[Week 4]
Path Planning

授課教師：郭重顯 教授
助教：Nguyen Thanh Thien Phuc、莊明洋
助教實驗室：工綜 106

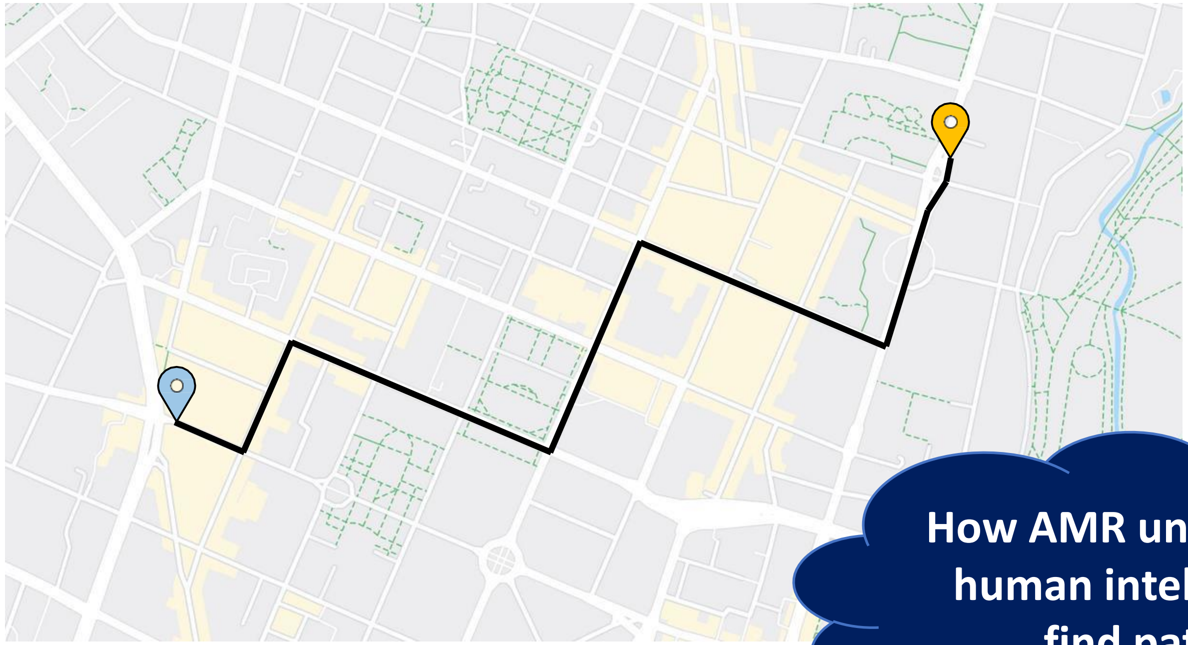
Motivation

- The human approach to finding routes on a street map is a very visual way
- Feasible routes/paths are developed by visually tracing different street lines in the general direction of the target

Motivation

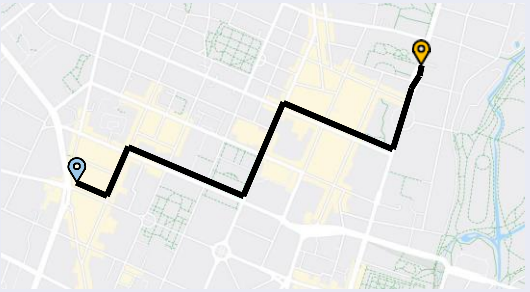
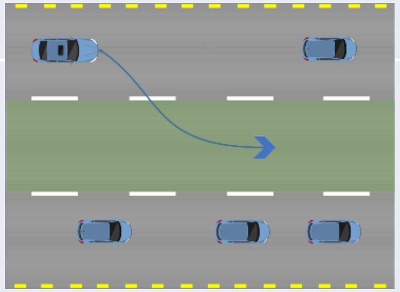
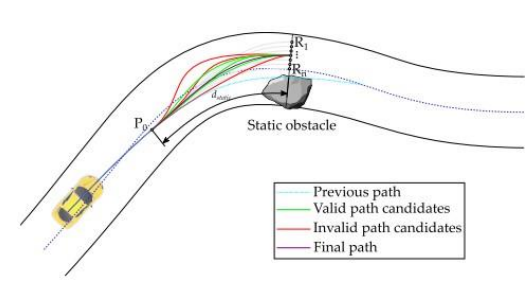
❖ The human approach to finding routes on a street map is a very visual way

❖ Feasible routes/paths are developed by visually tracing different street lines in the general direction of the target



How AMR understands human intelligent to find paths?

Definition of terms

Global Planning	Uses map for planning without information of local environment. Focus: hours to minutes.	
Behavior Planning	High-level description of the vehicle motion. Focus: minutes to seconds.	
Local Planning	Consideration of local objects. Deals with reactive decisions. Focus: seconds to milliseconds.	

Global vs Local Planning

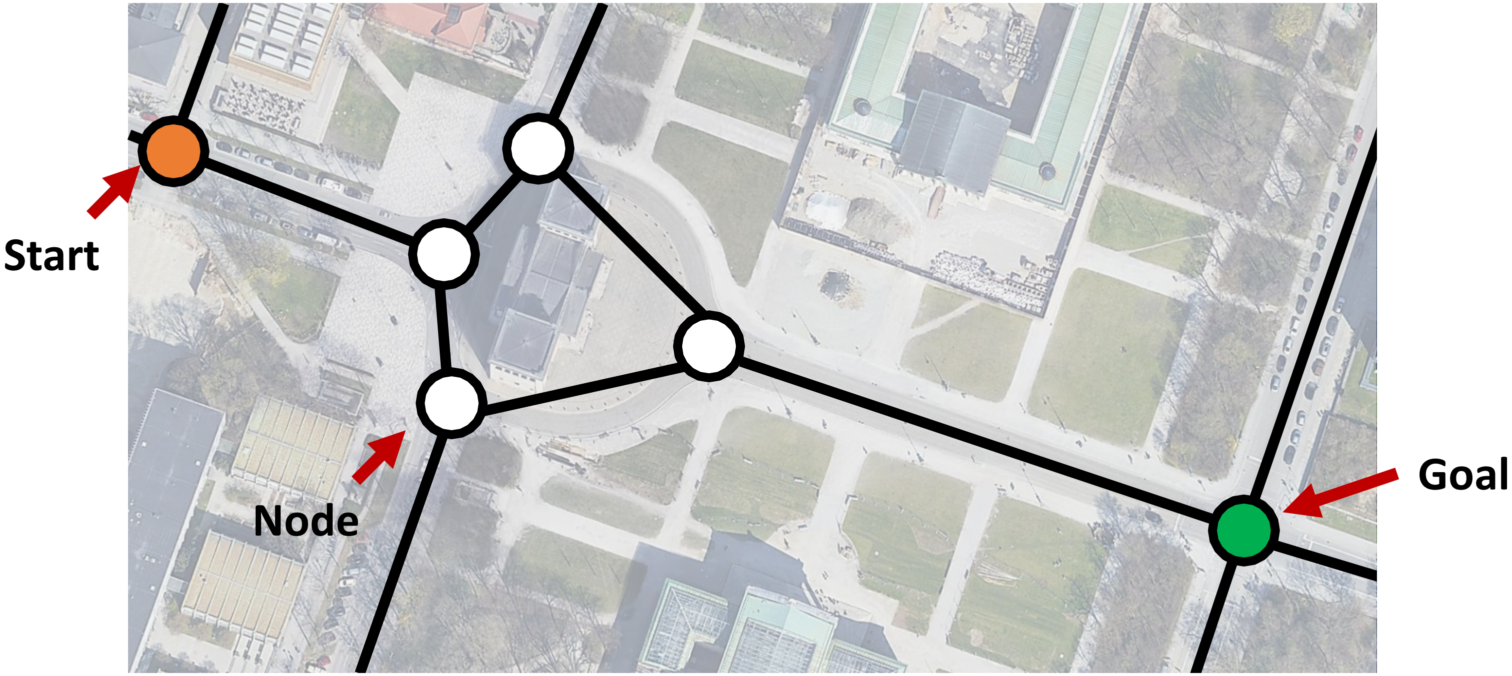
Global Planning	Local Planning
Map based	Sensor based
Relatively slower response	Fast response
Known workspace	Incomplete workspace
Generate path/route before moving	Planning and moving at the same time

Route/Path/Trajectory Planning

❖ **Route Planning**

❑ Decision to take a route from source (start) to destination (goal)

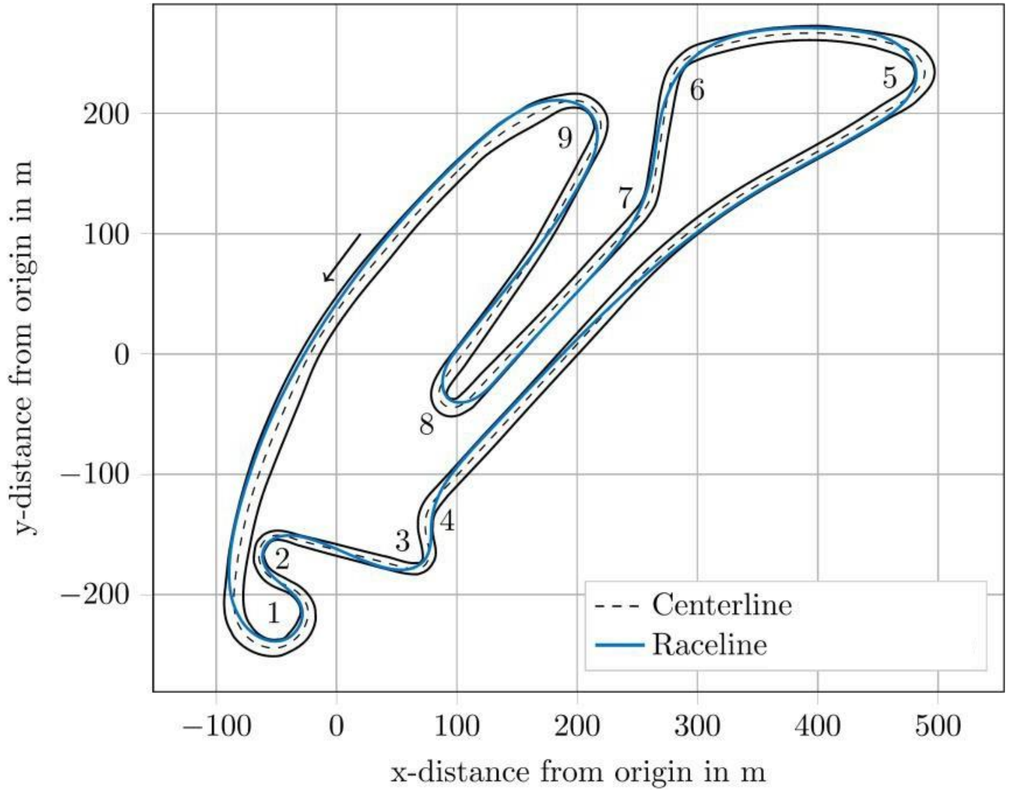
❑ Sequence of discrete geometrical nodes in map network



Route/Path/Trajectory Planning

❖ **Path Planning**

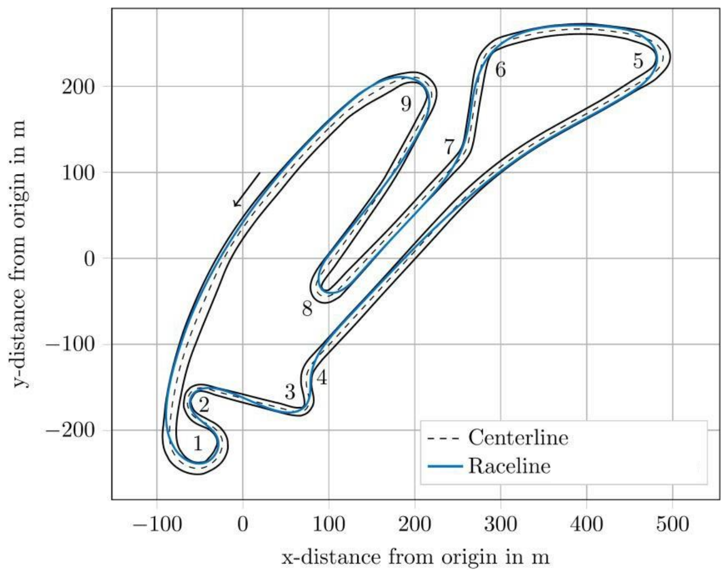
- ❑ Continuous curve in spatial domain
- ❑ Consideration of spatial / geometrical boundary conditions possible (e.g., track width, maximum curvature)



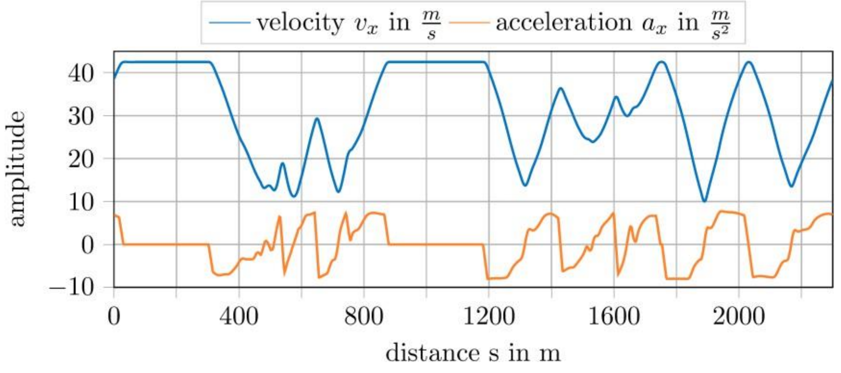
Route/Path/Trajectory Planning

❖ **Trajectory Planning**

- ❑ Continuous curve in **spatial-temporal** domain
- ❑ Further conditions can be checked by **temporal information** (e.g., freedom from collisions, compliance with acceleration limits)



+

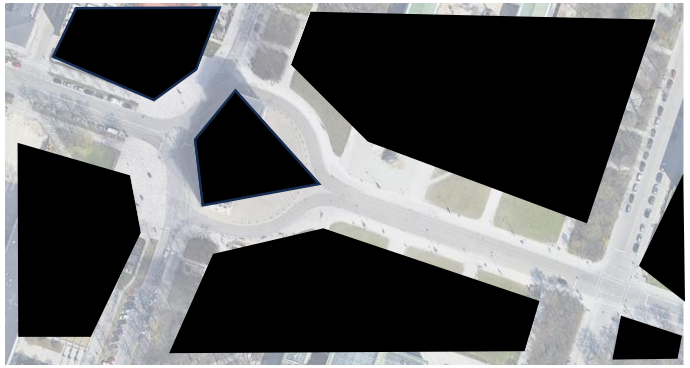


Global Route Planning vs. Global Path Planning

- **Global Route Planning** is generally used to generate a **route** from a source node to a destination node
  - Certain **target** variables, such as distance or time, can be **minimized** in this context
  - **Exact** geometric curve lines of the vehicle **are not** taken into account

- **Global path planning** considers the continuous planning of the **vehicle's curve**
  - Spatial and geometrical boundary conditions must be observed (e.g., track width, maximum curvature)

Global Planning Framework



Continuous Representation

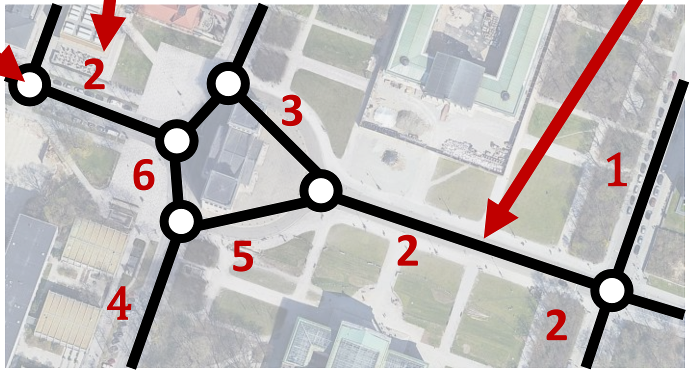
(**configuration space** formulation)

Weight (cost such as time, distance)

Node (start/goal point, obstacles)

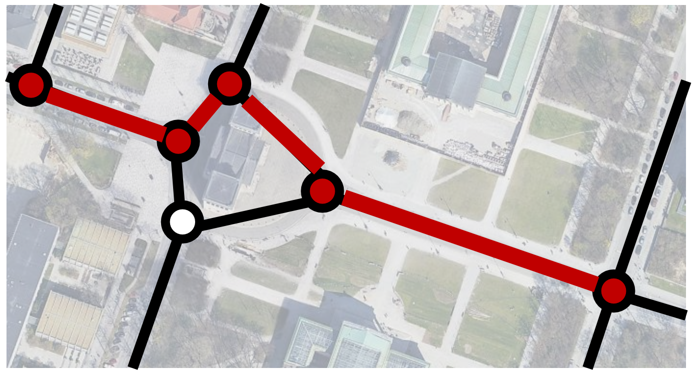
Visibility graph

Edges (exists between nodes)



Discretization

(random sampling, processing critical geometric events)



Graph Searching

(Dijkstra, A*)

Global Planning – Dijkstra’s Algorithm

- Dijkstra’s Algorithm invented by Edsger W. Dijkstra in 1956
  - Finding optimal path from source node to destination node
  - Graph-based algorithm
  - Informed, complete and optimal path search algorithm

- **Assumptions**
  - All edges are weighted by costs
  - No **negative** weighted costs
  - Path to start node has no costs
  - Starting node has no predecessor
  - For initialization, costs to all other nodes are infinite

Dijkstra, EW 1959, 'A note on two problems in connexion with graphs', *Numerische Mathematik*, vol. 1, pp. 269-271. https://doi.org/10.1007/BF01386390

Global Planning – Dijkstra's Algorithm

You can try: https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

❖ Steps

1. Initialization: Starting node=0, other nodes=∞
2. Distance of starting node is set to permanent, all other distances are temporarily
3. Setting starting node as **active**
4. Calculation of the temporary distances to the **current active node**:
   i. Consideration of all **reachable neighbor nodes**
   ii. Summing up its distance with the weights of the edges
5. If calculated distance is smaller as the current one:
   i. Update the distance
   ii. Set the current node as antecessor
6. Setting node with minimal temporary distance as **active**. Mark its distance as permanent.

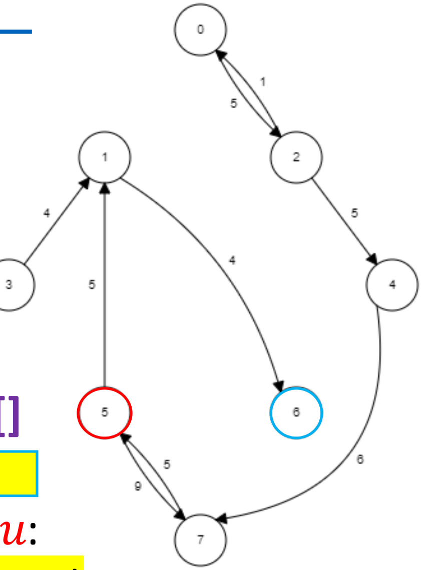
Explore all shortest paths until destination is reached

Pseudocode:

```
1:  function Dijkstra(graph G, source s):
2:      for each v in G:
3:          dist[v] ← ∞
4:          previous[v] ← Null
5:      dist[s] := 0
6:      while G is not empty:
7:          u := node in G with smallest dist[]
8:          remove u from G
9:          for each reachable neighbor v of u:
10:             tmp := dist[u] + dist_between(u, v)
11:             if tmp < dist[v]:
12:                 dist[v] := tmp
13:                 previous[v] := u
14:     return dist[], previous[]
```

u equals s at the 1[st] time

u is active now

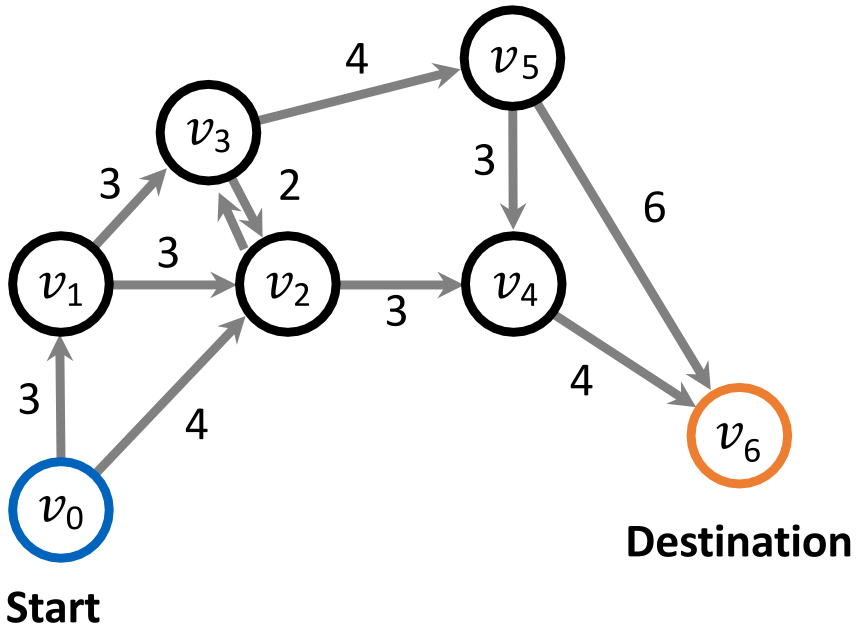


5(u)→1(v) feasible (5)
5(u)→7(v) feasible (5)

Hint: If there are some nodes never reach the goal, and they will go through the final selection of G in step 7 of the minimum distance, and that node will be removed in step 8; however, the steps 9-13 will never be executed because the distance is large/ infinite (beginning setting of **dist[]).**

Global Planning – Dijkstra's Algorithm

for each v in G:
  dist[v] ← ∞
  previous[v] ← $Null$
dist[s] := 0



s = 0 in this case

d = distance
P = predecessor
c = considered

Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

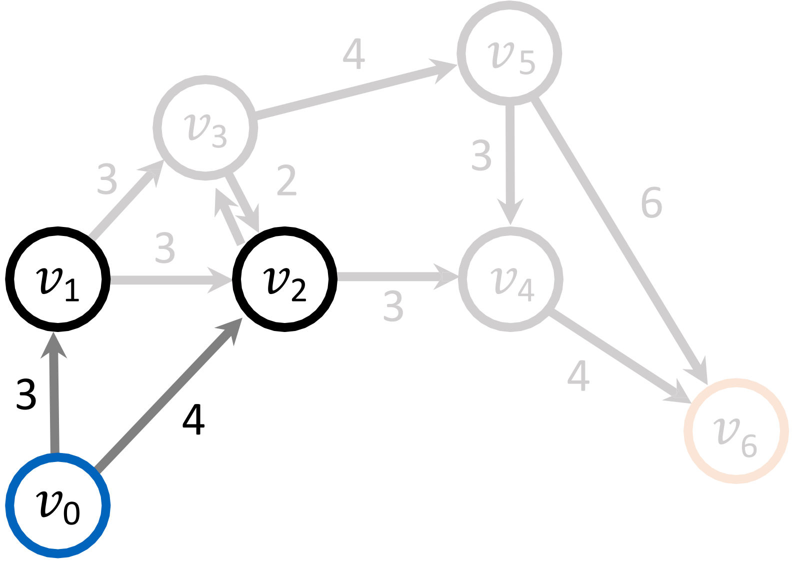
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	∞	∞	∞	∞	∞	∞

Global Planning – Dijkstra's Algorithm

```
while G is not empty:
   u := node in G with smallest dist[]
   remove u from G
   for each neighbor v of u:
      alt := dist[u] + dist_between(u, v)
      if alt < dist[v]:
         dist[v] := alt
         previous[v] := u
return dist[], previous[]
```

Active node stays in G; otherwise remove inactive node from G

d = distance
P = predecessor
c = considered



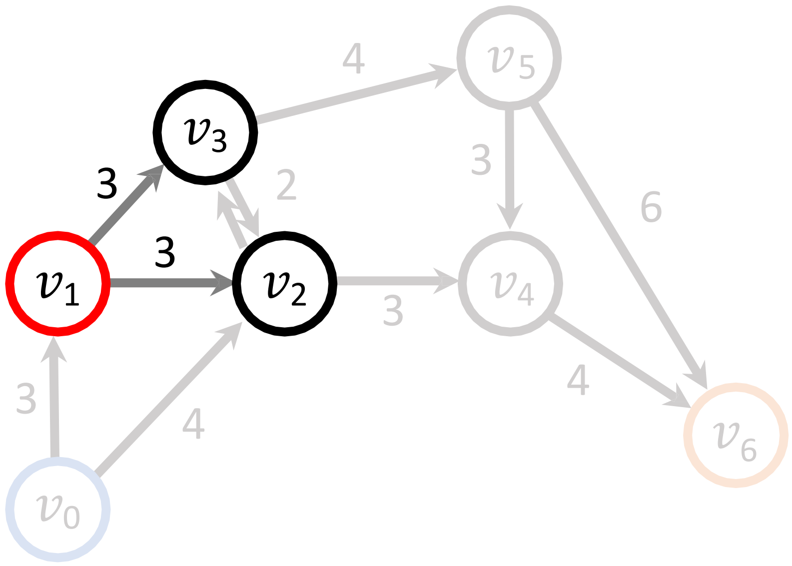
Step 1:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	∞	∞	∞	∞
p[v]	-	v_0	v_0	-	-	-	-
c[v]	✓	-	-	-	-	-	-

Priority queue/ accumulated distance:

v	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	3	4	∞	∞	∞	∞

Global Planning – Dijkstra's Algorithm



d = distance
P = predecessor
c = considered

Step 2:

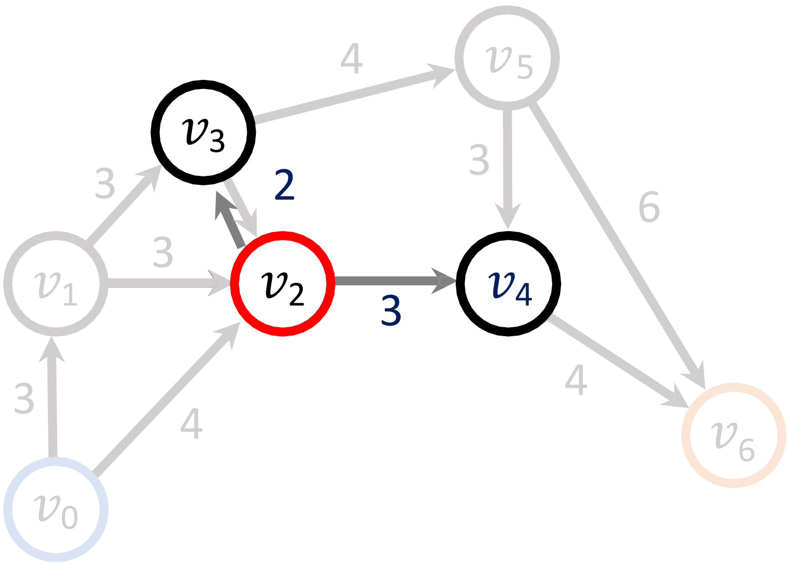
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	∞	∞	∞
p[v]	-	v_0	v_0	v_1	-	-	-
c[v]	✓	✓	-	-	-	-	-

Priority queue/ accumulated distance:

v	v_2	v_3	v_4	v_5	v_6
d[v]	4	6	∞	∞	∞

3+3 = 6

Global Planning – Dijkstra's Algorithm



d = distance
P = predecessor
c = considered

Step 3:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	∞	∞
p[v]	-	v_0	v_0	v_1	v_2	-	-
c[v]	✓	✓	✓	-	-	-	-

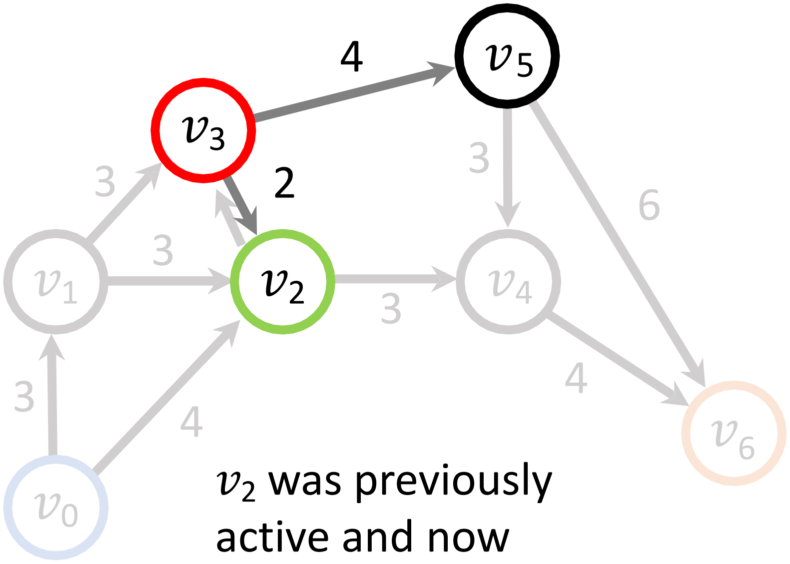
Priority queue/ accumulated distance:

v	v_3	v_4	v_5	v_6
d[v]	6	7	∞	∞

4+2 = 6

Global Planning – Dijkstra's Algorithm

d = distance
P = predecessor
c = considered



v_2 was previously active and now removed from G

Step 4:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	∞
p[v]	-	v_0	v_0	v_1	v_2	v_3	-
c[v]	✓	✓	✓	✓	-	-	-

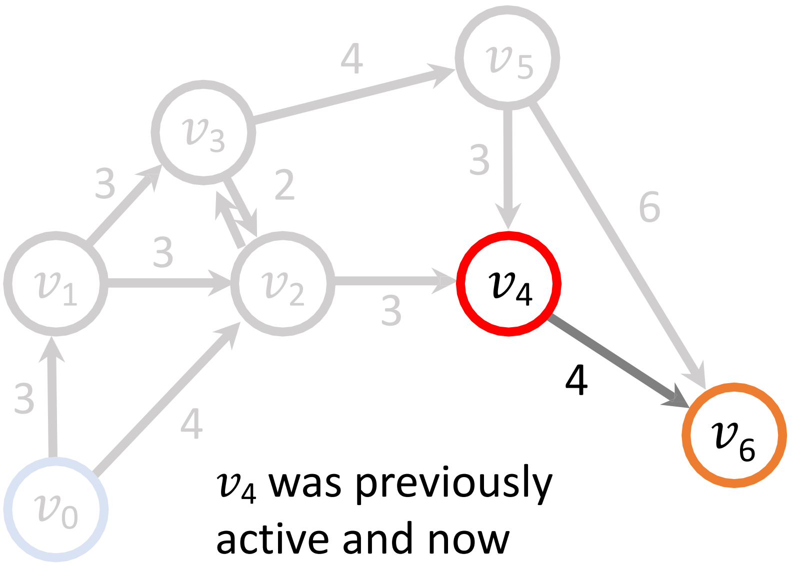
Priority queue/ accumulated distance:

v	v_4	v_5	v_6
d[v]	7	10	∞

6+4 = 10

Global Planning – Dijkstra's Algorithm

d = distance
P = predecessor
c = considered



v_4 was previously active and now removed from G

Step 5:

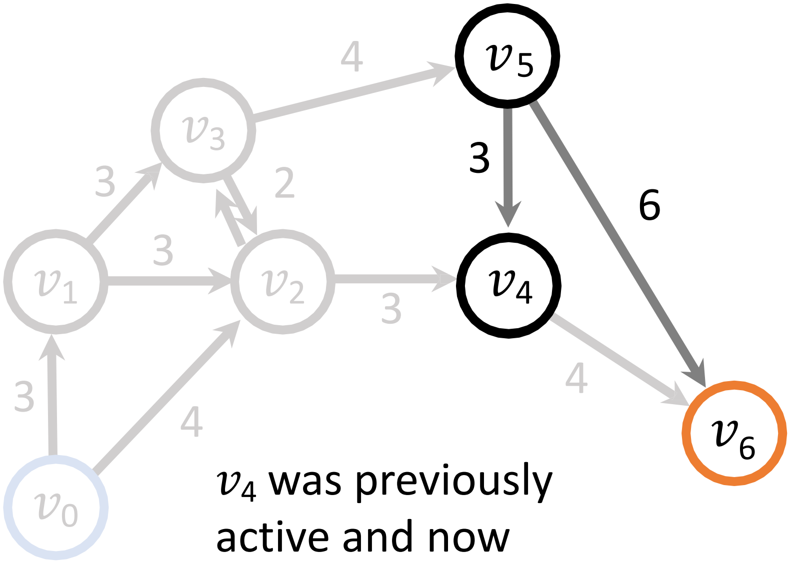
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
p[v]	-	v_0	v_0	v_1	v_2	v_3	v_4
c[v]	✓	✓	✓	✓	✓	-	-

Priority queue/ accumulated distance:

v	v_5	v_6
d[v]	10	11

7+4 = 11

Global Planning – Dijkstra's Algorithm



v_4 was previously active and now removed from G

d = distance
P = predecessor
c = considered

Step 6:

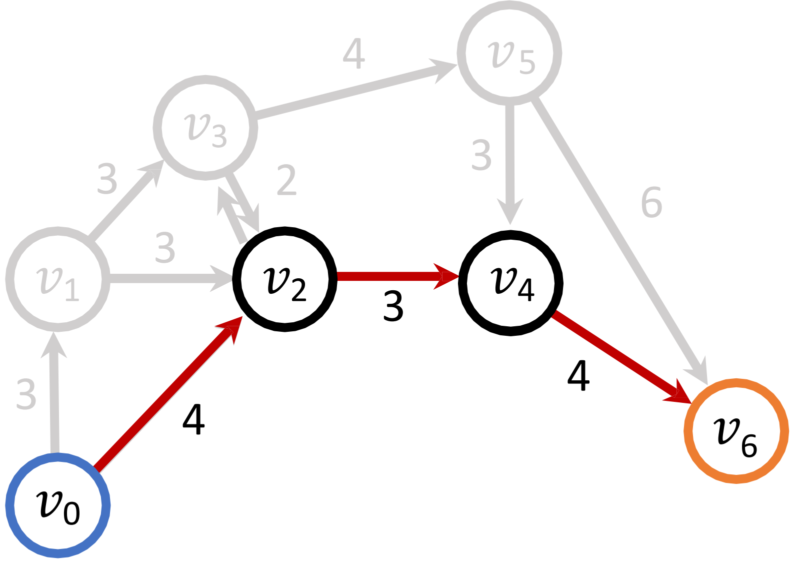
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
p[v]	-	v_0	v_0	v_1	v_2	v_3	v_4
c[v]	✓	✓	✓	✓	✓	✓	-

Priority queue/ accumulated distance:

v	v_6
d[v]	11

10+6 = 16

Global Planning – Dijkstra's Algorithm



d = distance
P = predecessor
c = considered

Step 7:

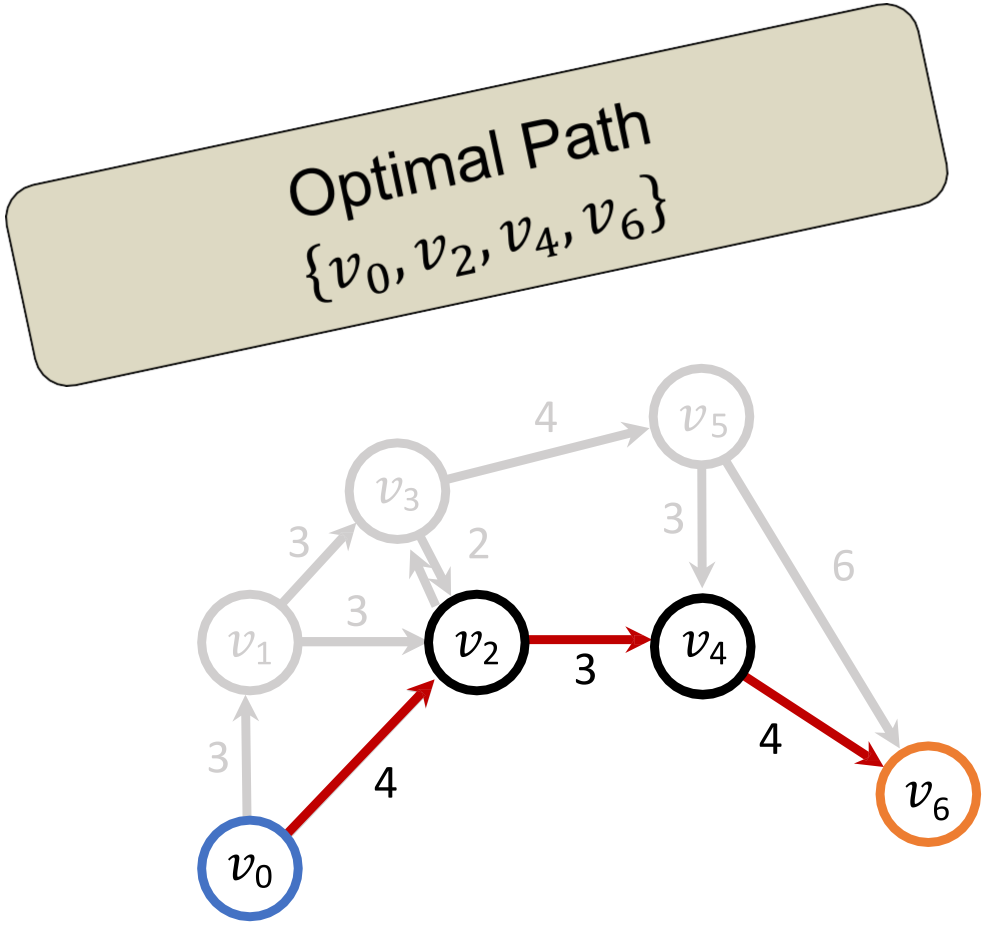
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
p[v]	-	v_0	v_0	v_1	v_2	v_3	v_4
c[v]	✓	✓	✓	✓	✓	✓	✓

Priority queue/ accumulated distance:

v	-
d[v]	-

$G = \emptyset$

Global Planning – Dijkstra's Algorithm



d = distance
P = predecessor
c = considered

Step 7:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
d[v]	0	3	4	6	7	10	11
p[v]	-	v_0	v_0	v_1	v_2	v_3	v_4
c[v]	✓	✓	✓	✓	✓	✓	✓

Priority queue/ accumulated distance:

v	-
d[v]	-

$G = \emptyset$

Dijkstra Visualization, University of San Francisco

You can try: https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

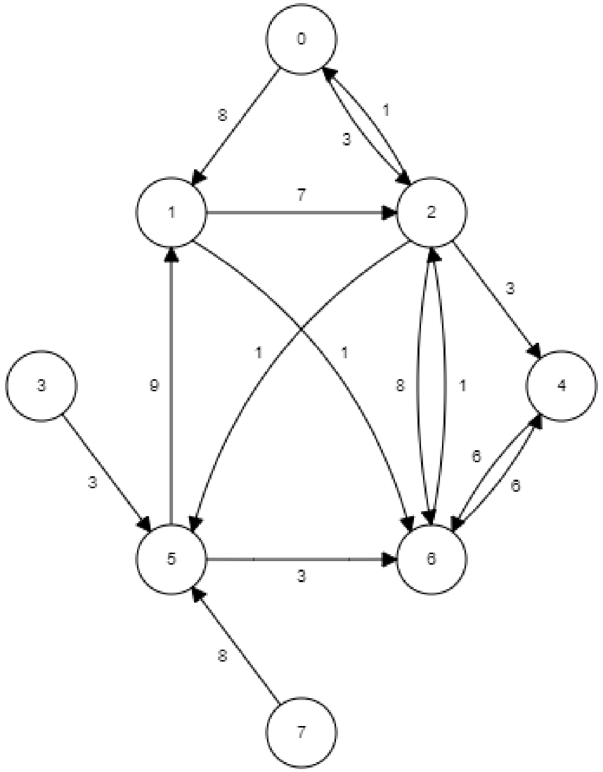
Dijkstra Shortest Path

Start Vertex: 5 | Run Dijkstra | New Graph

- (●) Directed Graph
- ( ) Undirected Graph
- (●) Small Graph
- ( ) Large Graph
- (●) Logical Representation
- ( ) Adjacency List Representation
- ( ) Adjacency Matrix Representation

antecessor

Vertex	Known	Cost	Path	
0	T	5	2	5 6 2 0
1	T	9	5	5 1
2	T	4	6	5 6 2
3	F	INF	-1	No Path
4	T	7	2	5 6 2 4
5	T	0	-1	5
6	T	3	5	5 6
7	F	INF	-1	No Path



Global Planning – A* Algorithm

- Invented by Peter Hart, Nils Nilsson and Bertram Raphael in 1968
- In literature frequently described as extension of Dijkstra's algorithm
- Informed, complete and optimal path search algorithm

- **Differences to Dijkstra's algorithm**
  - Informed heuristic search algorithm
  - An estimation function accelerates the search process
  - Node costs = distance to the start node + the estimated distance to the destination node
  - Node with lowest overall costs has **priority**

- **Estimation function**
  - Free choice of the estimation function
  - Estimator must never overestimate the cost of a route

P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths,"in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

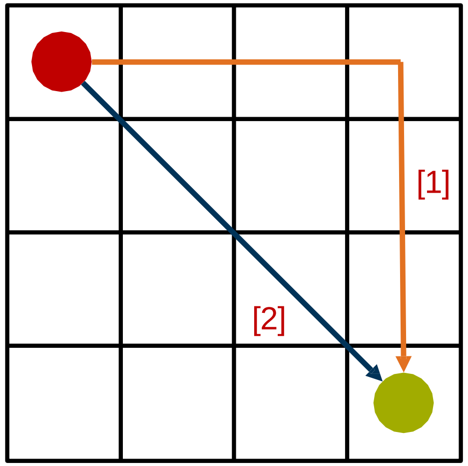
Global Planning – A* Algorithm

❖ **Cost estimation heuristics**

❑ Exact heuristics → time consuming

❑ Approximation heuristics:

- **The Manhattan Distance Heuristic [1]**
- **Euclidean Distance [2]**



❖ **Cost estimation function**

$$f(v) = g(v) + h(v)$$

$f(v)$: total estimated cost of path through node v

$g(v)$: cost so far to reach v

$h(v)$: estimated cost from v to destination

Global Planning – A* Algorithm

❖ Steps

Create open-list and closed list

Add starting node to open-list

Repeat algorithm until destination is found

1. Calculation of the temporary distances of the active node:
   i. Consideration of all neighbor nodes
   ii. Summing up its distance with the weights of the edges plus estimated distance to destination
2. If calculated distance is smaller than current:
   i. Update the distance
   ii. Set the current node as antecessor
   iii. Add node to open-list
3. Add current node to closed-list
4. Proceed with minimal temporary distance node as active node

Pseudocode:

```
1:  make open-list with starting node
2:  make empty closed-list
3:  while destination not reached:
4:      consider node with min f[v]
5:      for each child u of current node v:
6:        set child costs f[u]
7:        if child u is in open-list:
8:          if g[u] < g[v]:
9:            continue to line 17
10:       else if child u is in closed-list:
11:         if g[u] < g[v]:
12:           add u to open-list
13:       else:
15:         add u to open-list
16:     Set g[u] = successor current cost
17:     Set prev u = v
18:     add v to closed-list
```

Global Planning – A* Algorithm

❖ Steps

Create open-list and closed list

Add starting node to open-list

Repeat algorithm until destination is found

1. Calculation of the temporary distances of the active node:
   i. Consideration of all neighbor nodes
   ii. Summing up its distance with the ... edges plus estimated dista...
2. If calculated distance ...
   i. Update the distance
   ii. Set the current node as ...
   iii. Add node to open-list
3. Add current node to closed-list
4. Proceed with minimal temporary distance node as active node

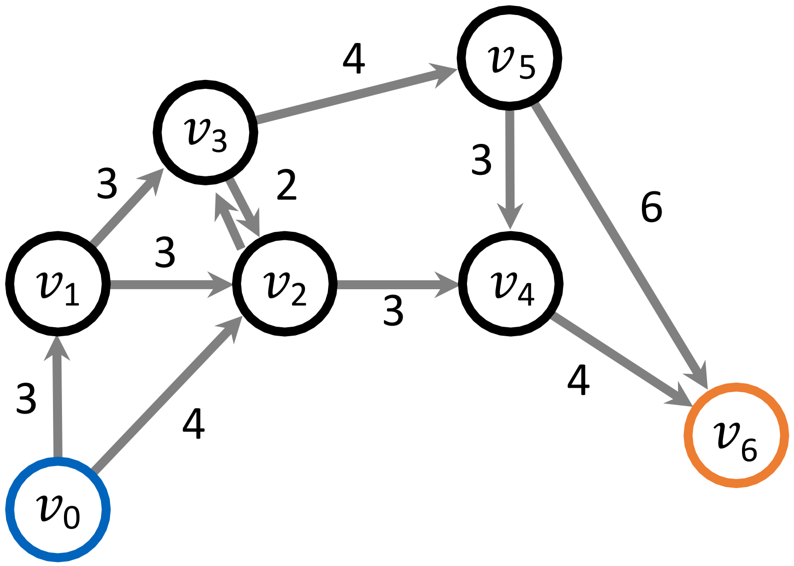
Compare Lower Bound Estimate:
Euclidean Distance

Pseudocode:

```
1:  make open-list with starting node
2:  make empty closed-list
3:  while ...ination not reached:
4:  ... node with min f[v]
    ...child u of current node v:
    ...costs f[u]
    ... is in open-list:
    ...if g[u]  < g[v]:
            continue to line 17
10:     else if child u is in closed-list:
11:         if g[u]  < g[v]:
12:             add u to open-list
13:     else:
15:         add u to open-list
16:     Set g[u] = successor current cost
17:     Set prev u  = v
18:   add v to closed-list
```

Global Planning – A* Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



Step 0:

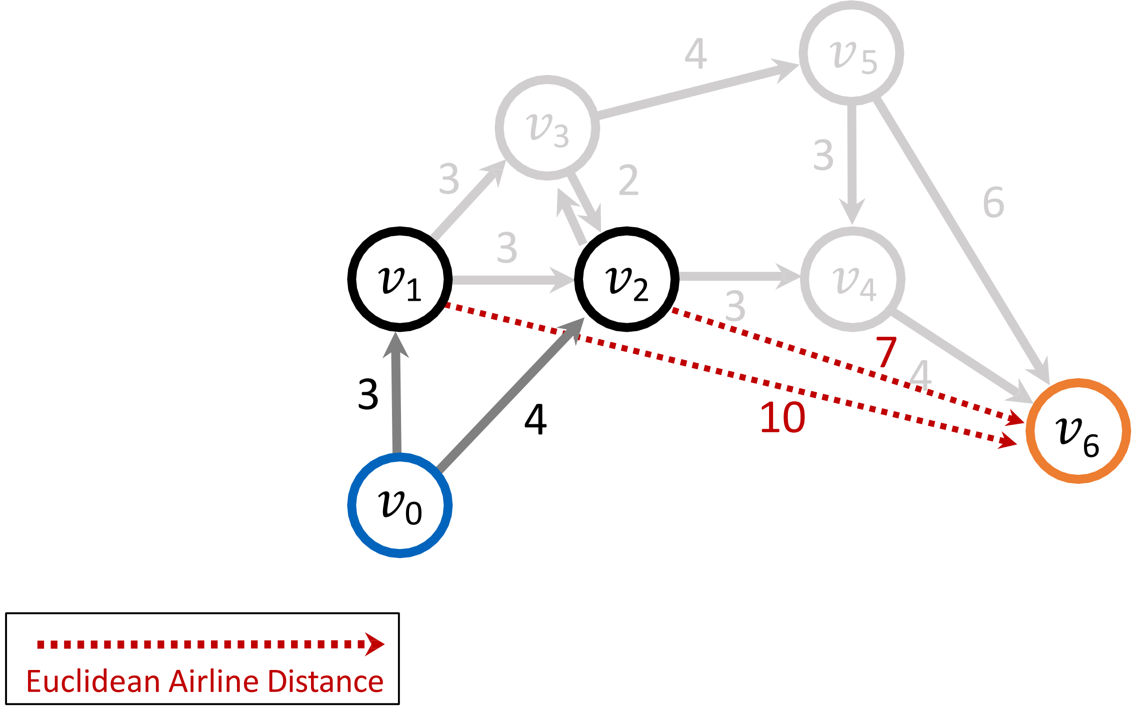
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	∞	∞	∞	∞	∞	∞
h[v]	-	-	-	-	-	-	-
f[v]	-	-	-	-	-	-	-
p[v]	-	-	-	-	-	-	-
c[v]	-	-	-	-	-	-	-

Priority Queue:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	0	∞	∞	∞	∞	∞	∞

Global Planning – A* Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



Step 0:

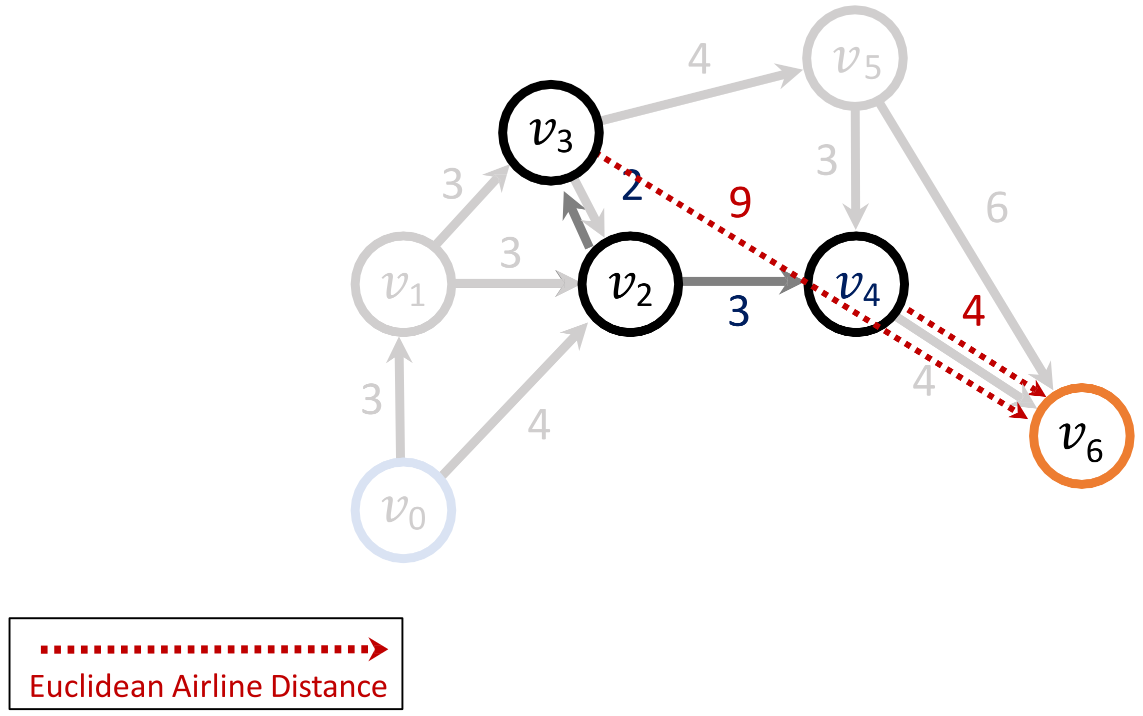
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	∞	∞	∞	∞
h[v]	10	10	7	-	-	-	-
f[v]	10	13	11	-	-	-	-
p[v]	-	v_0	v_0	-	-	-	-
c[v]	✓	-	-	-	-	-	-

Priority Queue:

v	v_1	v_2	v_3	v_4	v_5	v_6
f[v]	13	11	∞	∞	∞	∞

Global Planning – A* Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



Step 0:

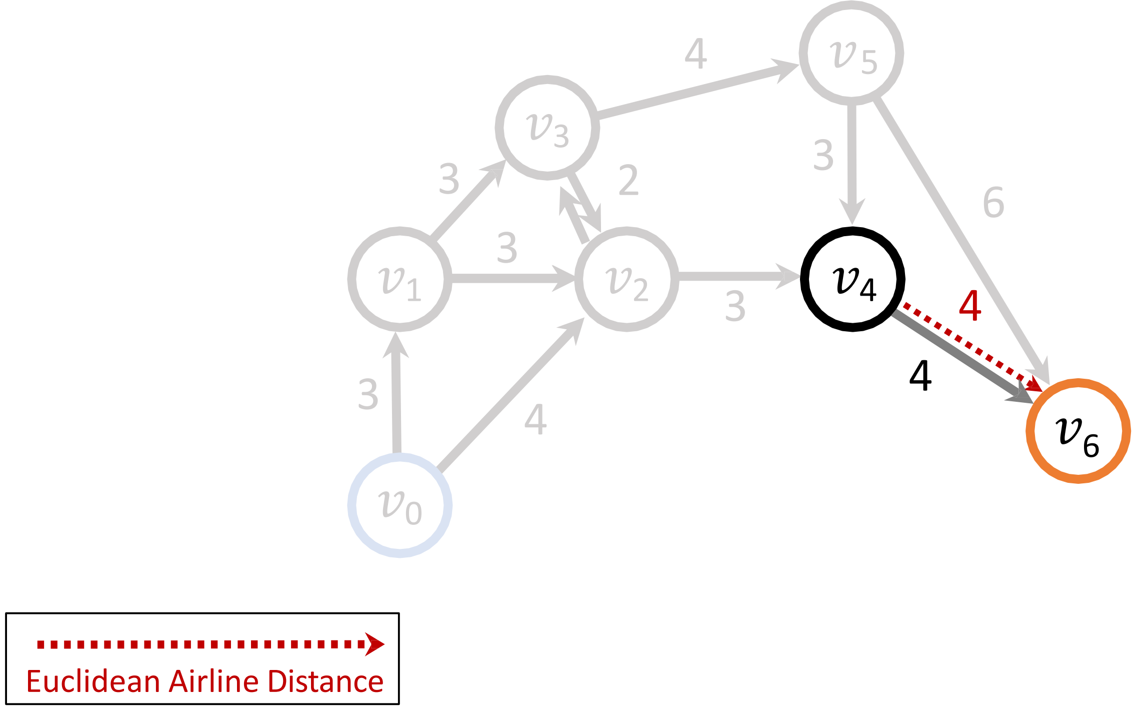
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	6	7	∞	∞
h[v]	10	10	7	9	4	-	-
f[v]	10	13	11	15	11	-	-
p[v]	-	v_0	v_0	v_2	v_2	-	-
c[v]	✓	-	✓	-	-	-	-

Priority Queue:

v	v_1	v_3	v_4	v_5	v_6
f[v]	13	15	11	∞	∞

Global Planning – A* Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



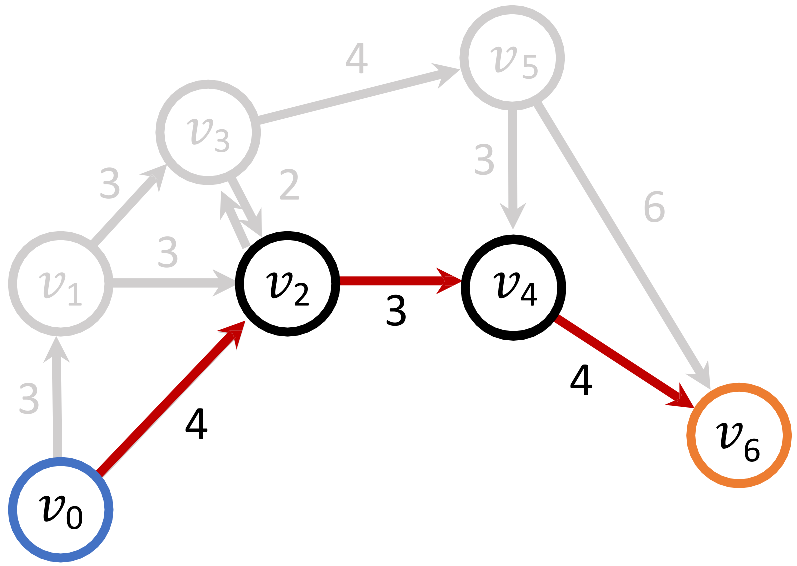
Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	6	7	∞	11
h[v]	10	10	7	9	4	-	0
f[v]	10	13	11	15	11	-	11
p[v]	-	v_0	v_0	v_2	v_2	-	v_4
c[v]	✓	-	✓	-	✓	-	-

Priority Queue:

v	v_1	v_3	v_5	v_6
f[v]	13	15	∞	11

Global Planning – A* Algorithm



g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered

Step 0:

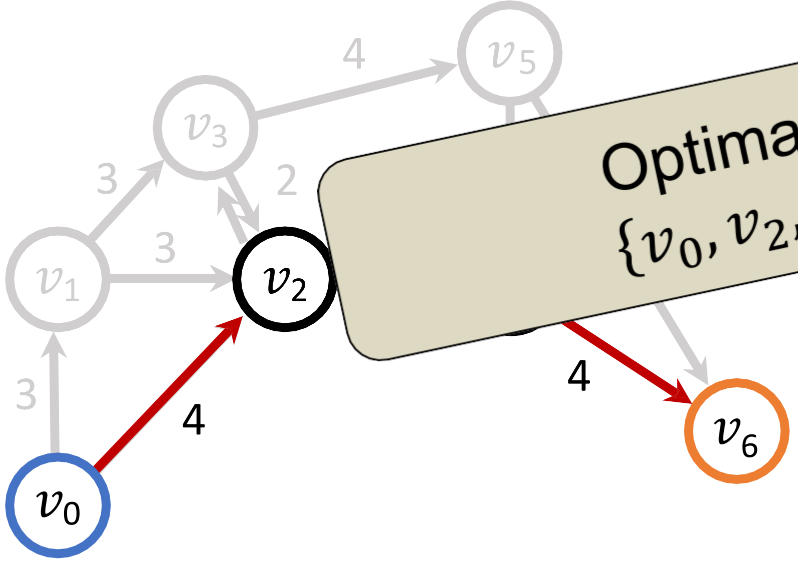
v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]	0	3	4	6	7	∞	11
h[v]	10	10	7	9	4	-	0
f[v]	10	13	11	15	11	-	11
p[v]	-	v_0	v_0	v_2	v_2	-	v_4
c[v]	✓	-	✓	-	✓	-	-

Priority Queue:

v	v_1	v_3	v_5	v_6
f[v]	13	15	∞	11

Global Planning – A* Algorithm

g=distance so far
h=v to destination
f=total estimated cost
p=predecessor
c=considered



Optimal Path
$\{v_0, v_2, v_4, v_6\}$

Step 0:

v	v_0	v_1	v_2	v_3	v_4	v_5	v_6
g[v]			4	6	7	∞	11
			7	9	4	-	0
			11	15	11	-	11
p[v]	-	v_0	v_0	v_2	v_2	-	v_4
c[v]	✓	-	✓	-	✓	-	-

Priority Queue:

In order to always obtain an **optimal path**, **all nodes must be visited**, or **heuristic must not overestimate the cost to reach goal**

Global Planning – Dijkstra’s vs. A* Algorithm

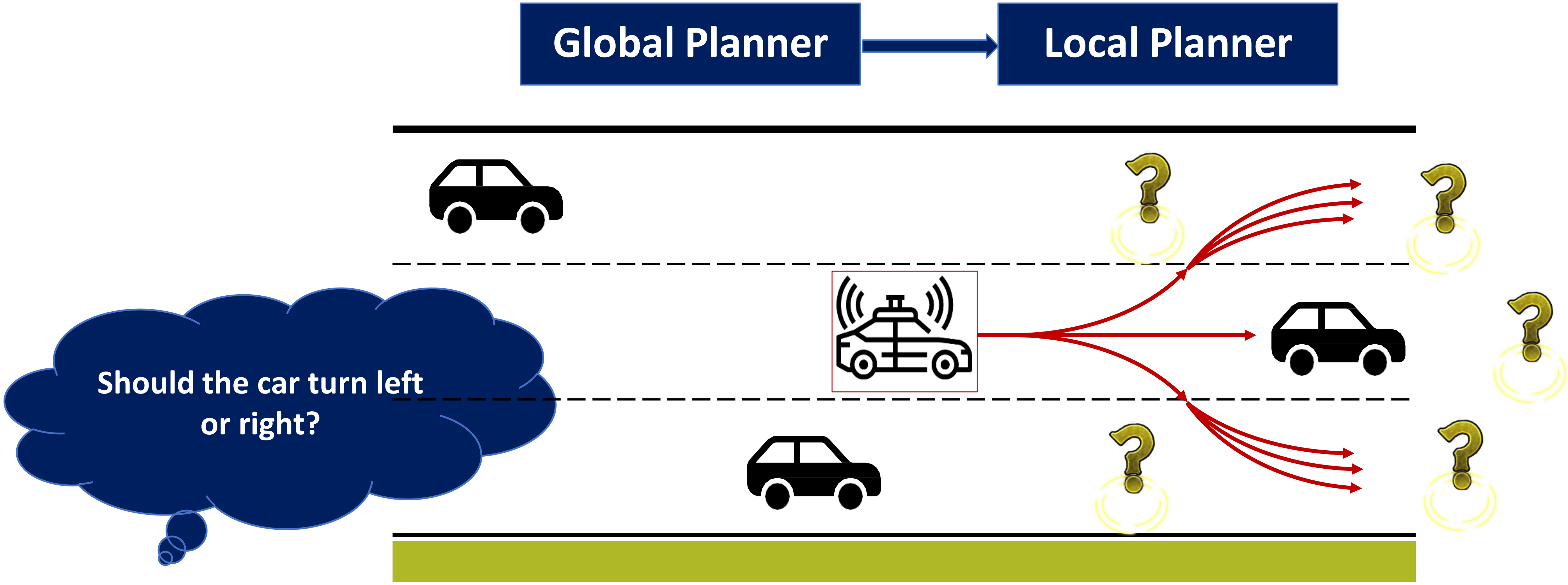
- A* converges much **faster** toward the destination

- A* explores **all partial paths** in the order of their potential to reach the destination with a **minimum** amount of steps

- A* continues to explore the graph even **if a feasible path has been found**, because – unlike Dijkstra – it can not be sure that the first feasible path is an optimal path

Behavior Planning

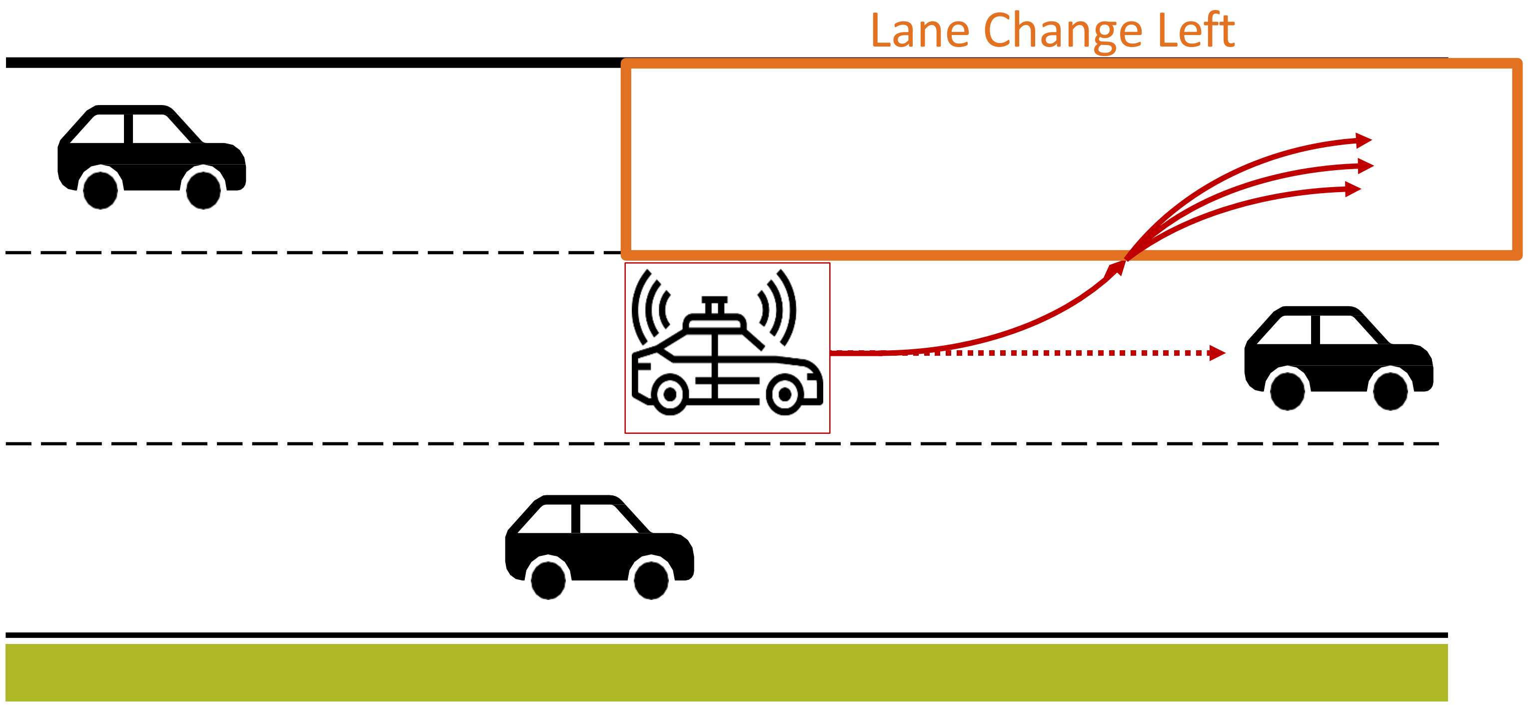
- ❖ Fills technical gap between Global Planner and Local Planner
- ❖ Considers rules of the road and static/dynamic objects around the vehicle



Behavior Planning

❖ Behavior planning plans high-level driving actions to safely achieve the driving mission under various driving situations

Global Planner → Behavior Planner → Local Planner



Behavior Planning

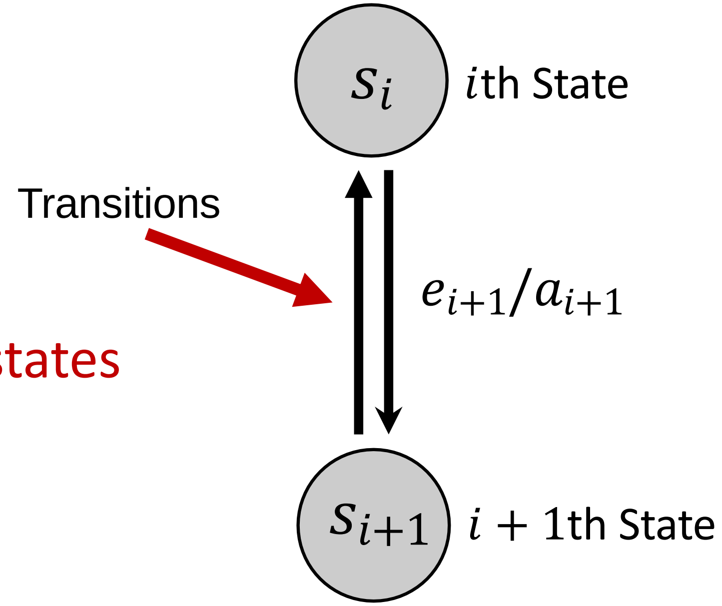
- **Decision of behavior can be determined by costs**
  - Feasibility costs, security costs, legality costs, comfort costs, speed costs, etc.

- **Possible constraints**
  - Global objective, road speed limit, road lane boundaries, stop locations, set of interest vehicles, etc.

Finite-state Machine (FSM)

- Behavior of the vehicle can be modeled by **Finite-state Machines**

- **Finite-state Machine (FSM)**
  - Mathematical model of computation
  - Consists of finite number of states
  - Behavior of the vehicle can be modeled by transitions between states
  - Transitions based on current state and given input
  - Deterministic behavior

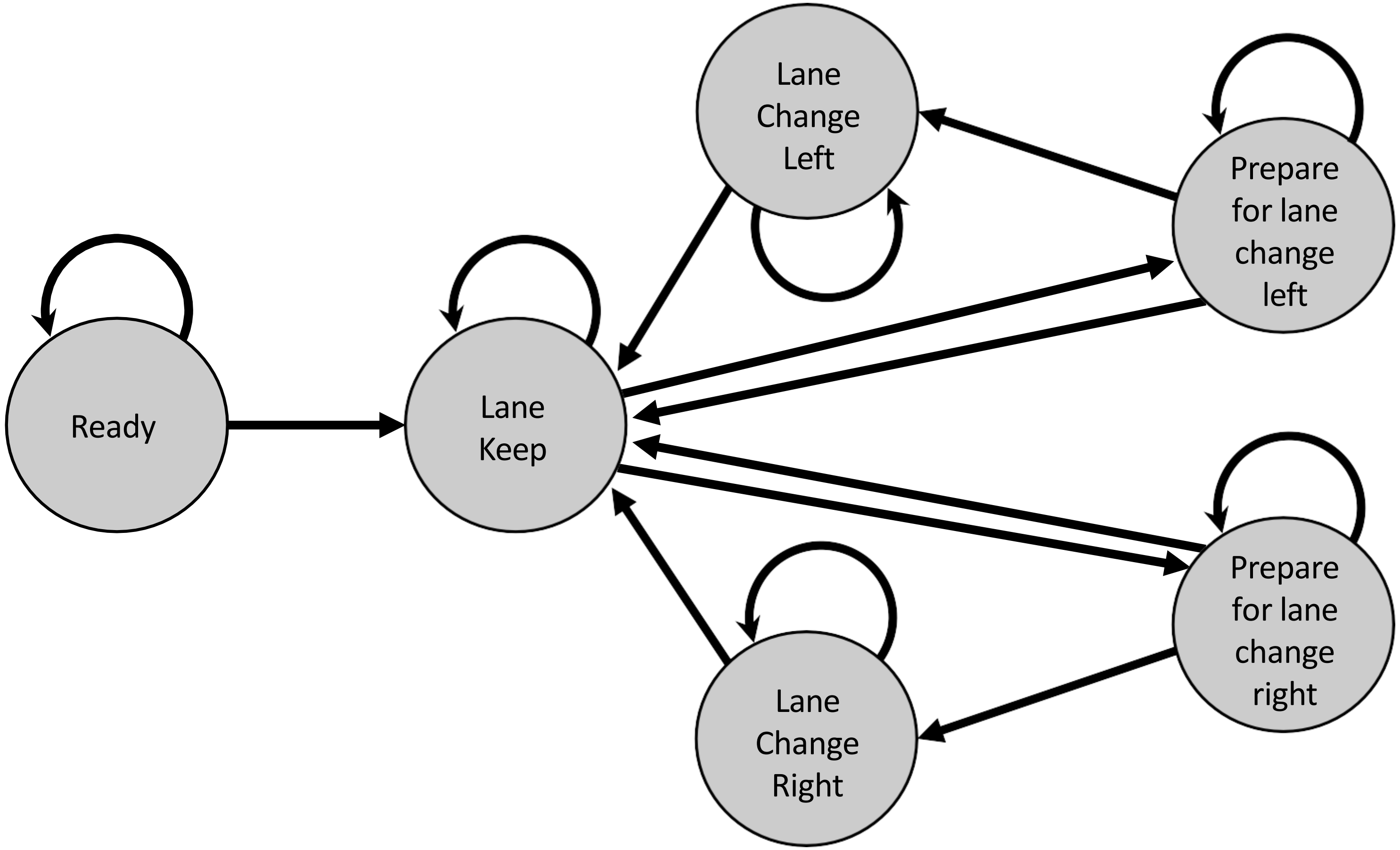


- **Predictions**
  - There is exactly one state per timestep
  - The time delay at the state transition is irrelevant

States s_{i+1}
represent a specific condition or configuration that the machine can be in

Transition functions defines the rules/actions a_i for machine to change from one state to another based on input events e_i

Finite-state Machine (FSM) - An Example



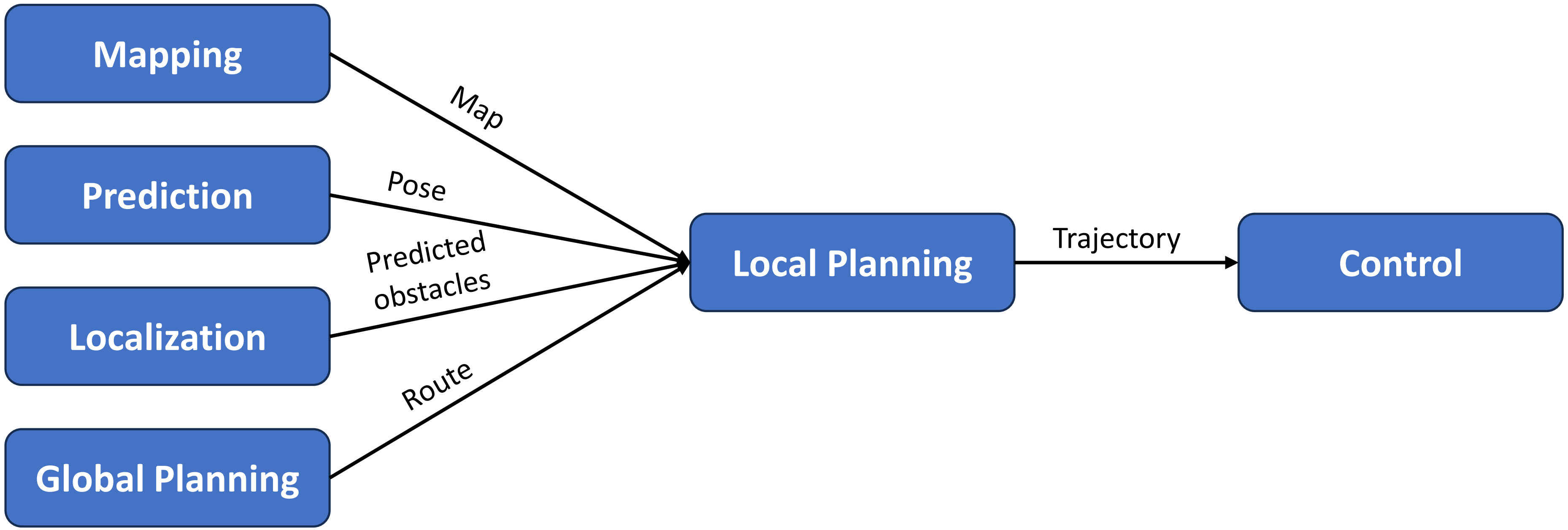
Finite-state Machine (FSM)

- **Advantages**
  - Limiting number of rule checks
  - Clear in structure
  - Easy to calibrate / optimize
  - Simple implementation
  - High flexibility
  - Easy determination of reachability of a state

- **Disadvantages**
  - High knowledge of system design required
  - Large FSMs hard to visualize
  - Difficult transferability to other projects

Local Planning

❖The main task of **the local planner** is to generate a **feasible and collision free trajectory** that leads the vehicle in the current **local** environment and **follow the global planning target as good as possible**



Local Planning – Rapidly-exploring random tree (RRT)

❖ Proposed by LaValle in 1998

❖**RRT** a simple, iterative algorithm that quickly searches complicated, high-dimensional spaces for feasible paths

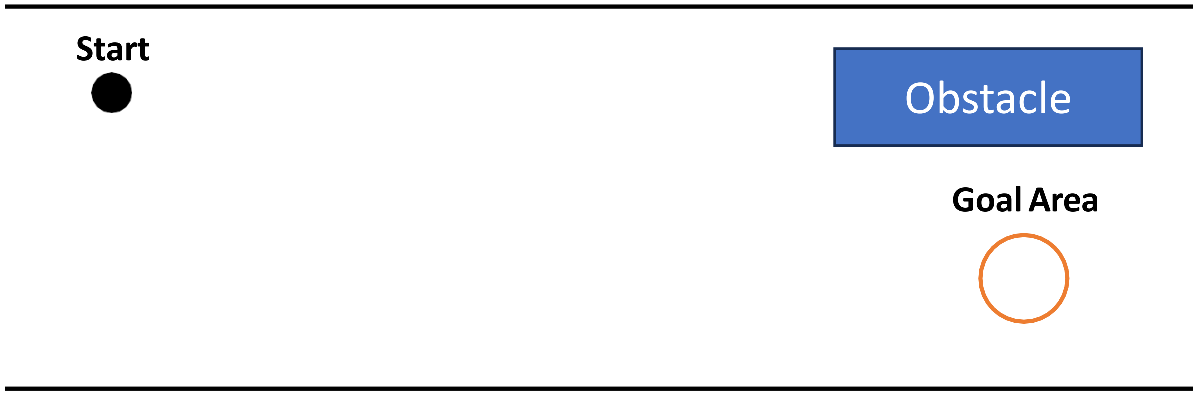
❖ The idea is to incrementally grow a space-filling tree by sampling the space at random and connecting the nearest point in the tree to the new random sample

Credit: https://lavalle.pl/rrtpubs.html

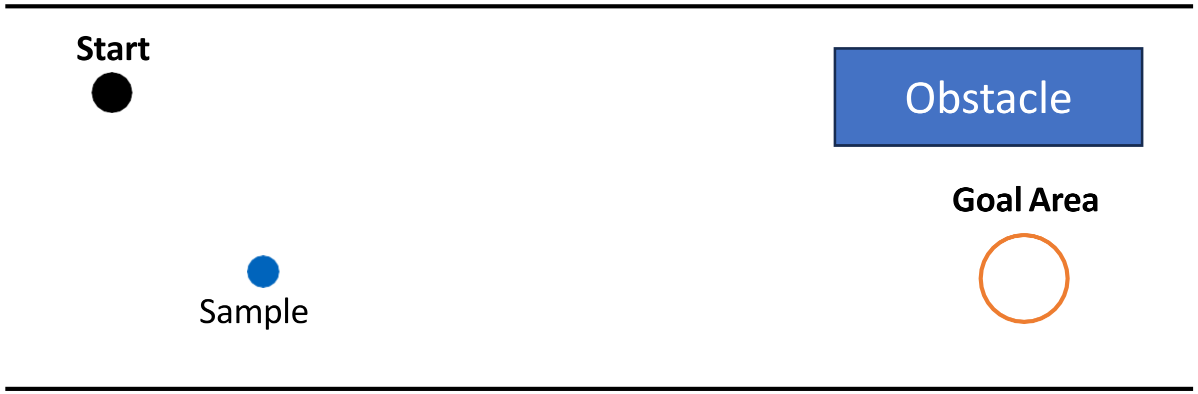
Rapidly-exploring random trees: A new tool for path planning. S. M. LaValle. TR 98-11, Computer Science Dept., Iowa State University, October 1998

Local Planning – Rapidly-exploring random tree (RRT)

❖ **Repeat until goal are is reached:**

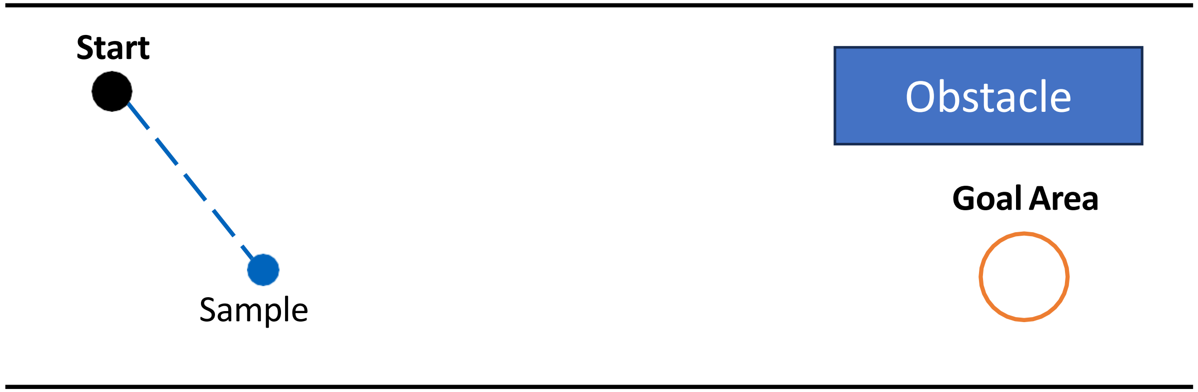


Local Planning – Rapidly-exploring random tree (RRT)



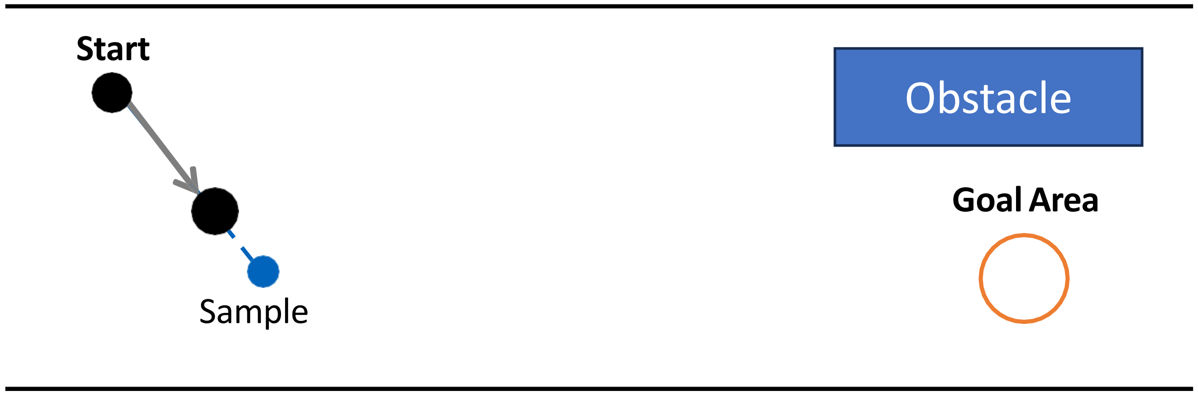
- **Repeat until goal are is reached:**
  1. Sample in configuration space

Local Planning – Rapidly-exploring random tree (RRT)



- **Repeat until goal are is reached:**
  1. Sample in configuration space
  2. Get the nearest node

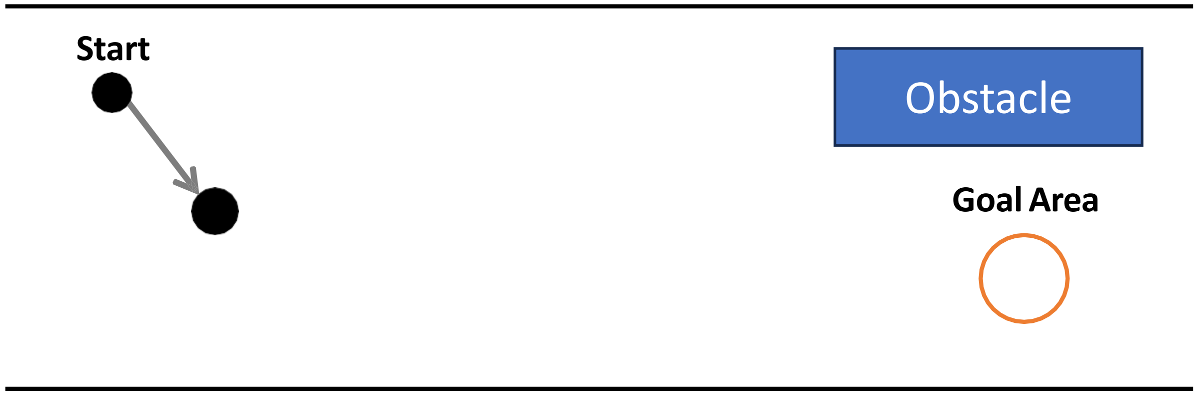
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

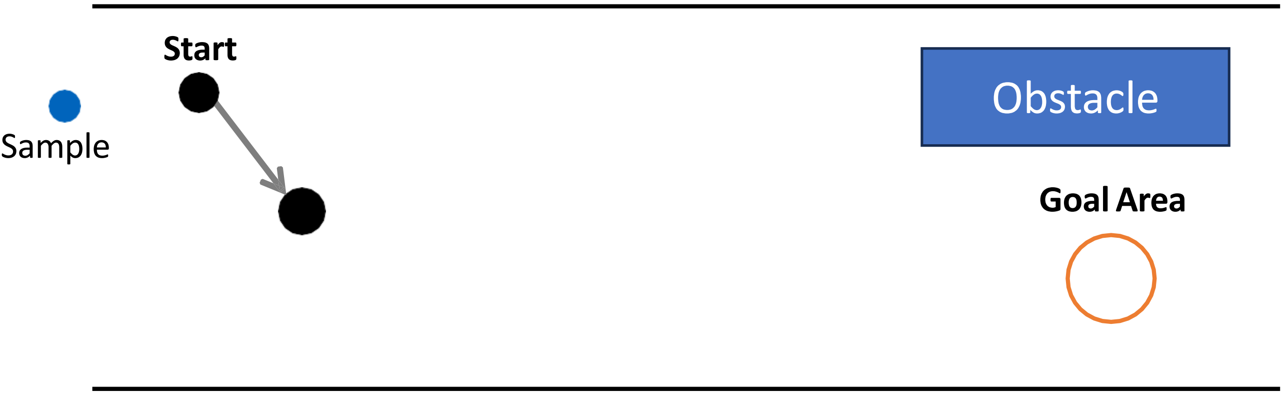
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

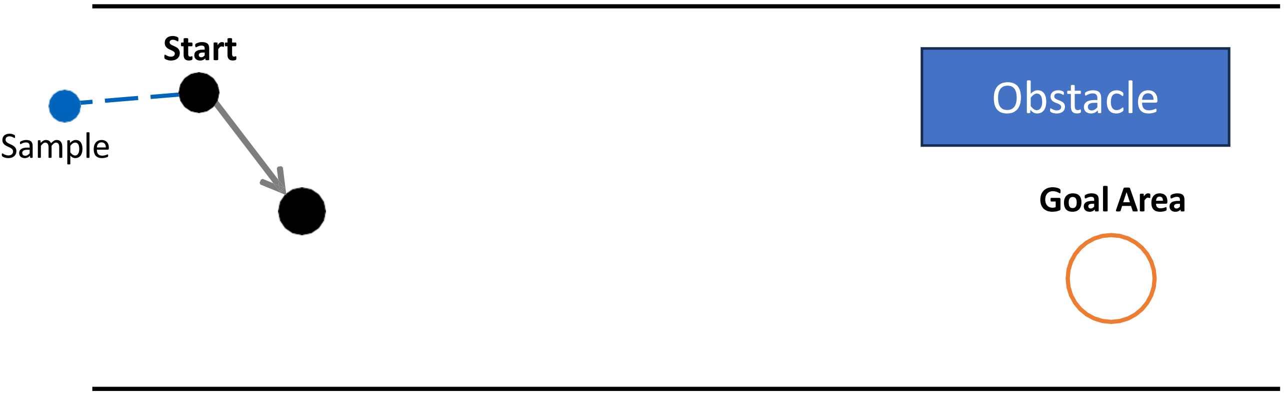
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

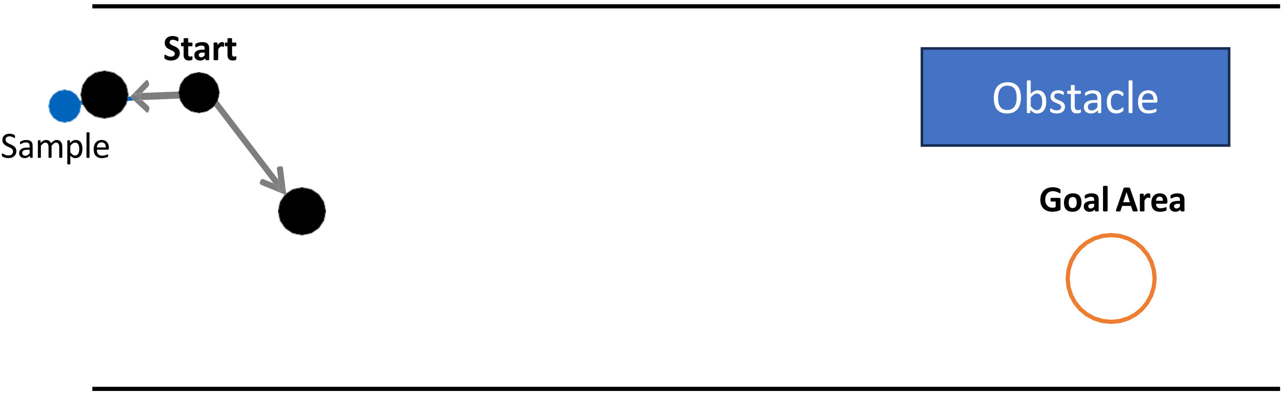
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

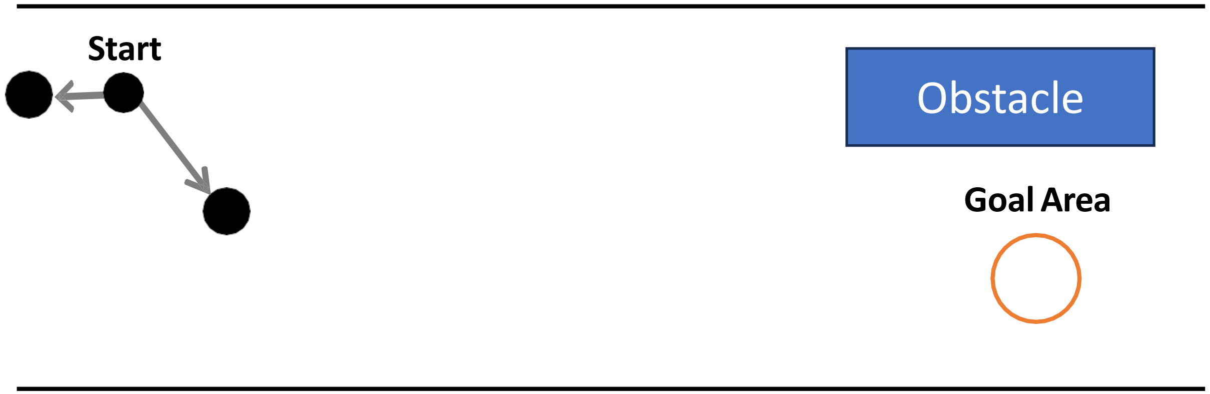
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

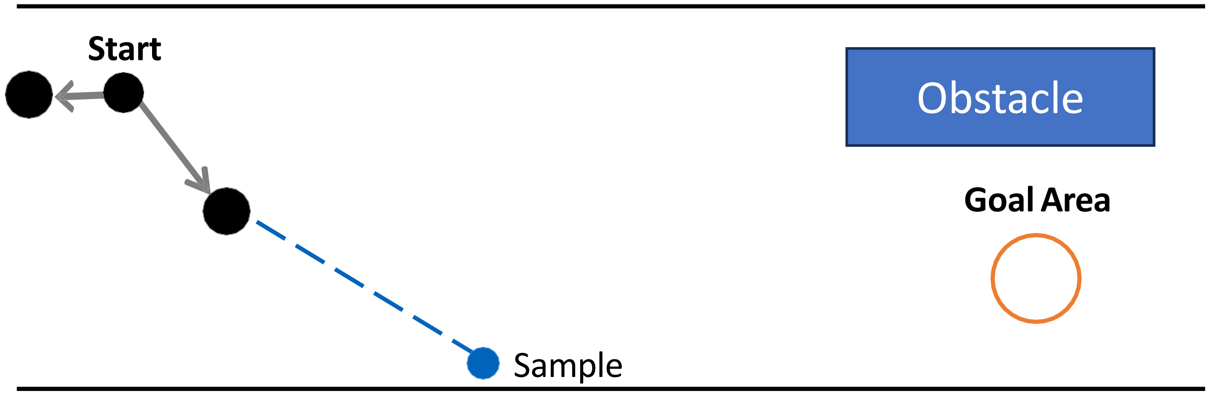
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

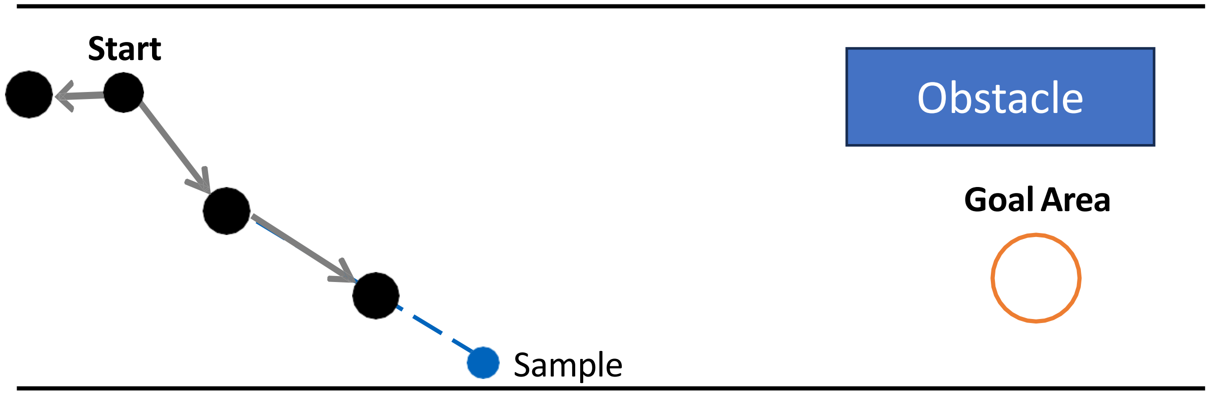
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

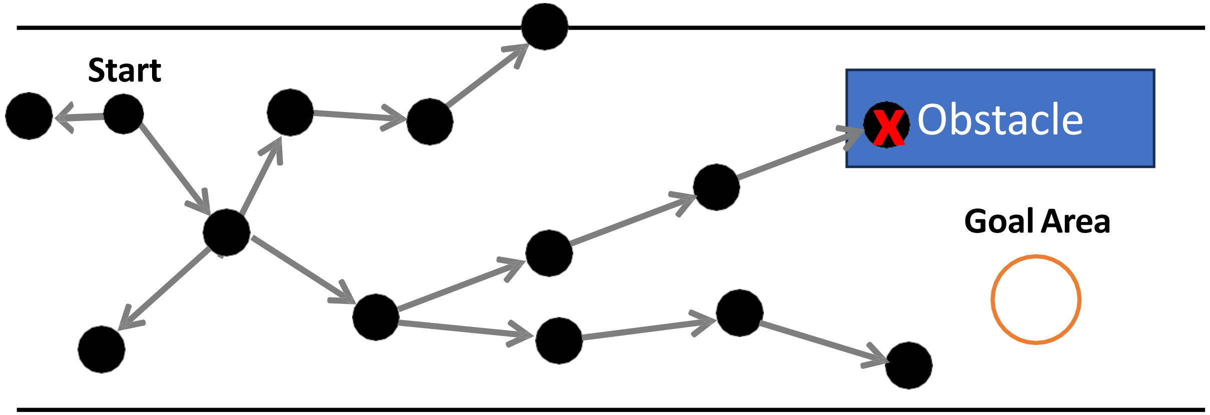
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

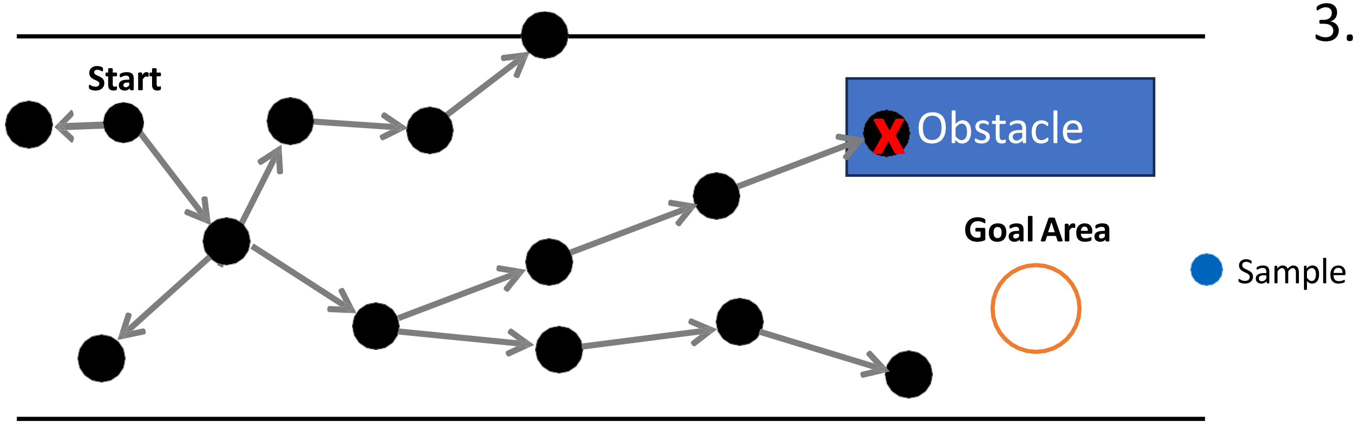
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

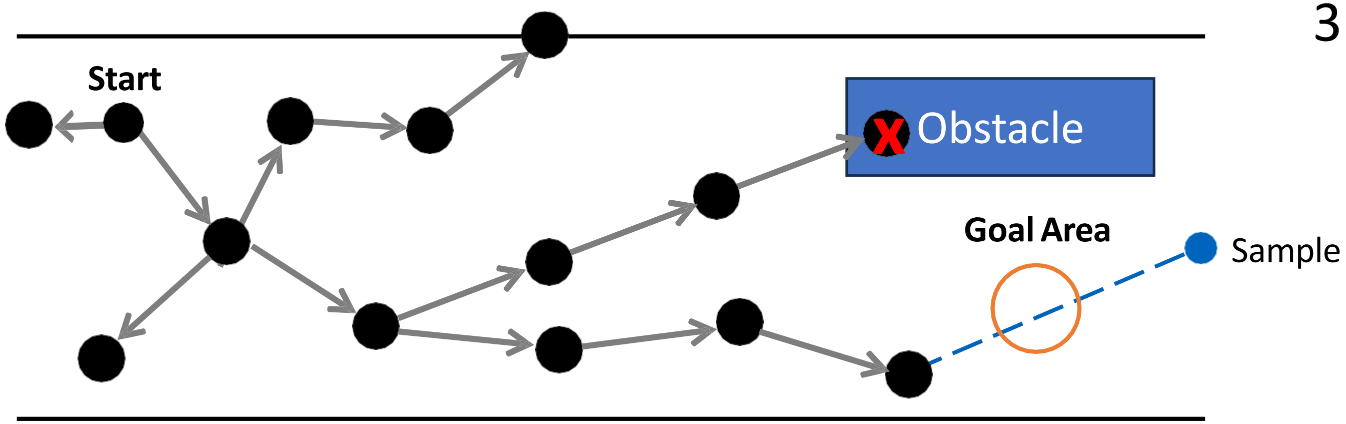
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

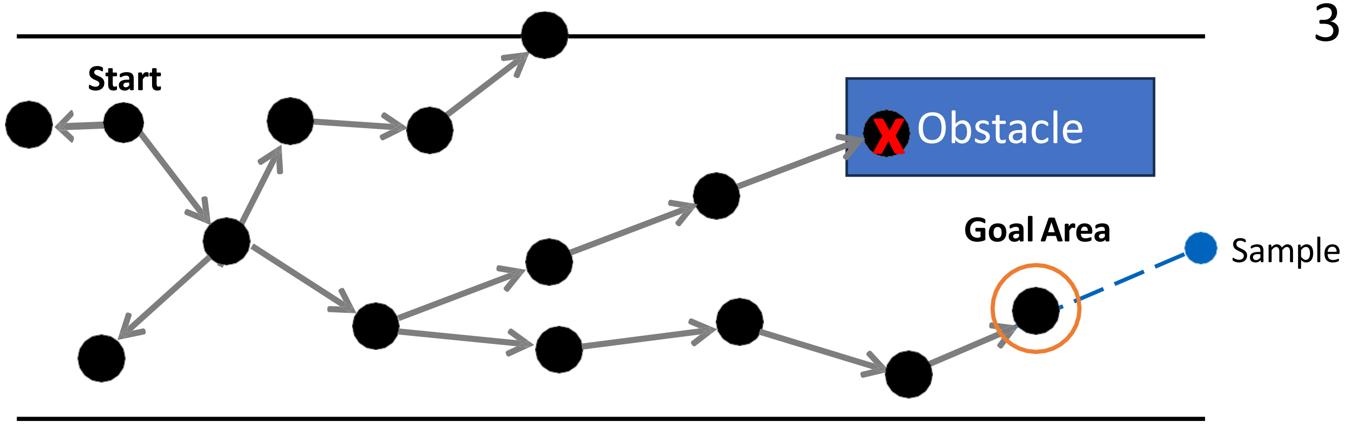
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

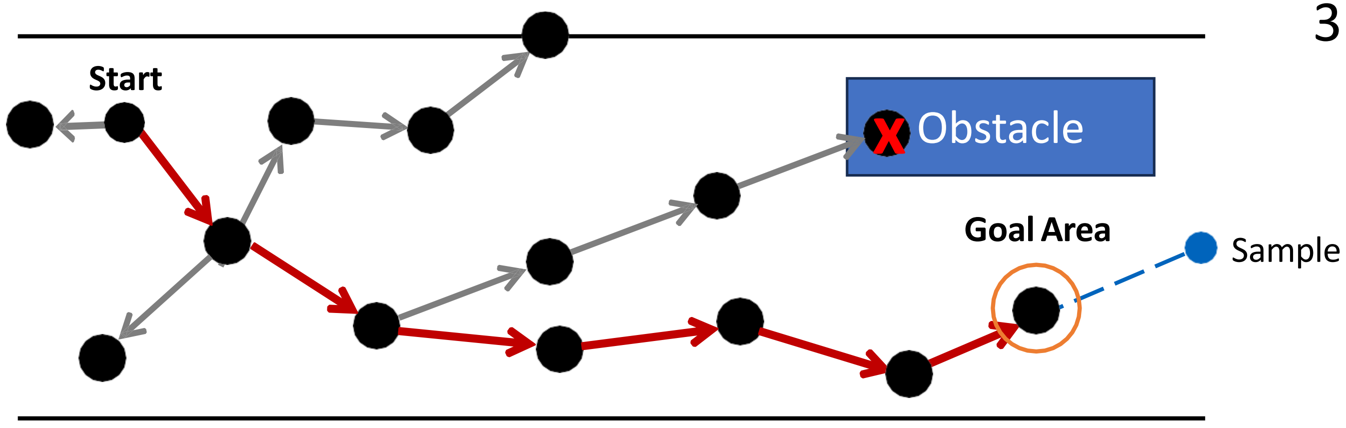
Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

Local Planning – Rapidly-exploring random tree (RRT)



❖ **Repeat until goal are is reached:**

1. Sample in configuration space
2. Get the nearest node
3. From the nearest node, extend tree in direction of sample point (if new node is collision free)

Credit

- ❖ Lectures from CS686: Robot Motion Planning and Applications (KAIST - Fall 2013)
- ❖ Lectures from Autonomous Driving Software Engineering (TUM)

Thank you