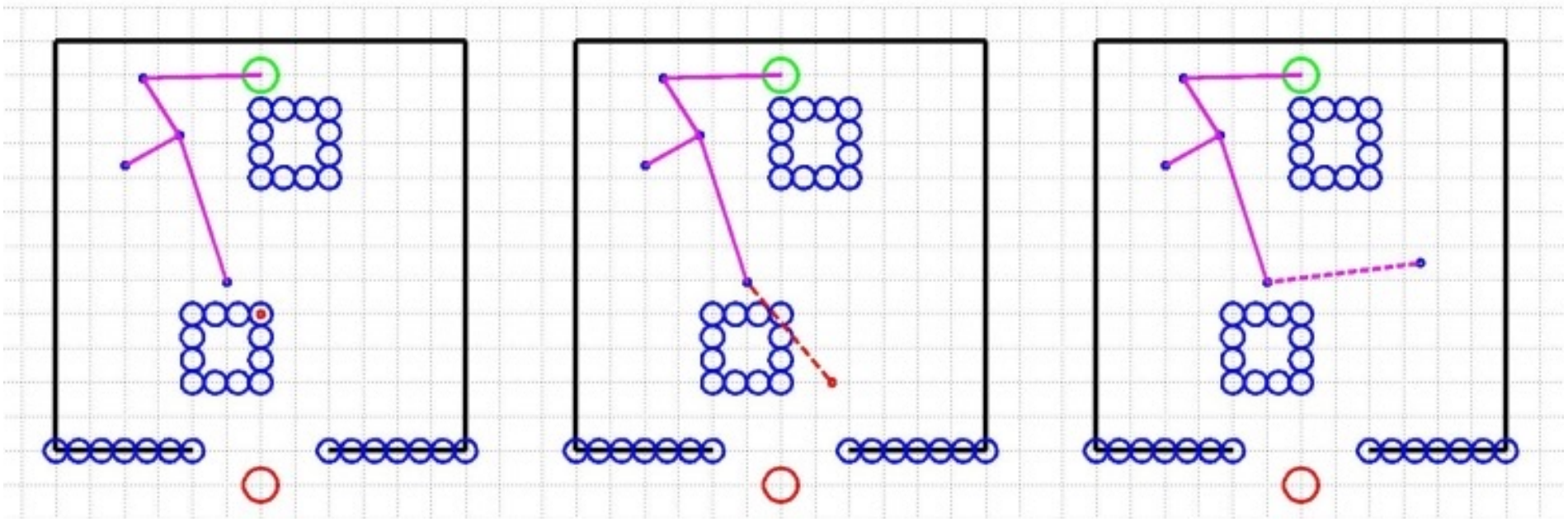


Instead of a discretized grid, positions are sampled from the continuous state space
(grid in images shown for visualization only)

Initial position $p_0 = [x_0, y_0]$

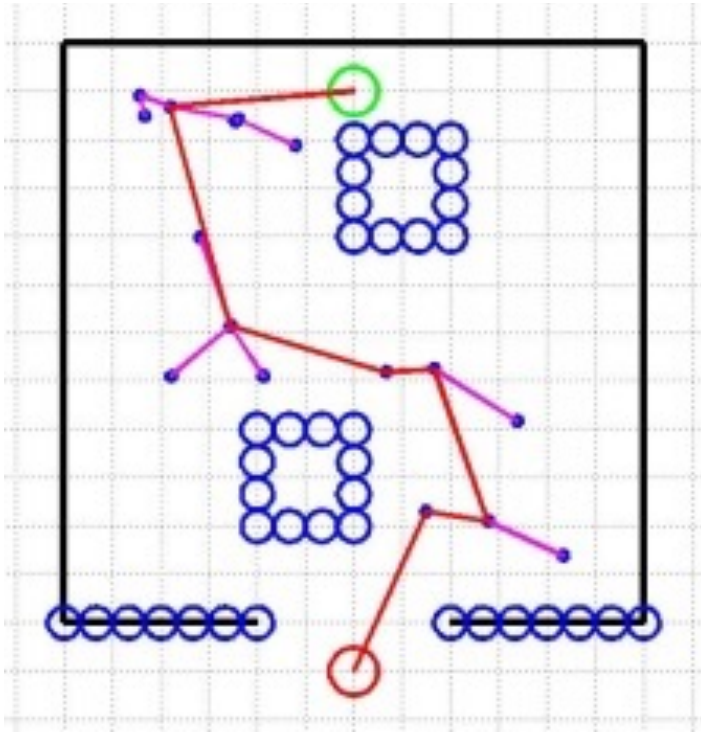
Target position $p_t = [x_t, y_t]$

Single Robot Planning: RRT

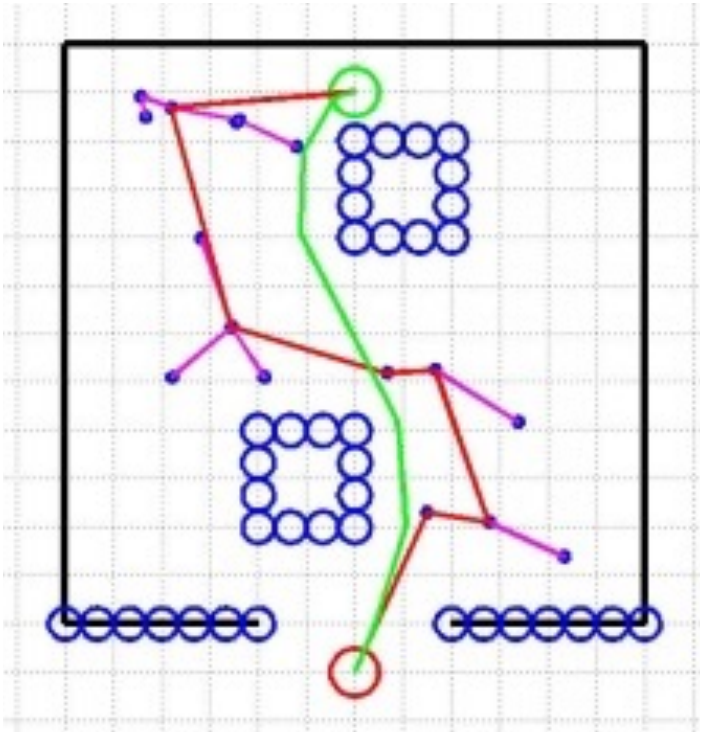


Points are added to the “tree”, then checked for 1) collisions with obstacles, 2) collisions with obstacles between nodes

Single Robot Planning: RRT



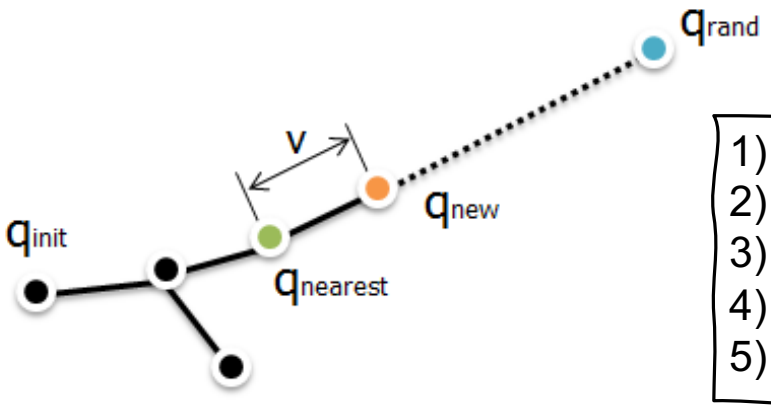
Valid path formed to target



Path can be “smoothed” by re-sampling between path points

Single Robot Planning: RRT

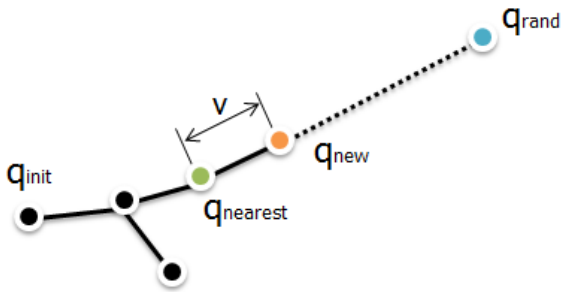
$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
while $i < N$ **do**
    $G \leftarrow (V, E)$;
    $x_{\text{rand}} \leftarrow \texttt{Sample}(i)$; $i \leftarrow i + 1$;
    $(V, E) \leftarrow \texttt{Extend}(G, x_{\text{rand}})$;



1) Find the nearest node in tree
2) Generate a new node in direction of sampled node
3) Collision check
4) Add new node to tree
5) Return status (reached/advanced/trapped)

Single Robot Planning: RRT

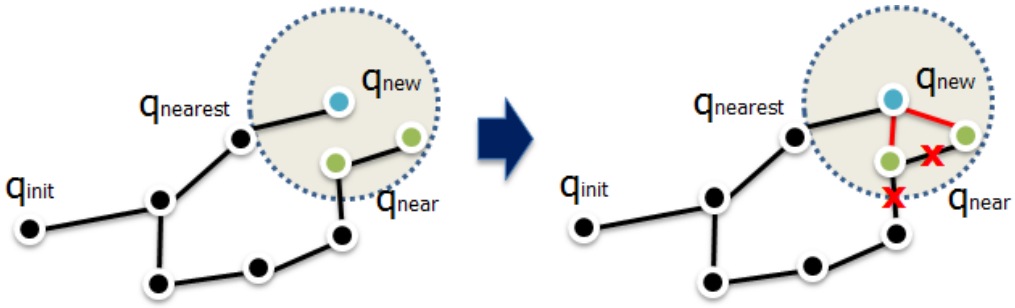
$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
while $i < N$ **do**
  $G \leftarrow (V, E)$;
  $x_{\text{rand}} \leftarrow \texttt{Sample}(i)$; $i \leftarrow i + 1$;
  $(V, E) \leftarrow \texttt{Extend}(G, x_{\text{rand}})$;



$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $E' \leftarrow E' \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G' = (V', E')$

Single Robot Planning: RRT*

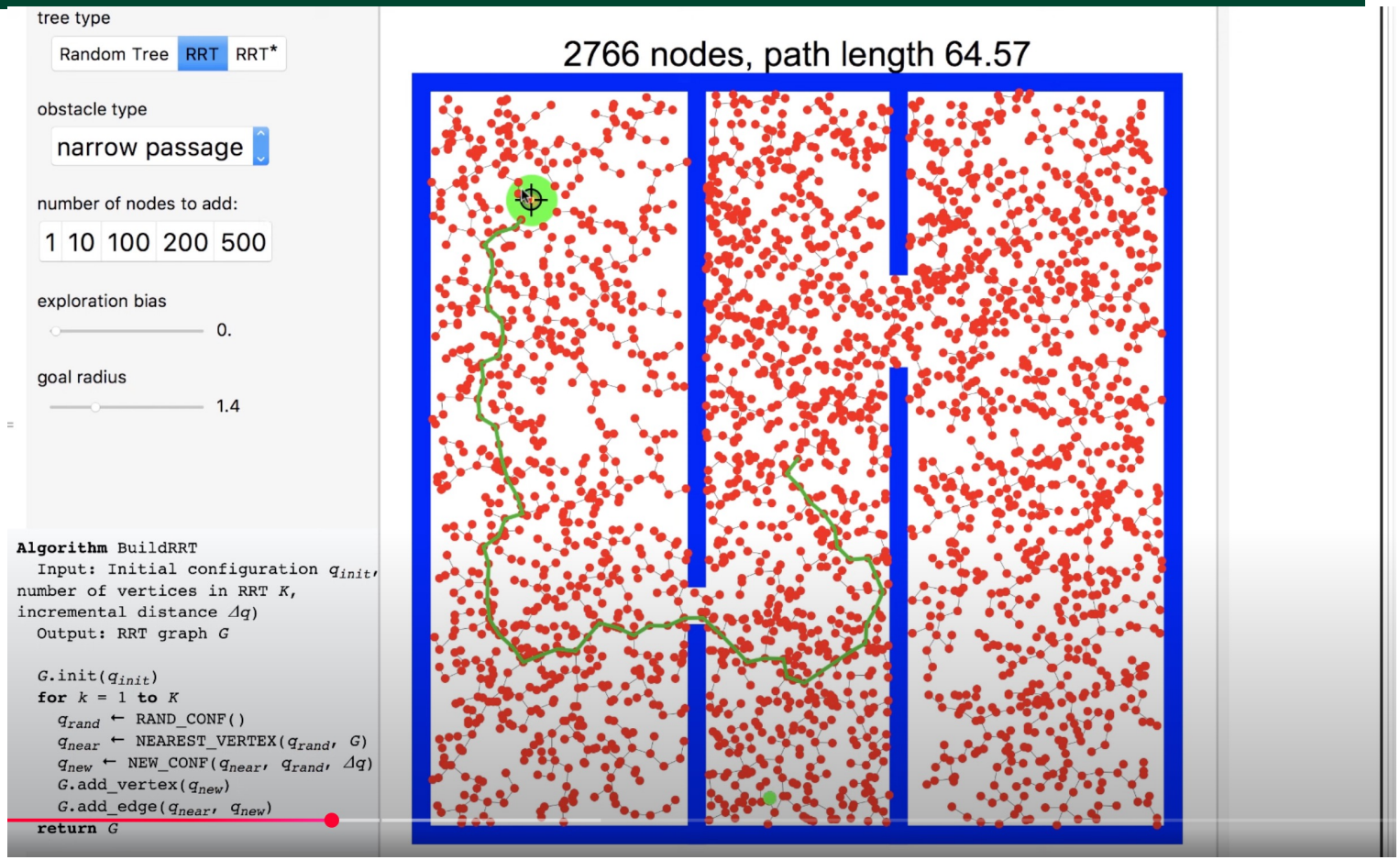
```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             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')
```



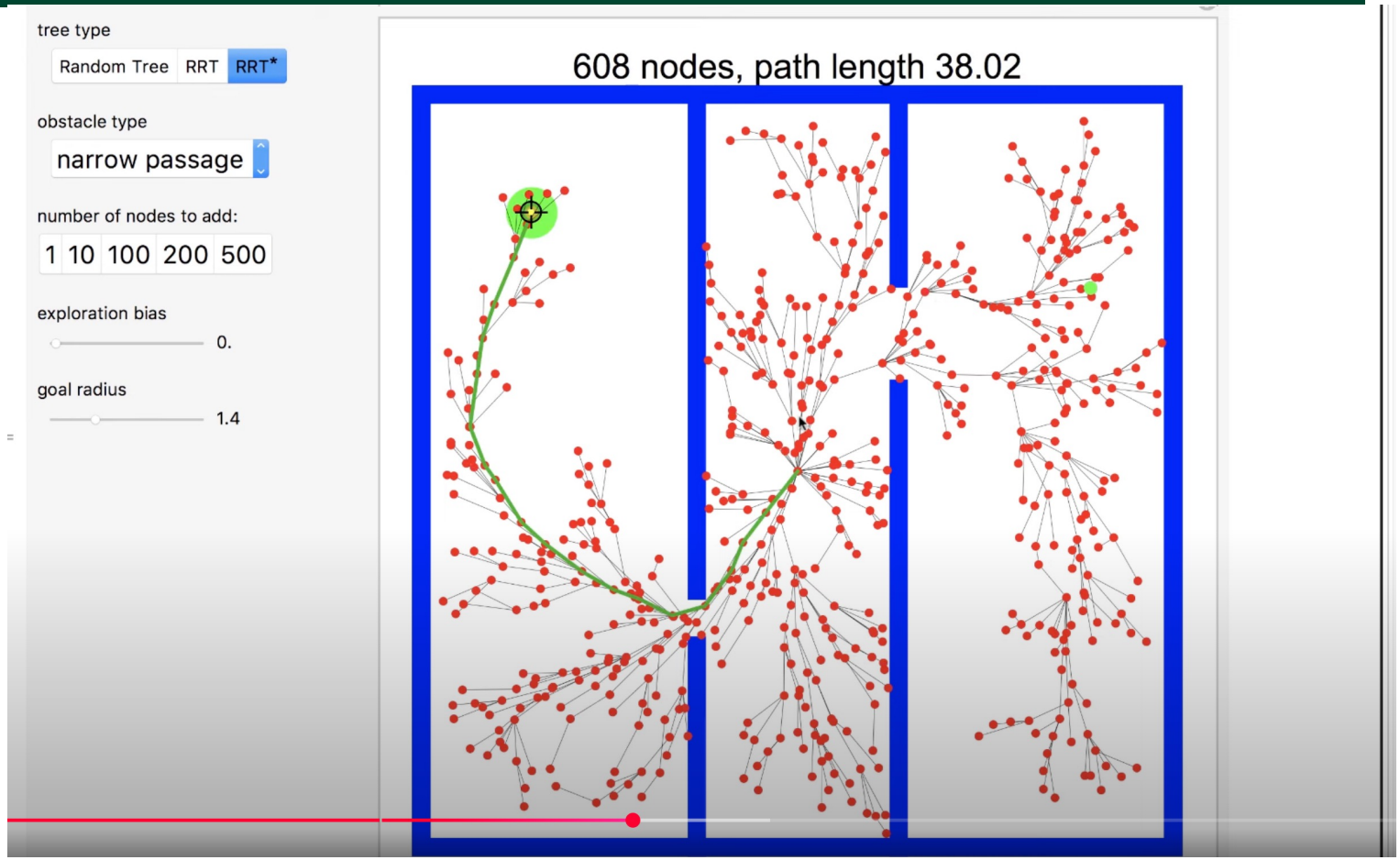
Continuously **"rewire"** the graph based on closest vertices

1) Find the nearest node in tree
2) Generate a new node in direction of sampled node
3) Collision check
4) **Check whether new node is closer to any existing nodes**
5) **Rewire tree to result in lowest total cost path**

Single Robot Planning: RRT Visualized



Single Robot Planning: RRT* Visualized



Multi-Agent RRT (MA-RRT*)

(first extend RRT* with **G-RRT***, which discretizes state space into a motion graph; when expanding from start, still samples points randomly, then uses graph-based search to find optimal path)

Then consider: **State space:** $\mathrm{x} = (w_1, \ldots, w_n)_i$ where w_1 is waypoint of agent 1; waypoint includes time

Distance between two joint spaces: $\sum_{i=1}^{n} cost_i^{LB}(x_i, y_i)$

Algorithm 4 GREEDY($G_M, \mathbf{s}, \mathbf{d}$)

```
1:  x ← s; c ← 0; path ← (∅, ..., ∅)
2:  while x ≠ d and c ≤ c_max do
3:      (path_i, ..., path_n) ← path
4:      for all x_i ∈ x do
5:          N ← children(G_M, x_i)
6:          x' ← arg min_{x∈children(G_M,x_i)} h(x_i)
7:          c ← c + cost(x_i, x'_i); path_i ← path_i ∪ (x_i, x'_i);
8:          x_i ← x'_i
9:      end for
10:     if not CollisionFree(path_1, ..., path_n) then
11:         return path
12:     else
13:         path ← (path_1, ..., path_n)
14:     end if
15: end while
16: return (x, path)
```

(obviously, this hurts our spatial intuition)

Discrete RRT (dRRT / dRRT*)

First compute *individual optimal roadmaps* using G-RRT*
THEN sample from joint configuration space of the roadmaps

Why? The joint configuration space is highly dimensional; faster to solve individual roadmaps then only sample and replan from those “good” solutions

Forming the solution set

For each robot:

- If H_i and U_i are empty, add to S
- If H_i is not empty, reject from S
- Otherwise, consider the **cost** of adding to S

$$cost(i) = |U_i|$$

Number of robots that, if i is added, would be rejected from S

- If $cost(i) \leq cost(j)\ \forall\ j \in U_i$, add to S

Example:

$H_i = \emptyset$	$H_j = \emptyset$
$L_i = \emptyset$	$L_j = \emptyset$
$U_i = \{j\}$	$U_j = \{i, k\}$

*$|U_1| = 1, |U_2| = 2$, so robot 1 will be added to solution set

Quick Summary

For scenarios with an omniscient planner: A* is ubiquitous (e.g., video games)
For highly dynamic environments: D*-lite is state of the art
For large, high-dimensional state spaces: dRRT* is state of the art

Multi-agent coordination: D*-lite does not explicitly account for other robots!

Common approach: "prioritized planning", where high-priority robots maintain their path, low-priority robots must replan.

1. Lecture (~20 minutes)
2. Subsystem check-in (~40 minutes)

Questions?

Lecture 1: MRS Survey
Lecture 2: Motion Planning
Lecture 3: Task Assignment (+Motion Planning if needed)
Lecture 3: Communication and Control Paradigms

693H Robotic Subsystems

-Second Design Review Presentations are **3/10, 3/12**
-You will be expected to have **functional prototypes** complete to show by then
-You will be expected to have **PCB designs and final BoMs** complete by then