Introduction to Autonomous Electric Vehicles

Lecture 5

- Recap
- Planning
- Collision checking
- Tracking

EEE department, BITS-Goa

14-April–2021


UPLOAD


hotstar
specials
OK
COMPUTER

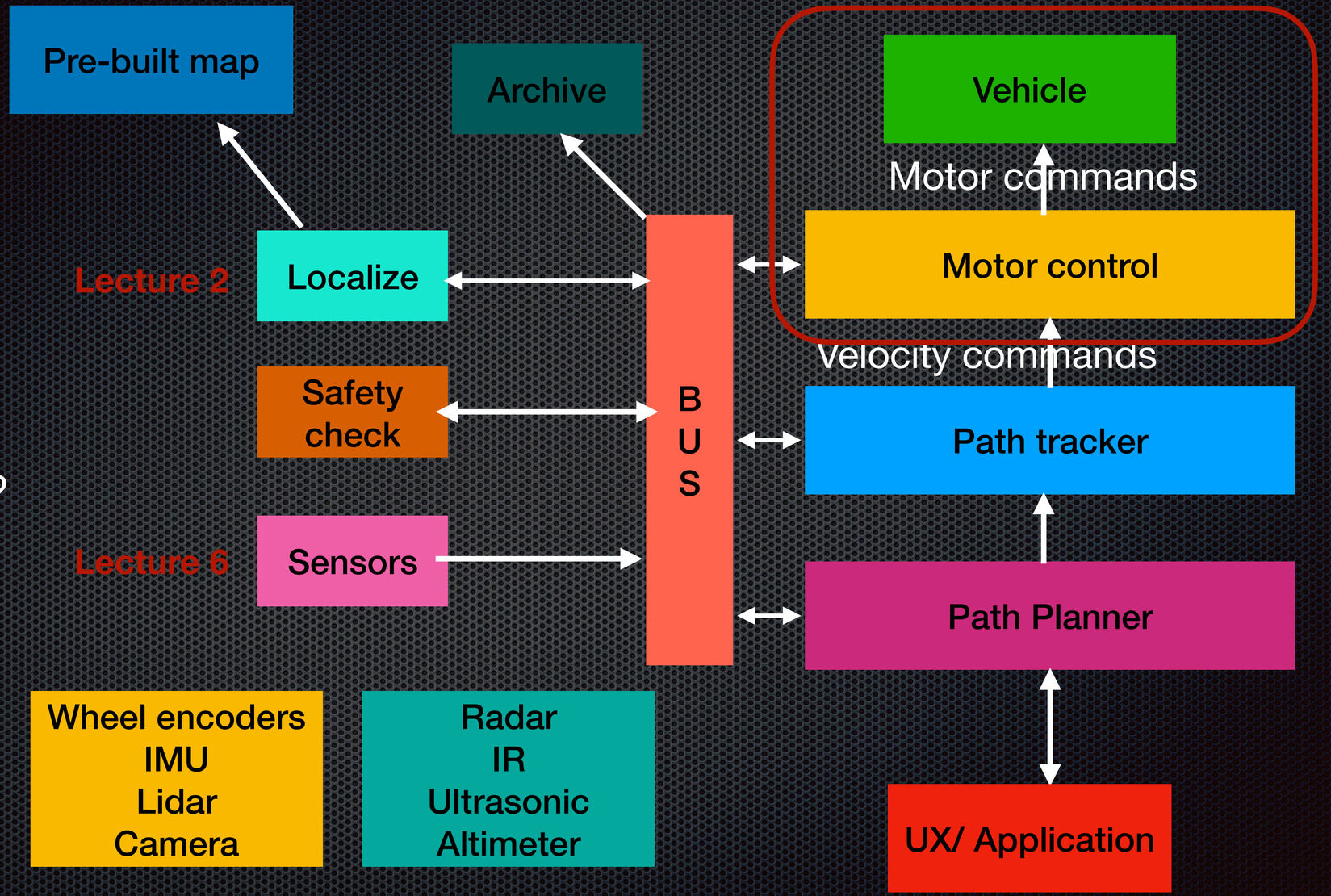
Lecture plan

- Recap
  - Autonomy and control architecture
  - Maps and Planning
  - Ground-truth and Loop closure

- Path Planning
  - Mission planning
    - Graph-based methods: Djikstra/ A*
  - Safe local planning
    - How to check for collisions
    - Lattice planner

- Path Tracking
  - Pure pursuit
  - Dynamic Window Avoidance

Autonomy and Control: Pub-Sub architecture



Iterative procedure

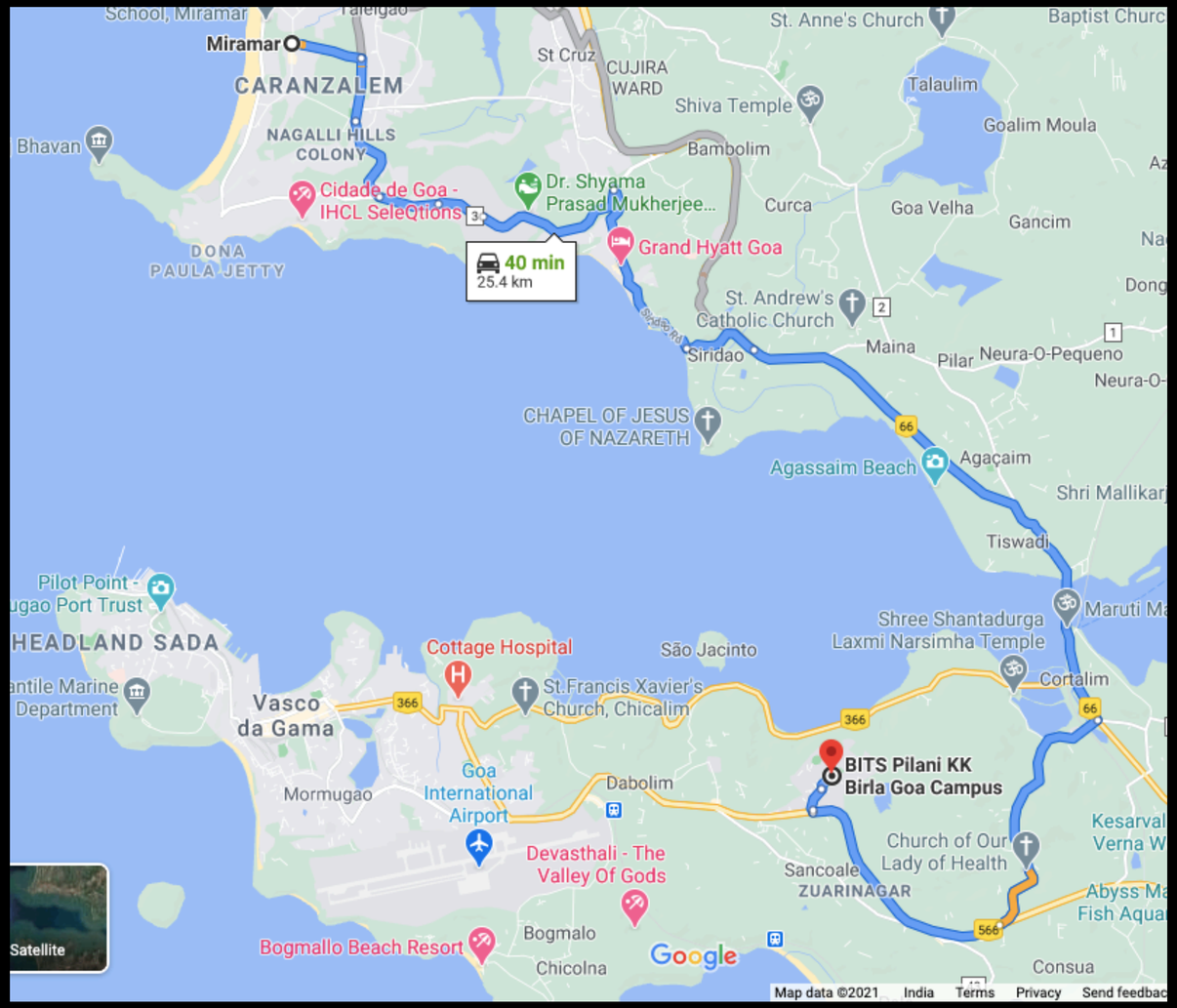
1. Where am I?
2. Where do I go next?
3. What should I do?

Key assumption: Operation in known environments

Recap

Maps and planning

Global planning: Go from campus to Miramar



Useful for High-level estimates

Locally safe planning

Vehicle action in immediate time horizon

- More annotations/ Semantically richer
- Suited for quick Replanning

Probabilistic Occupancy grid

Occupancy probability of grid cell i

$$m^i \in \{0,1\}$$

Belief map

$$bel_t(m^i) = p(m^i \mid z_t)$$

$bel(m^i)$ = occupancy belief of cell i

z_t = sensor measurements

Use multiple measurements to build a robust belief map

Recursive Bayesian update:

$$bel_t(m^i) = \eta bel_{t-1}(m^i) p(z_t \mid m_i)$$

Previous Belief

Measurement model

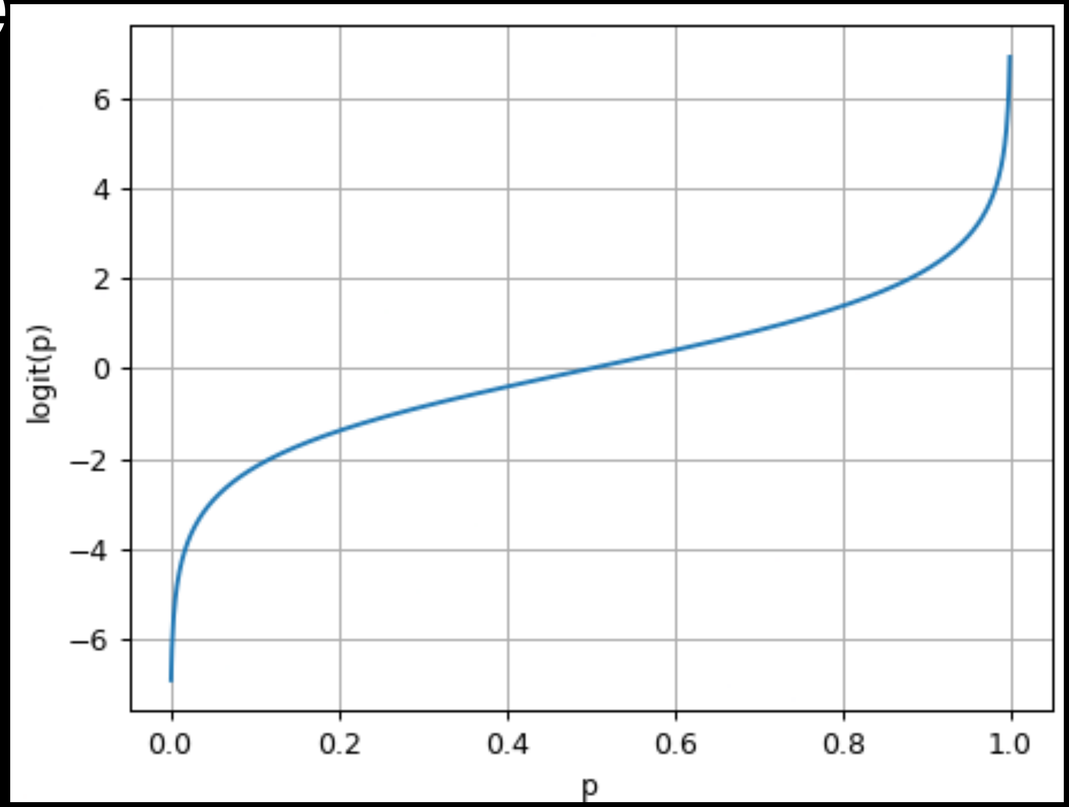
Key assumptions

- Cells are independent
- Static environment
- Robot position is known

Mapping: Log-odds update

- Floating point problems with repeated multiplication
- Move to log-odds (logit) transformation

$$logit(p) = log\left(\frac{p}{1-p}\right)$$

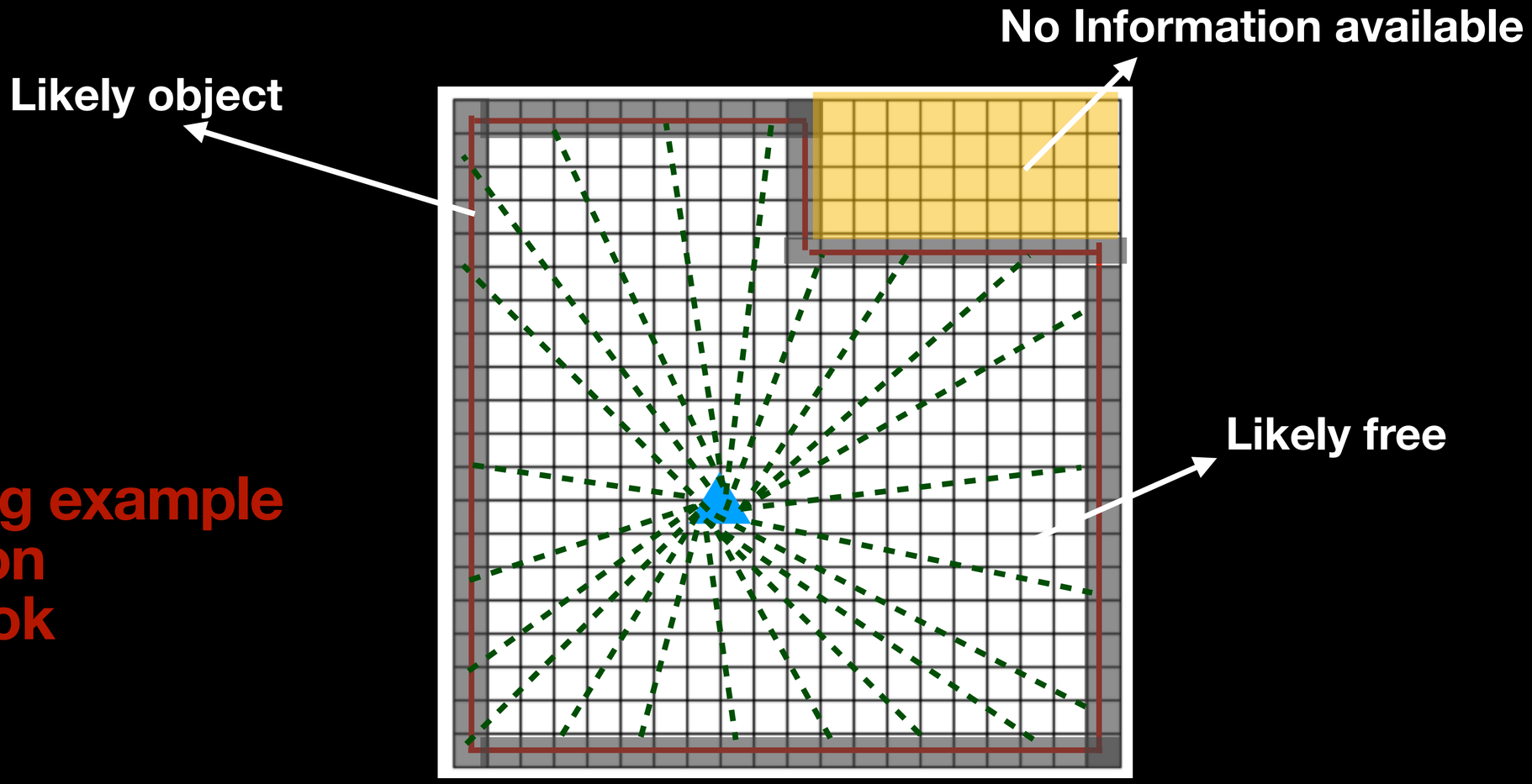


$$l_i(t) = logit(p(m^i \mid z_t)) + l_i(t-1)$$

Log-odds of occupancy belief at time t

Inverse measurement model of Lidar

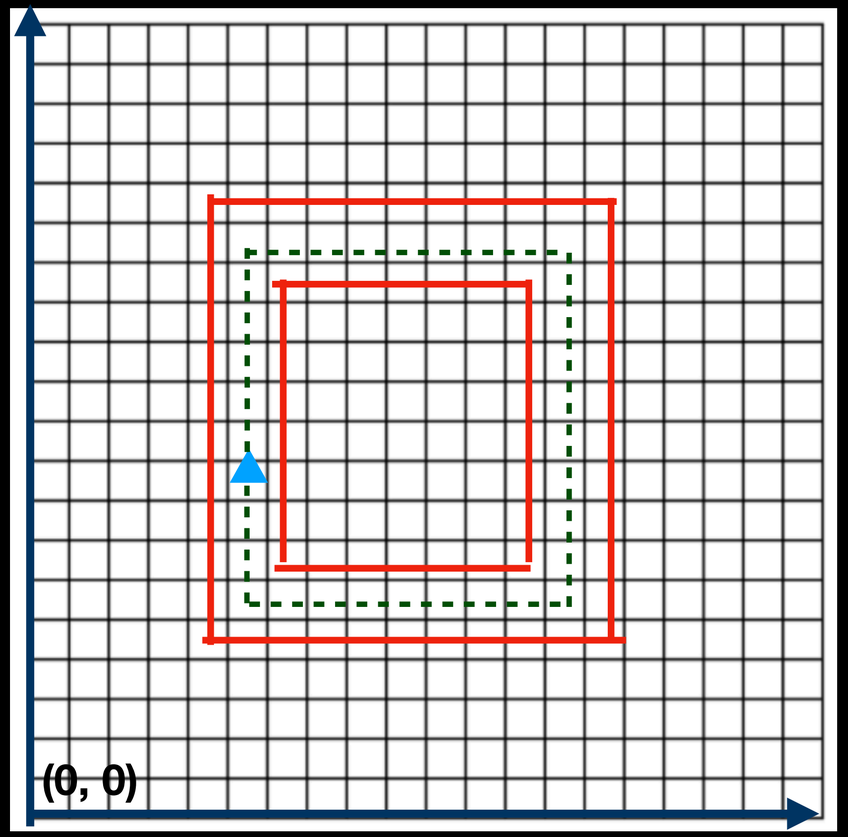
Inverse measurement model

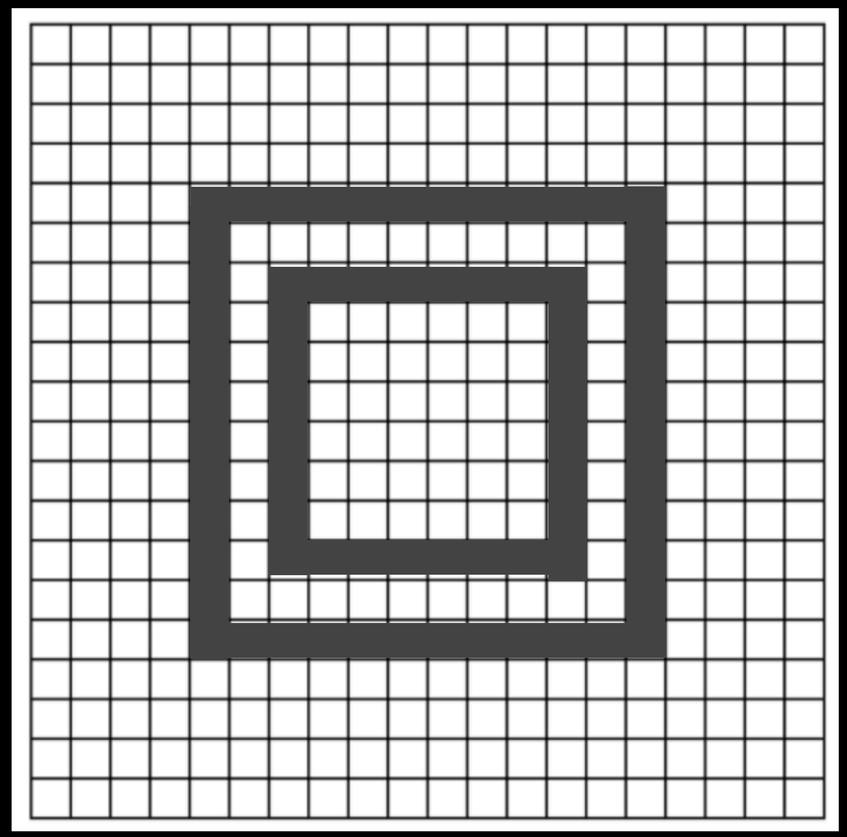


Mapping example in python notebook

3 possible regions from Lidar measurements

Ground truth and Loop closure





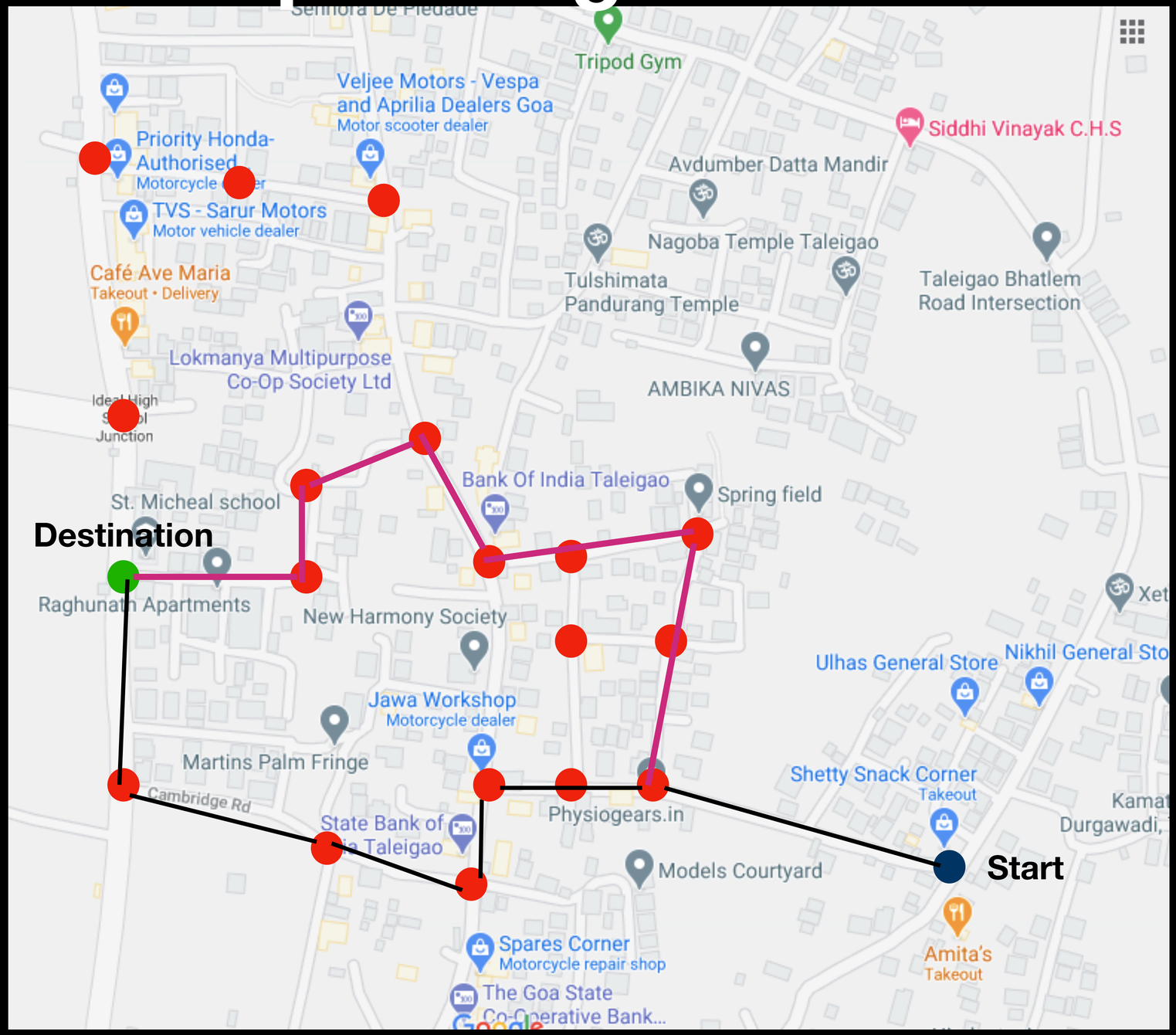
Not practical to manually verify generated maps
How do we know the built map reflects reality?

- Recognizing robot came back to same location => Loop closure

- Example: Consider robot moving around an annular square room
- SLAM - alternatively finds the new position of the robot and updates the map of the room
- Loop closed => Estimate of final robot position is same as the starting position
- Map is consistent with the ground reality

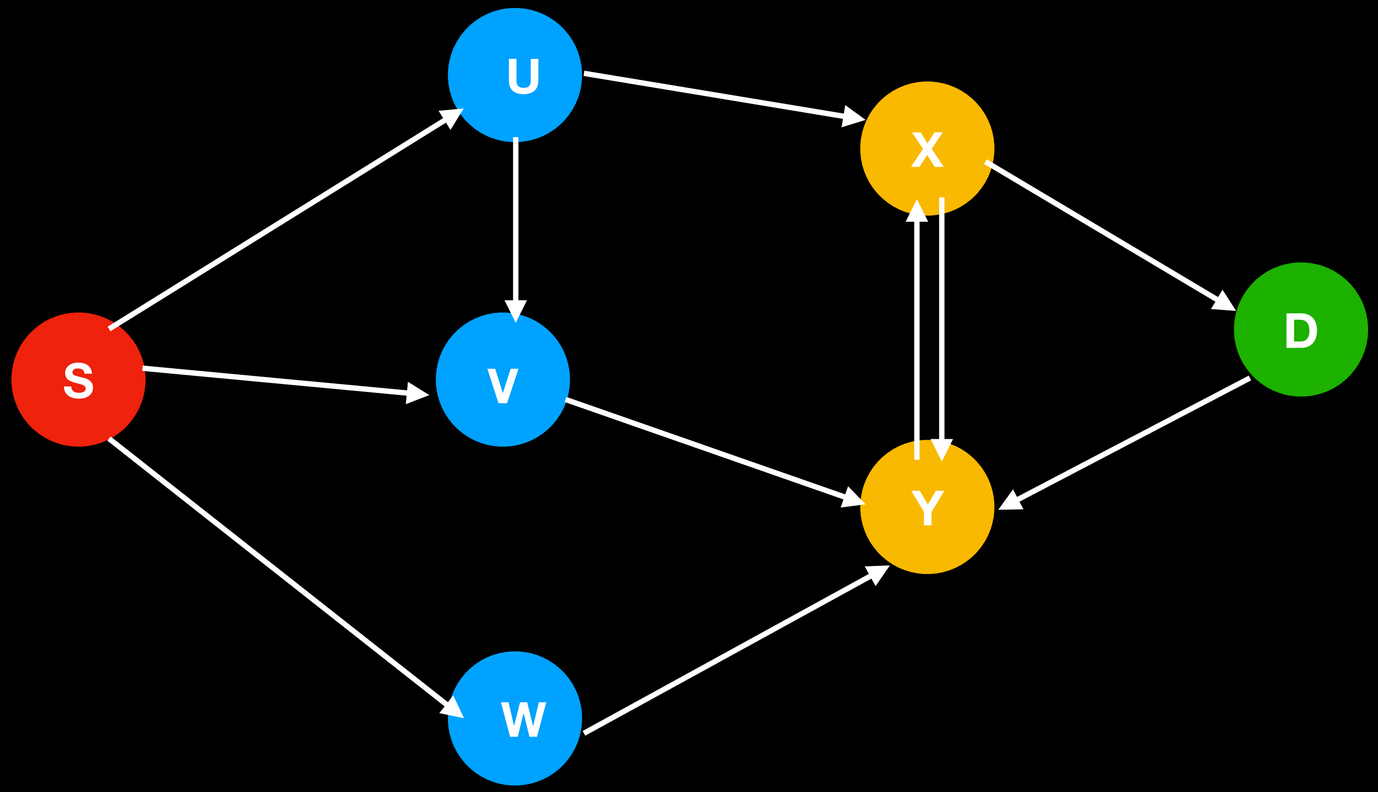
Path Planning and tracking

Mission planning



- How to go from start to destination
- Focus: Speed limits, traffic flow rates, road closures
- No visibility to dynamics and obstacles
- Directed graph can represent road-network
  - One-ways may be present

Shortest path in unweighted graphs



Road network = Topological graph

Graph $G = (V, E)$

Unweighted graph: all edges are identical

What is the shortest path from S to D?

Wavefront search

At stage k, process all nodes that are k-steps from S

Terminology

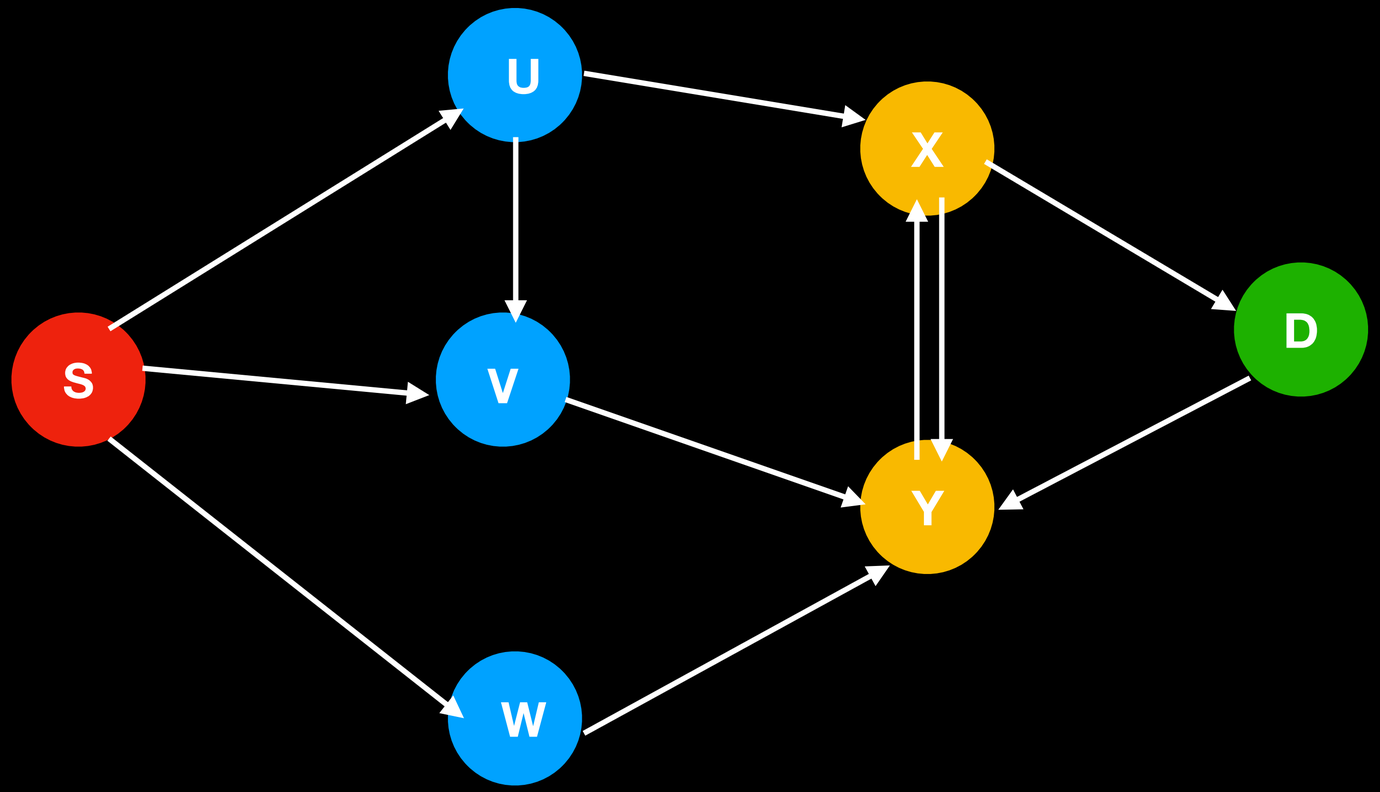
Closed set = vertices already processed

Predecessors = how to reach each node in the closed set

Open set = vertices under consideration

Wavefront search

Iteration 1



Open set

Predecessors	Vertex
-	S

Successors

U

V

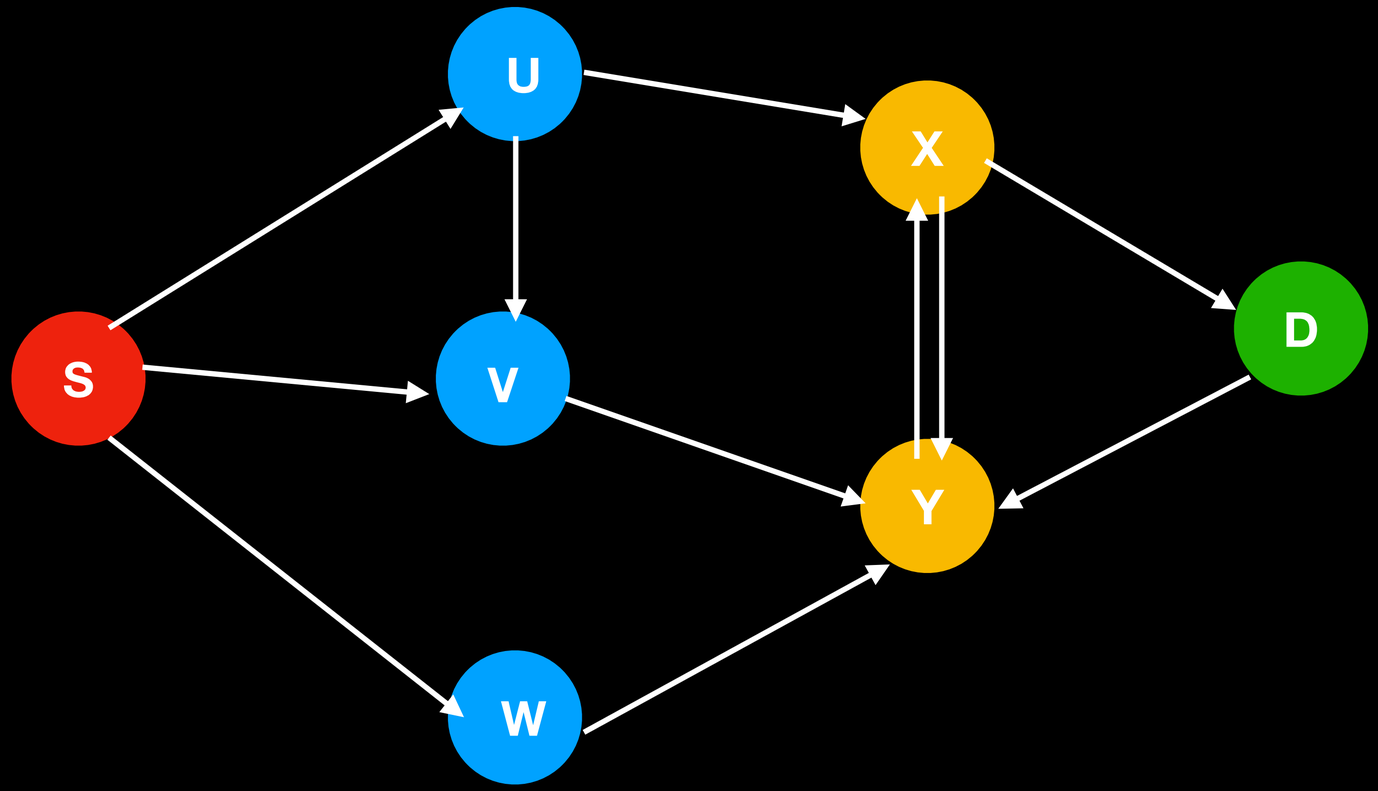
W

Closed set: Empty

1. Evaluate the successors of all nodes in Open set
2. Update predecessor-list of nodes being promoted to Open set

Wavefront search

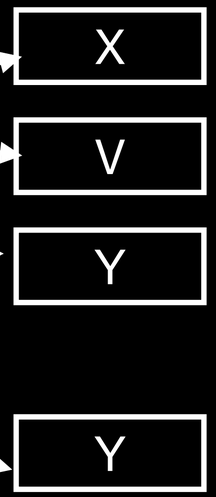
Iteration 2



Open set

Predecessors	Vertex
S	U
S	V
S	W

Successors

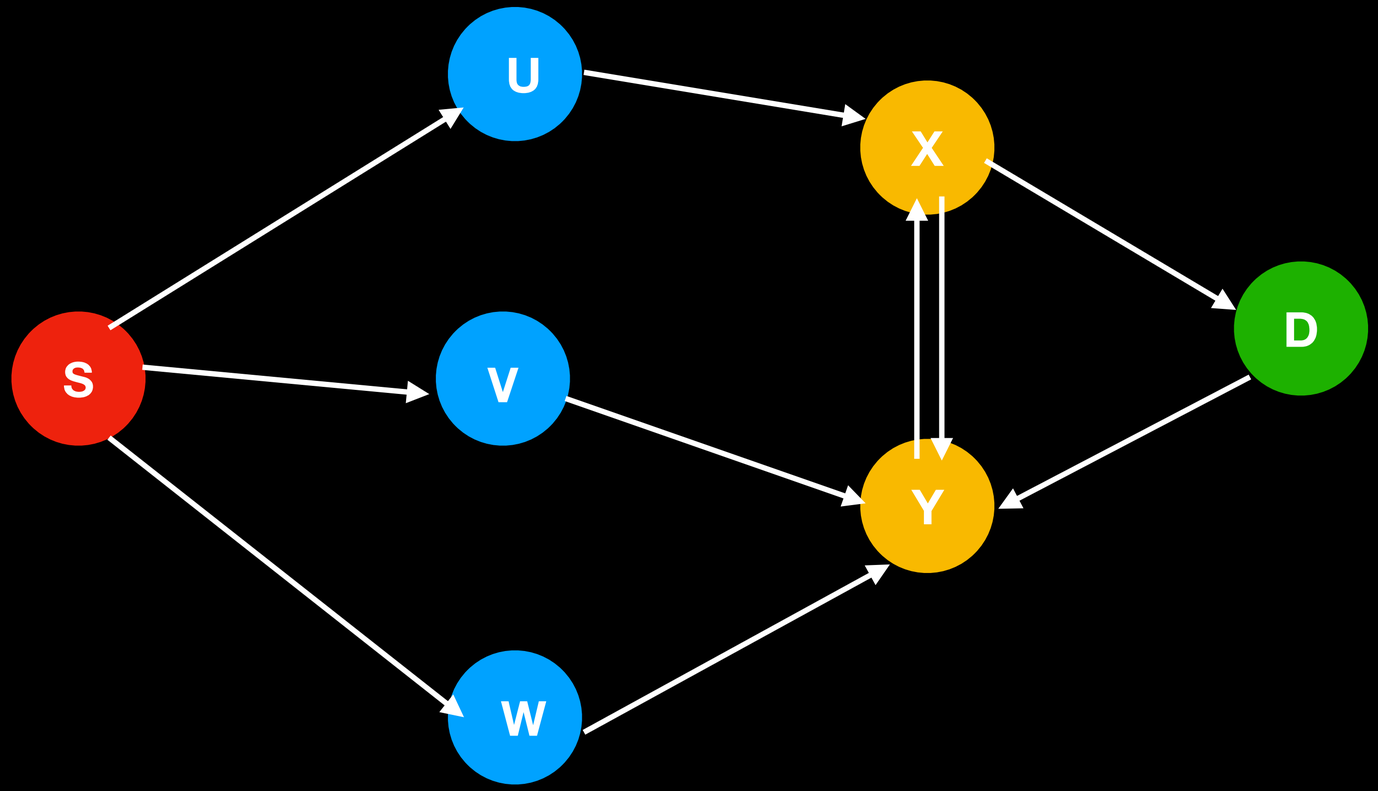


Closed set

S

Wavefront search

Iteration 3



Open set

Predecessors	Vertex
S, U	X
S, V	Y

Successors

D

X

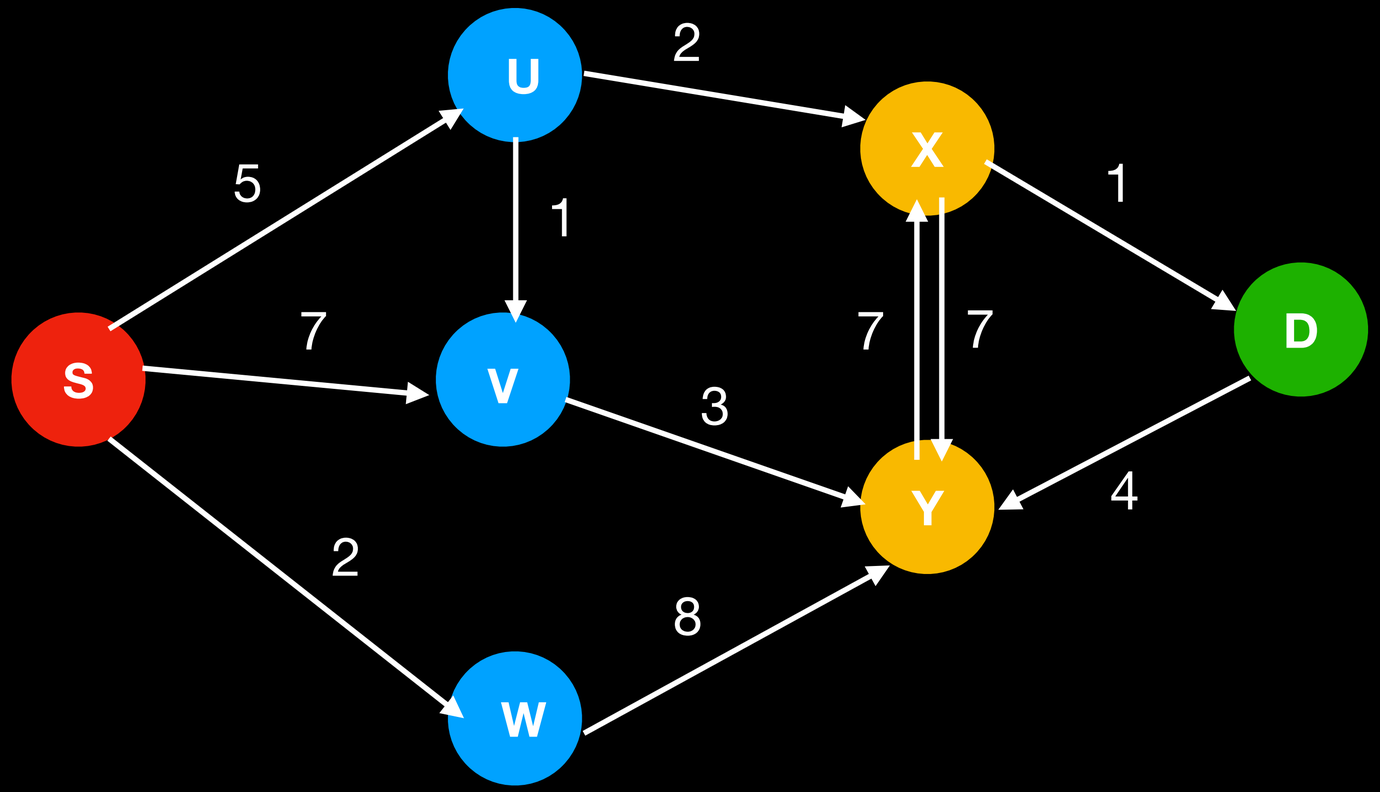
X

Closed set

S
U
V
W

Optimal path: S, U, X, D

Shortest path in weighted graphs



All edges may not have identical costs to traverse

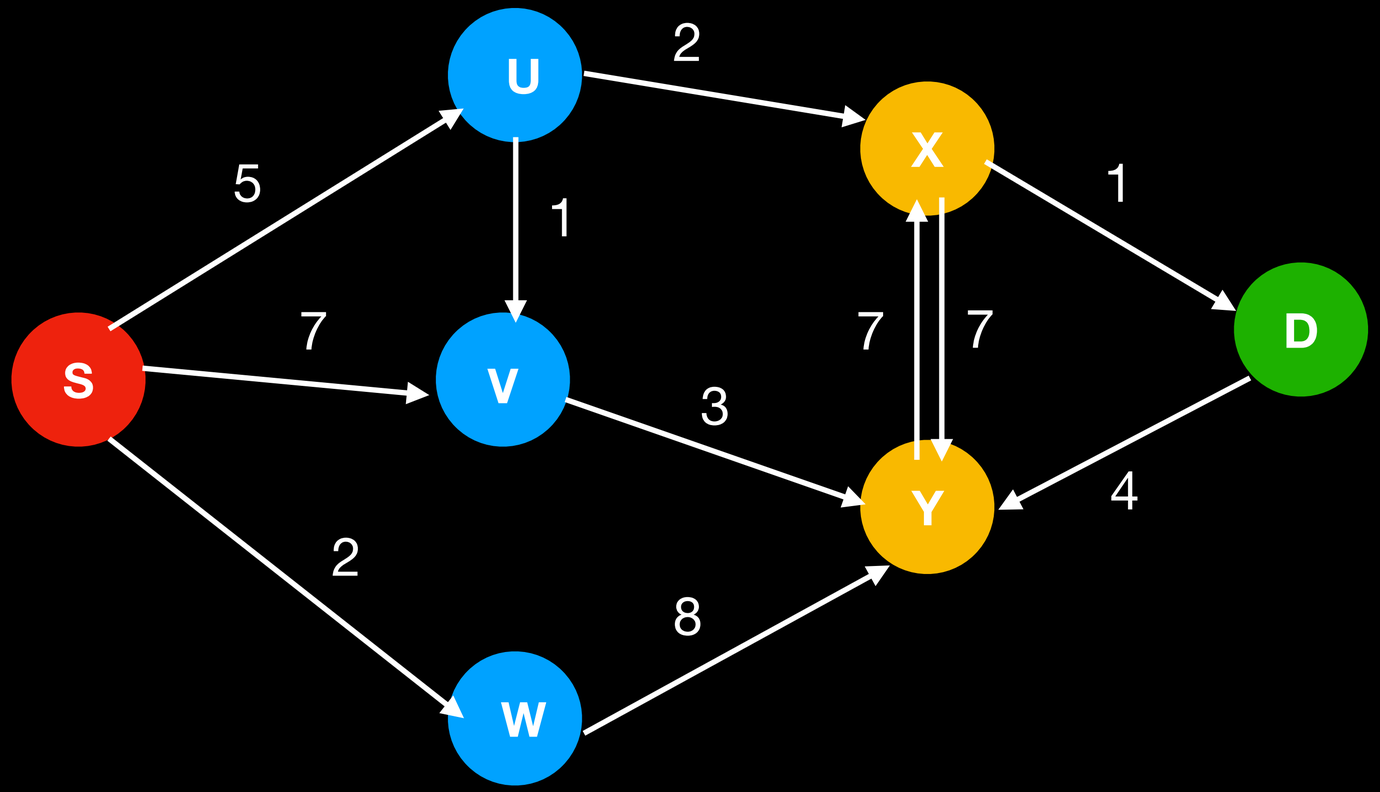
- road terrain conditions
- road length/ time to travel
- pay tolls

Djikstra's algorithm

Factors in cost of edge traversal

Djikstra's algorithm

Iteration 1: node W



Open set

Predecessors	Cost to Source	Vertex	Successors
S	2	W	Y
S	5	U	X, V
S	7	V	Y

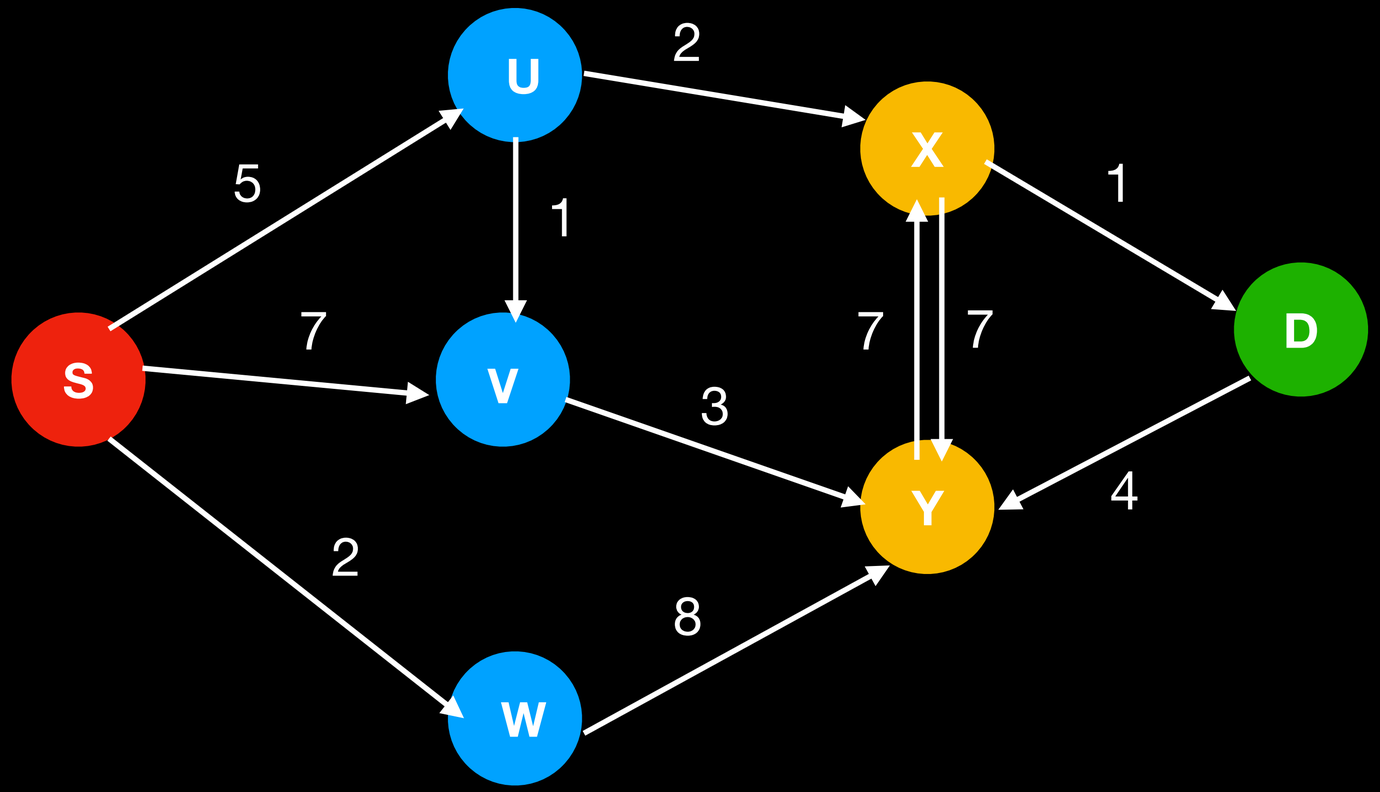
Closed set

S

1. Evaluate the successors of the node with min-cost in Open set
2. Update predecessor-list of nodes being promoted to Open set
3. Update cost to source of nodes being promoted to Open set

Djikstra’s algorithm

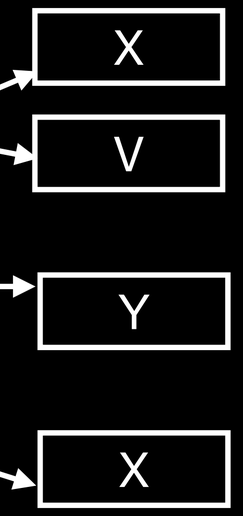
Iteration 2: node U



Open set

Predecessors	Cost to Source	Vertex
S	5	U
S	7	V
S, W	10	Y

Successors

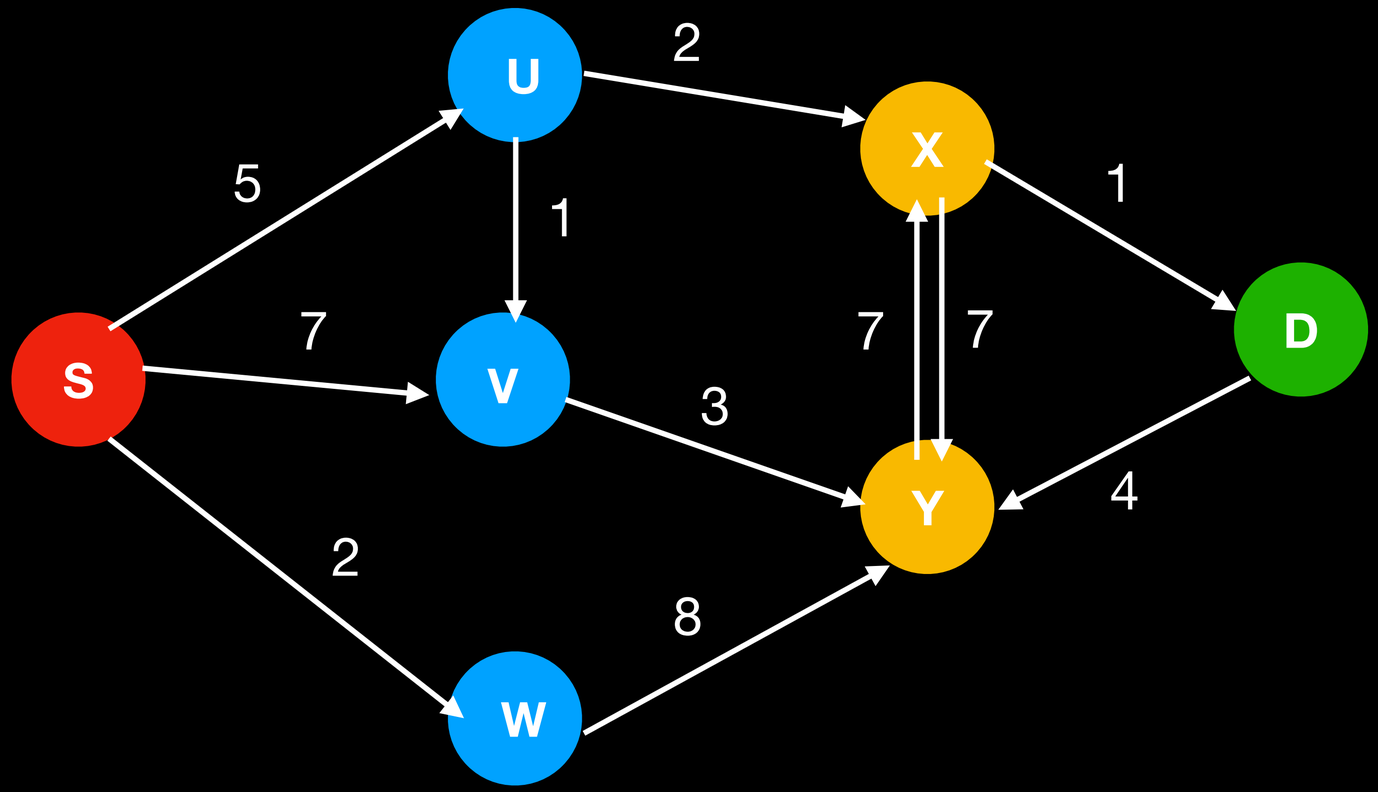


Closed set

S
W

Djikstra’s algorithm

Iteration 3: node V



Open set

Predecessors	Cost to Source	Vertex
S, U	6	V
S, U	7	X
S, W	10	Y

Successors

Y

D

Y

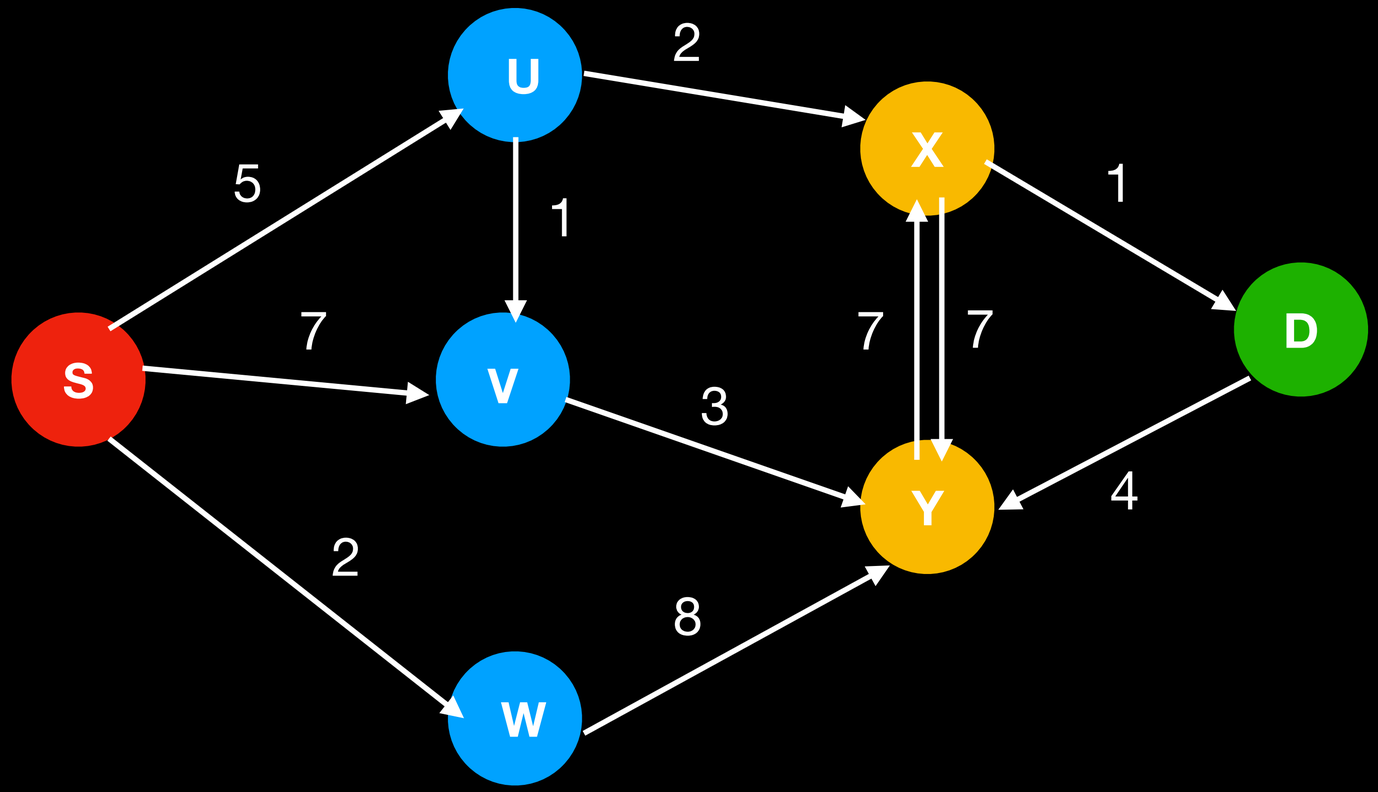
X

Closed set

S
W
U

Djikstra’s algorithm

Iteration 4: node X



Open set

Predecessors	Cost to Source	Vertex
S, U	7	X
S, U, V	9	Y

Successors

D

Y

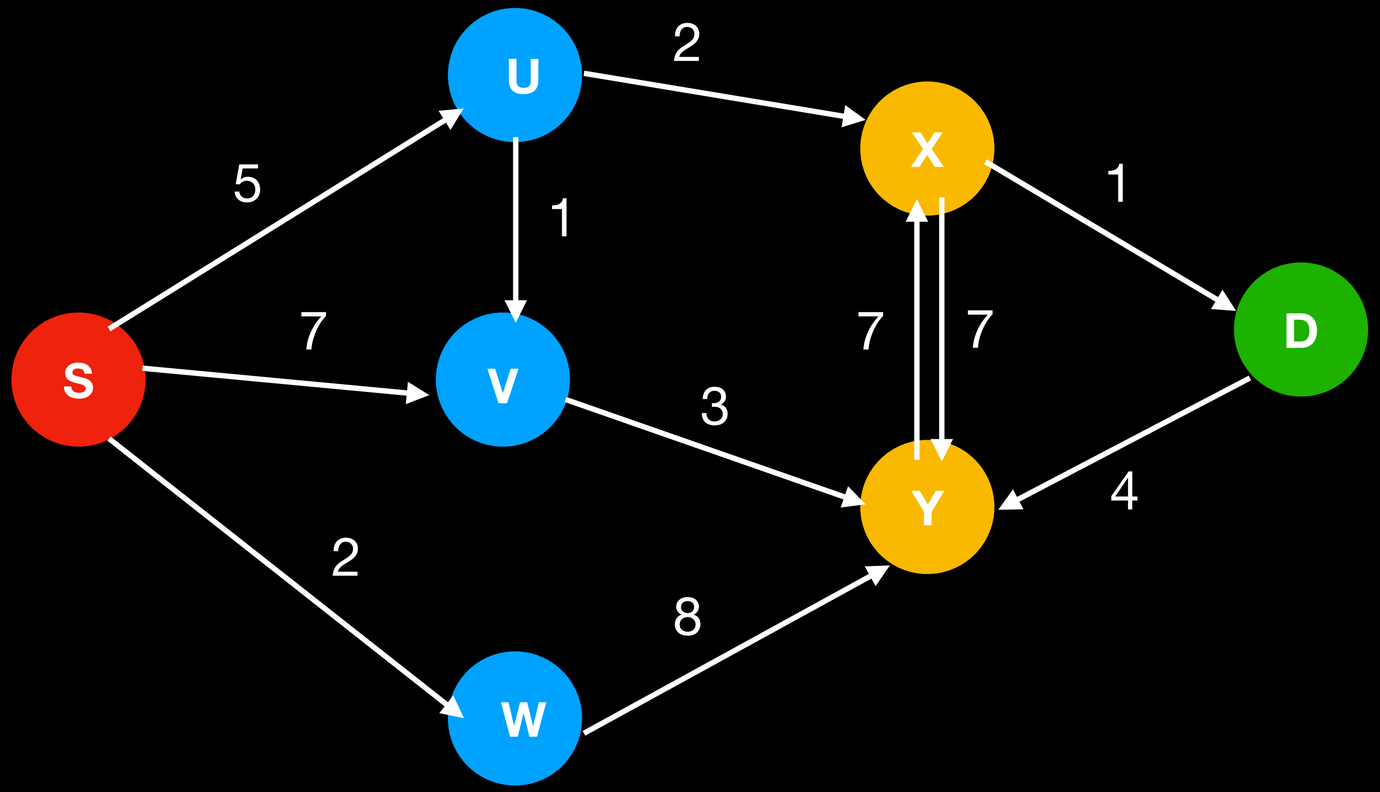
X

Closed set

S
W
U
V

Djikstra’s algorithm

Iteration 5: node D



Open set

Predecessors	Cost to Source	Vertex
S, U, X	7	D
S, U, V	9	Y

Successors

Terminated

X

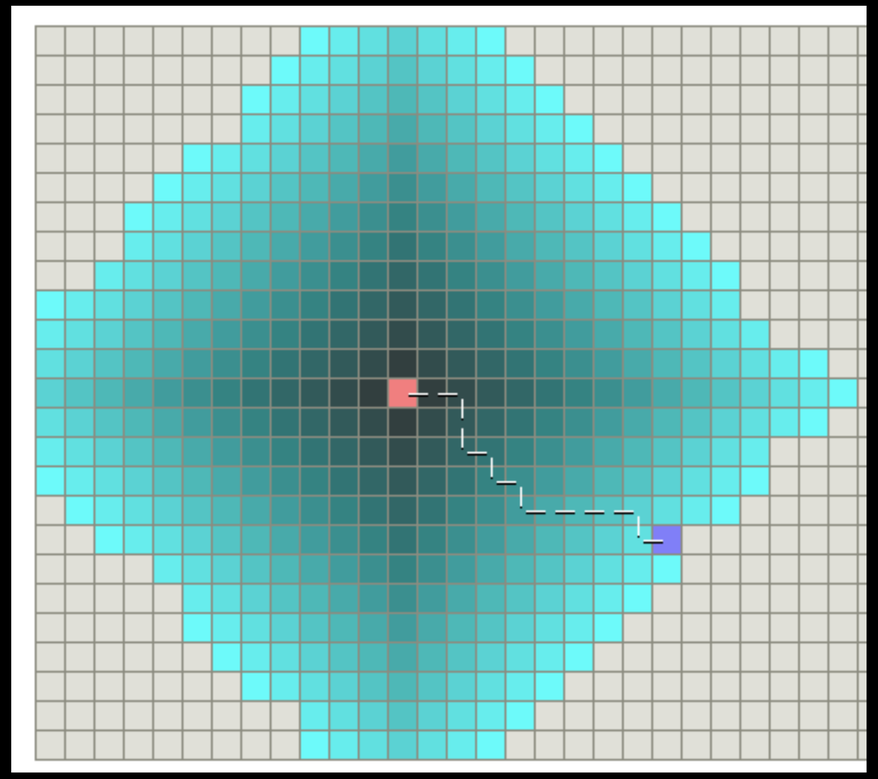
Closed set

S
W
U
V
X

Optimal path: S, U, X, D

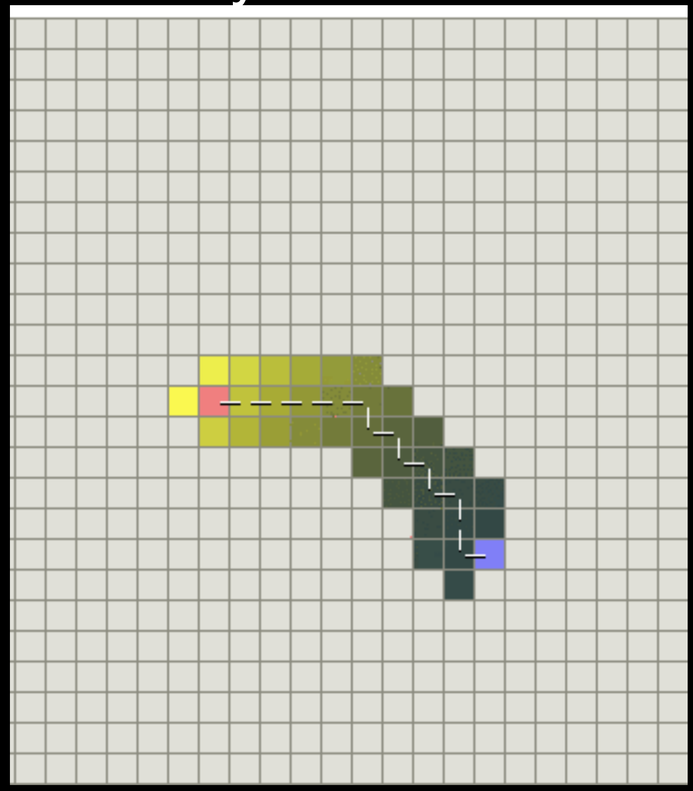
Djikstra

- Outward expansion from source node till destination is reached
- Slow but guaranteed shortest-path



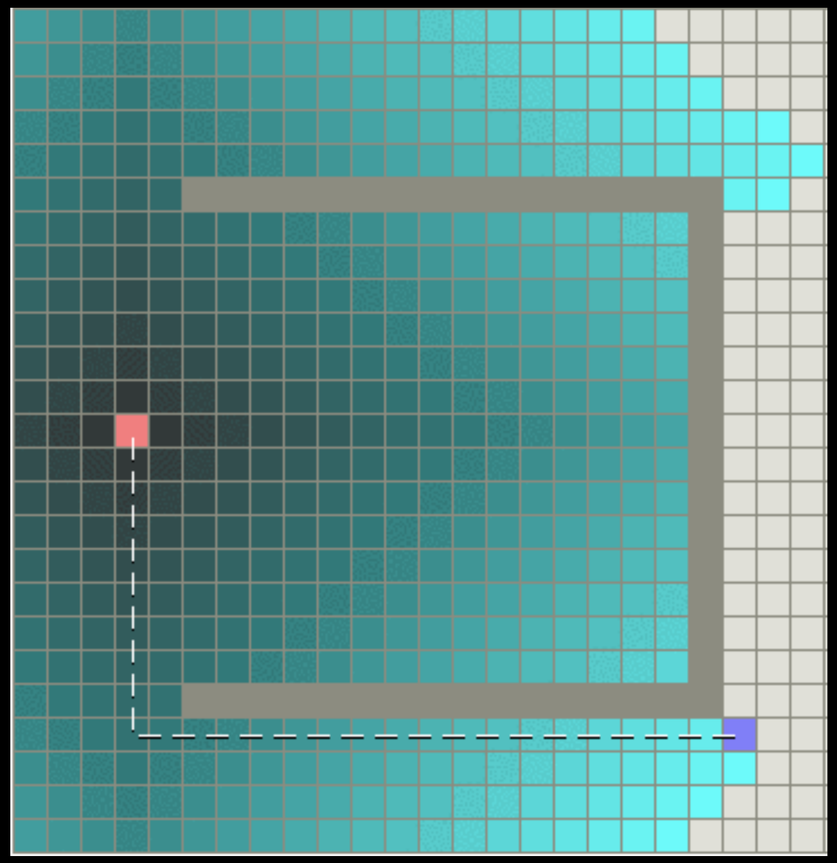
Greedy best-first

- Select the neighbor which has lowest distance to goal
- Distance to goal = some heuristic function
- Can be very fast



Djikstra

- Slow-ish for straight-forward courses



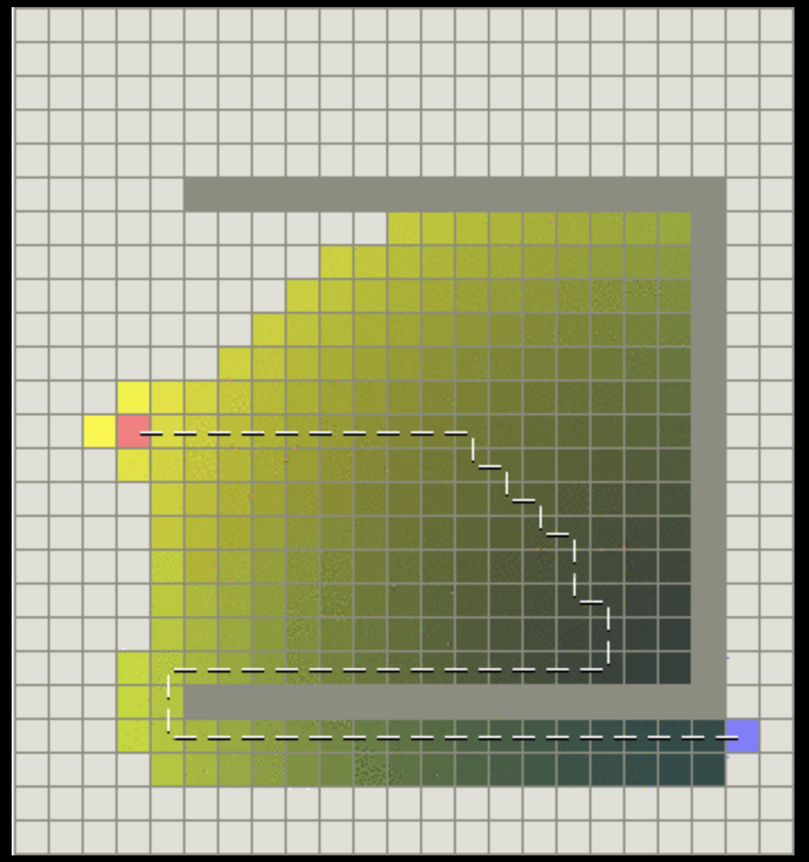
A* = combines the best features

Node Cost = Cost to Source + Heuristic cost to Destination

g(n) = c(n) + h(n)

Greedy best-first

- Not so desirable paths in face of some convex obstacles

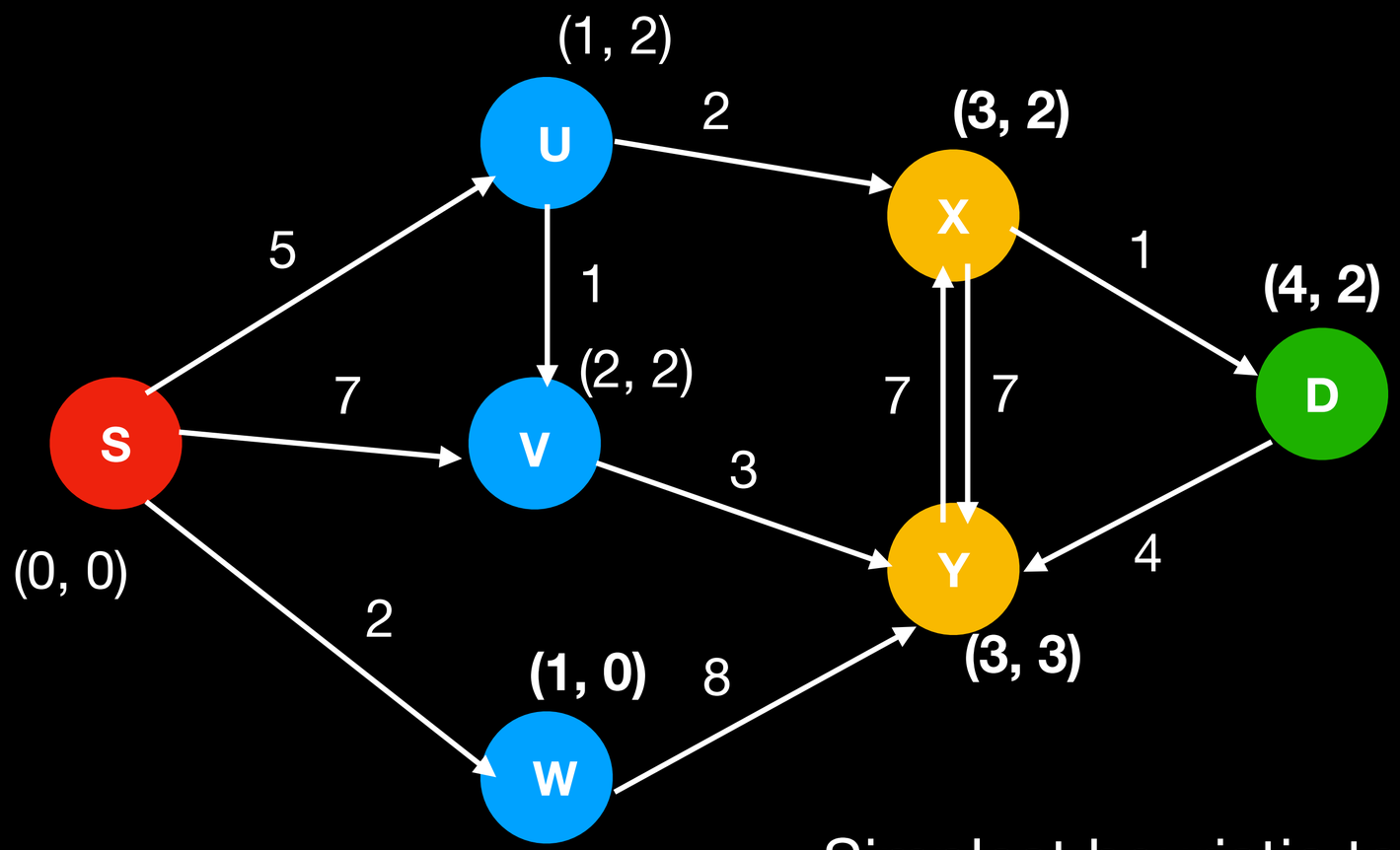


h(n) = 0 => Djikstra

h(n) >> c(n) => Greedy

A* search

Iteration 1: node S



Open set

Predecessors	Cost + Heuristic	Vertex
-	4.89	S

Successors

U

V

W

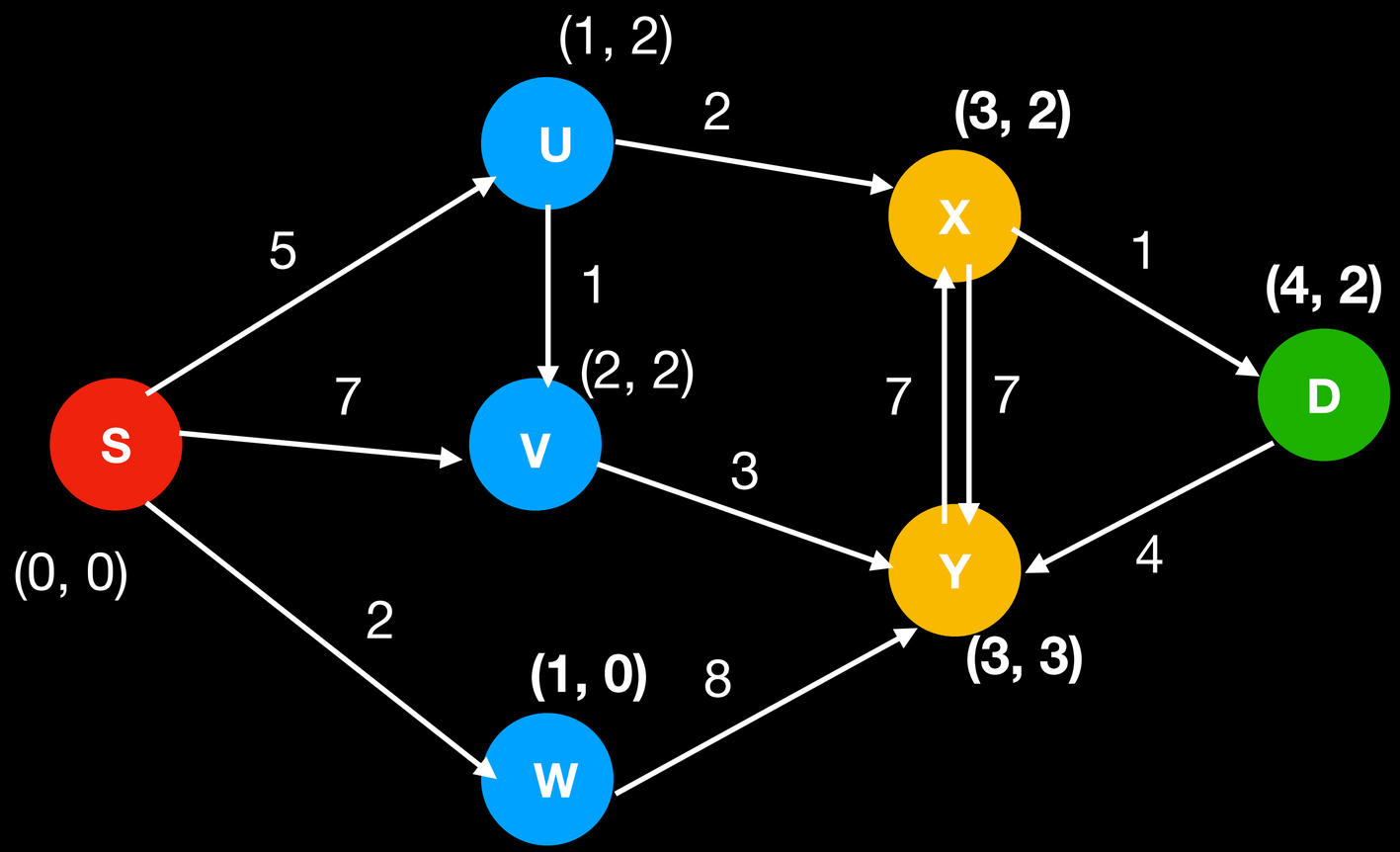
Closed set: empty

Simplest heuristic to destination = Euclidean distance

Can also use time estimate as heuristic

A* search

Iteration 2: node W



Open set

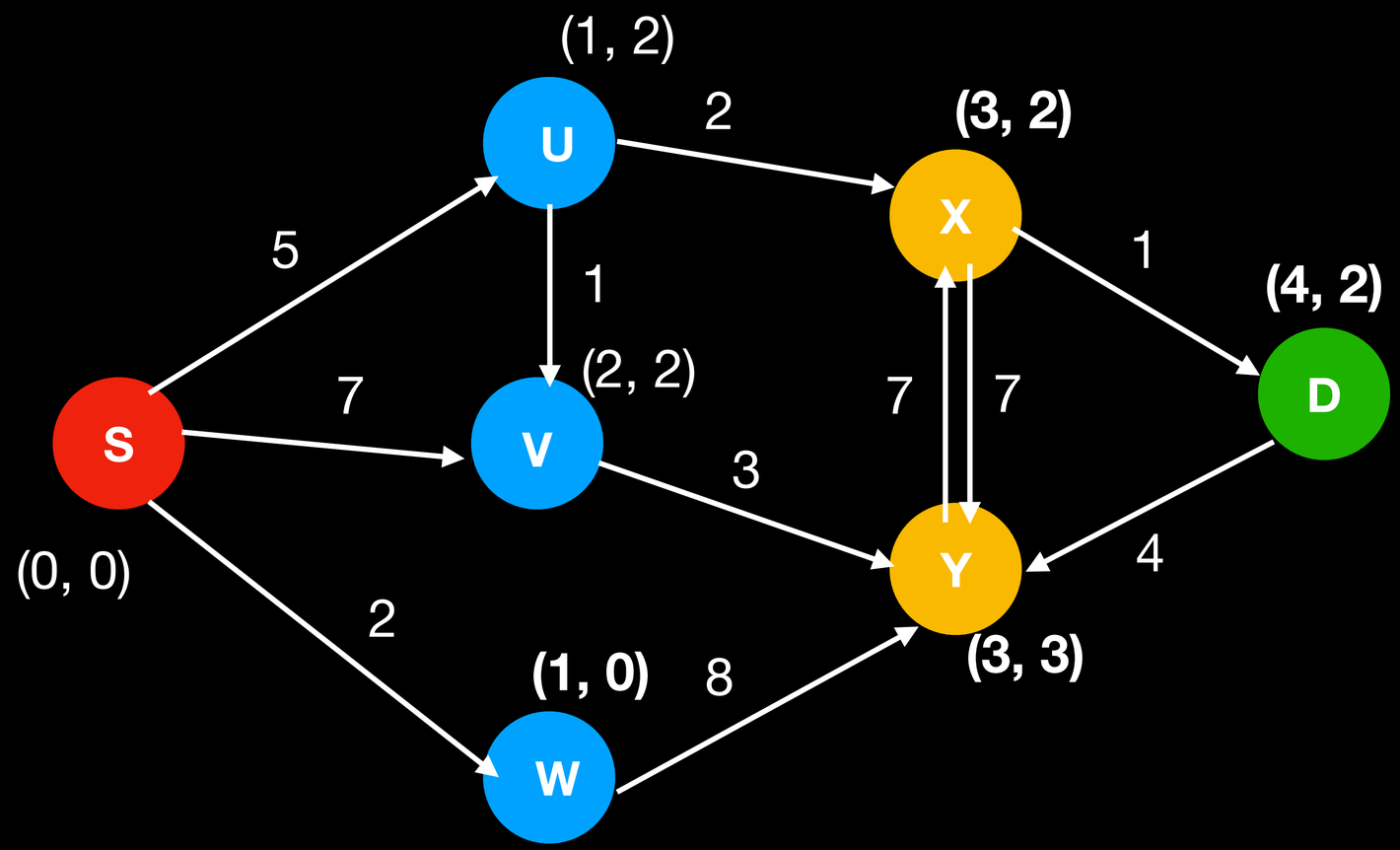
Predecessors	Cost + Heuristic	Vertex	Successors
S	5.6	W	Y
S	8	U	X, V
S	9	V	Y

Closed set

S

A* search

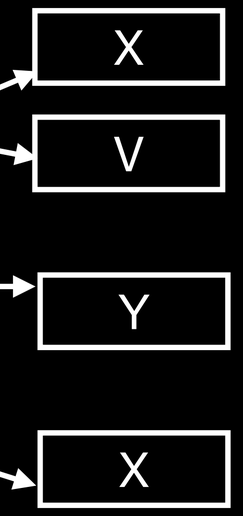
Iteration 3: node U



Open set

Predecessors	Cost + Heuristic	Vertex
S	8	U
S	9	V
S, W	11.414	Y

Successors

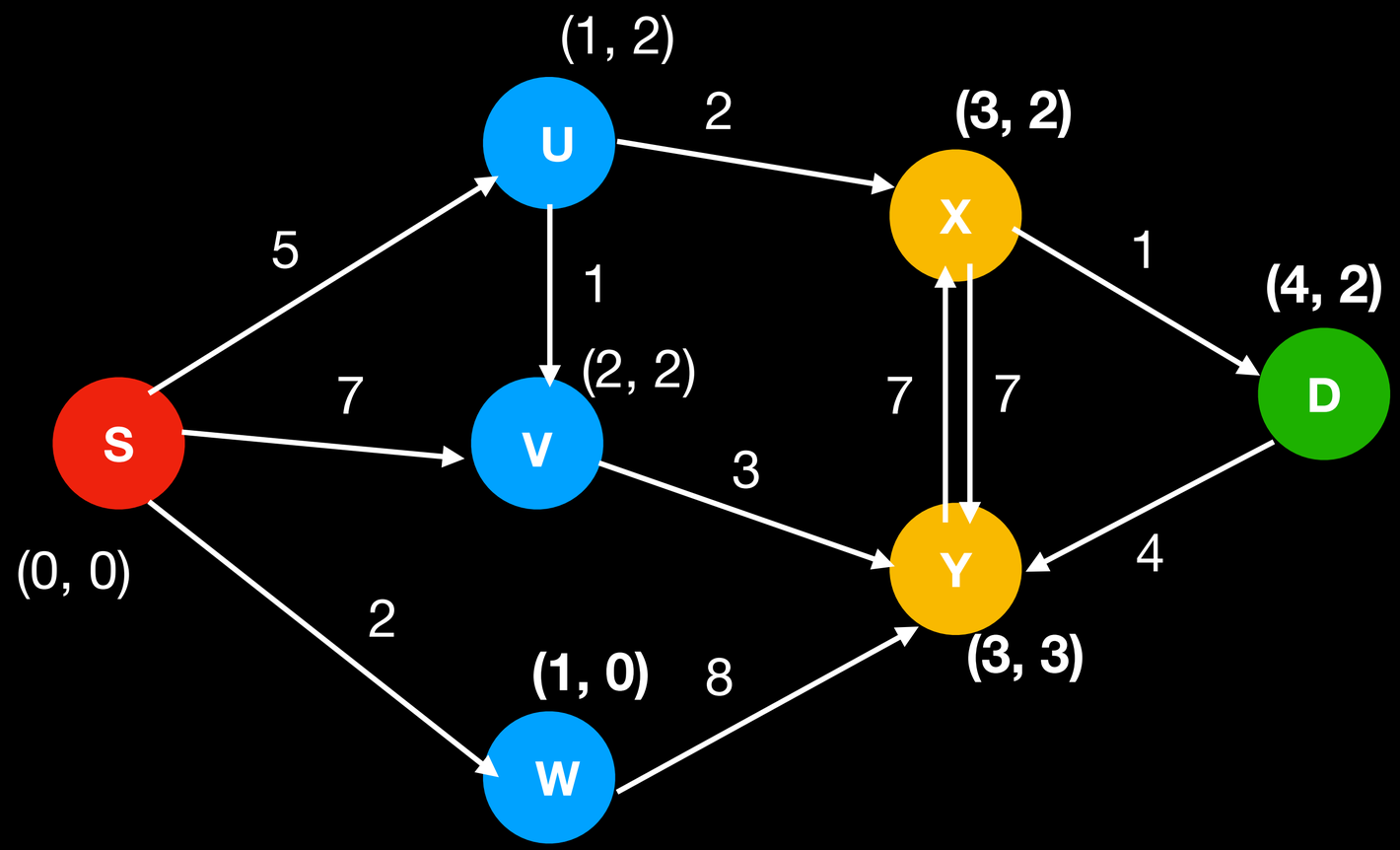


Closed set

S
W

A* search

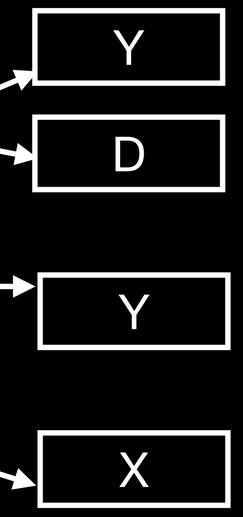
Iteration 3: node X



Open set

Predecessors	Cost + Heuristic	Vertex
S, U	8	X
S, U	8	V
S, W	11.414	Y

Successors

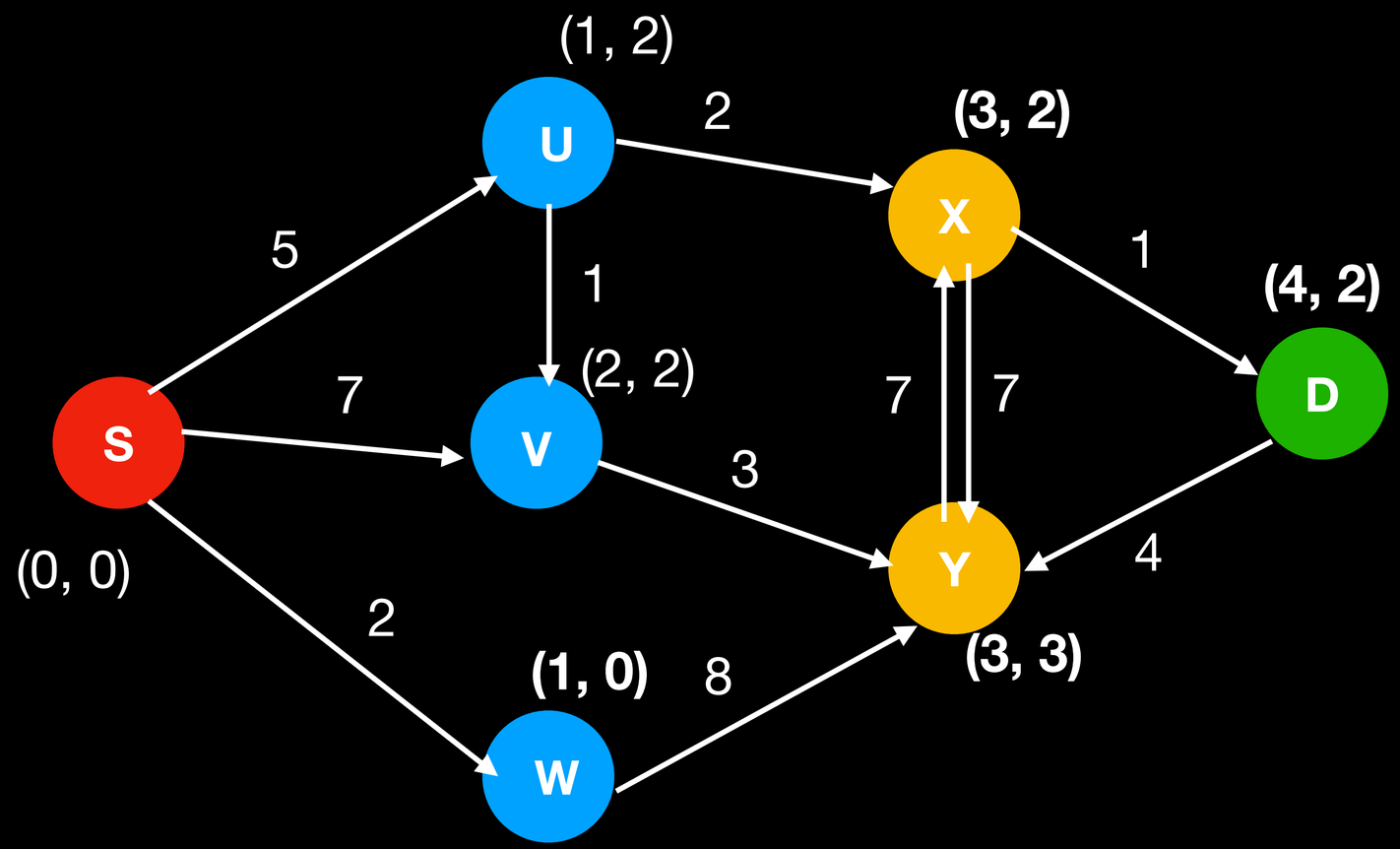


Closed set

S
W
U

A* search

Iteration 4: node D



Open set

Predecessors	Cost + Heuristic	Vertex	Successors
S, U, X	8	D	Terminated
S, U	8	V	Y
S, W	11.414	Y	X

Closed set

S
W
U
X

Optimal path: S, U, X, D

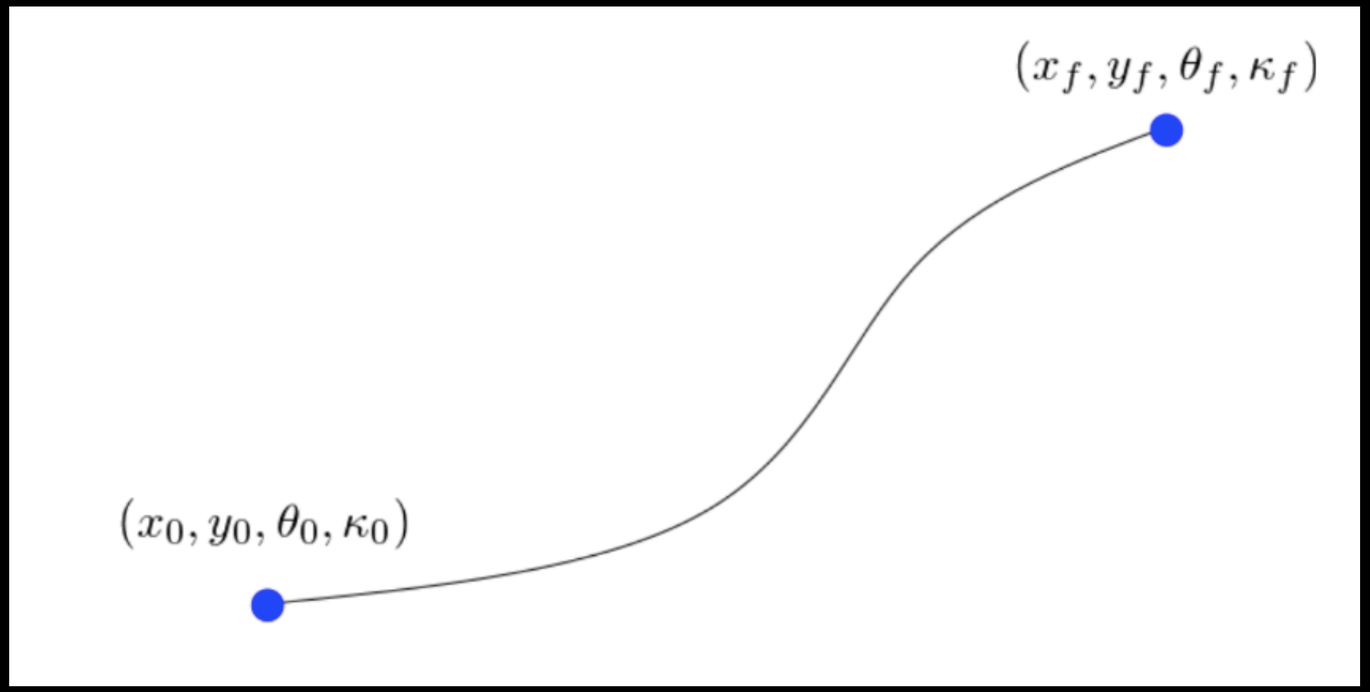
Note that nodes V and Y were not evaluated

Mission-planning in city maps

- Berkley, California
- Open-street maps
- ~2000 edges, 5000 vertices

Python notebook exercise

Local planner: Parameterized Curves



- Path smoothness / continuity are much desired properties
- Time-derivative of orientation => curvature
  - Higher-curvature means sharper turns
- Higher-order derivatives of vehicle position => velocity, acceleration, jerk, snap
- Passenger comfort may impose boundary constraints on higher-order terms

Curve-fitting approach

Fit a polynomial for smooth transition from θ_i to θ_f

For simplicity, $\theta_i = 1$ and $\theta_f = 0$

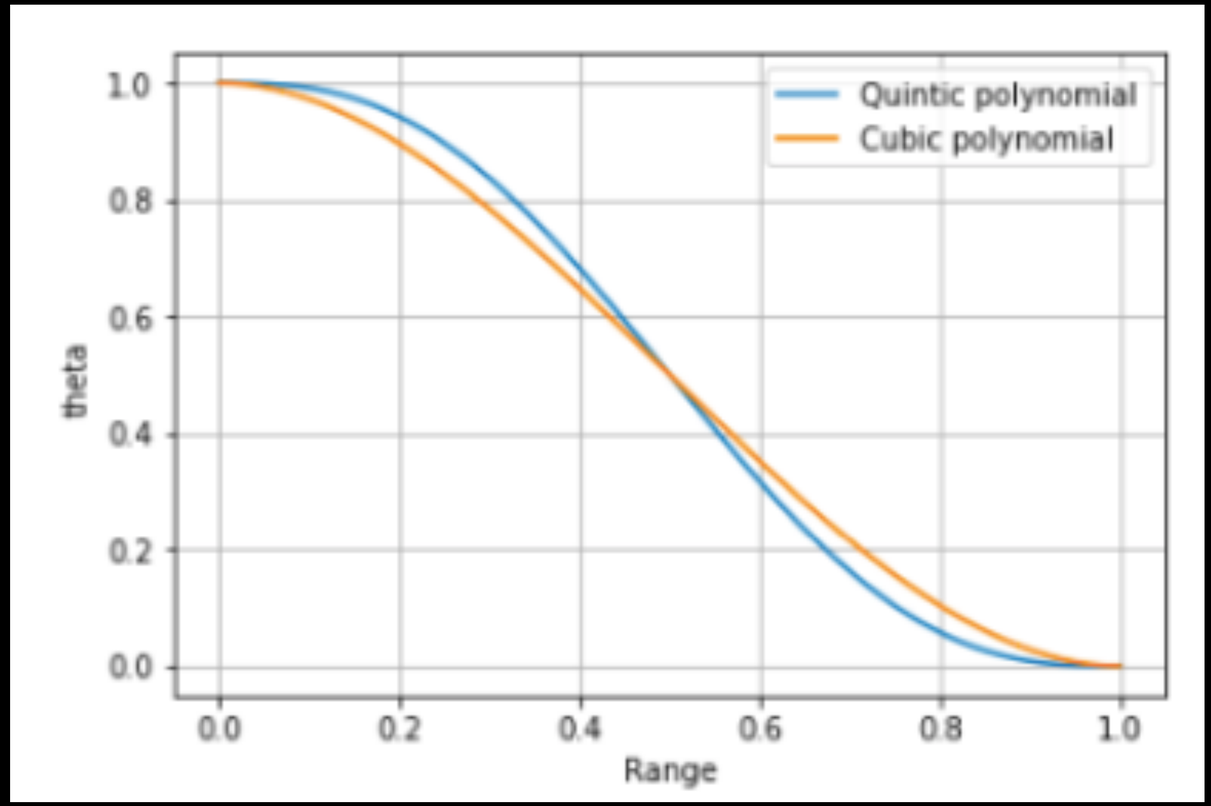
Our function can be parameterized as

$$f'(\theta) = K(\theta(1-\theta))^{2n}$$

- n = 1 -> cubic polynomial
- n = 2 -> quintic polynomial

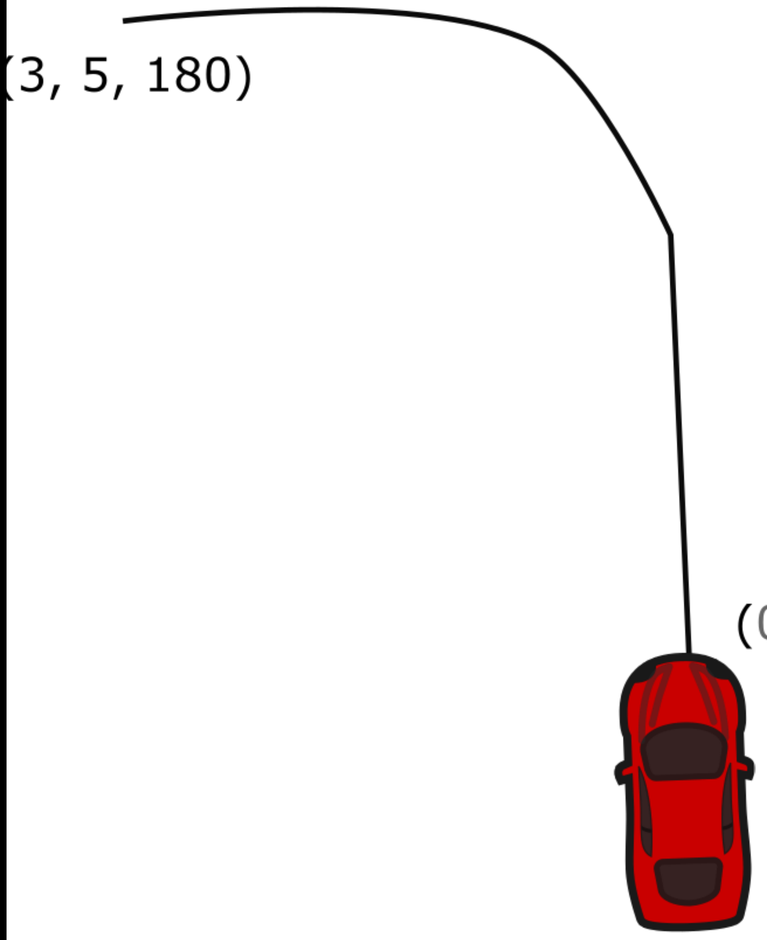
$\{\theta_i\}$ is fixed

$\{x_i\}$ and $\{y_i\}$ can be computed for a fixed v

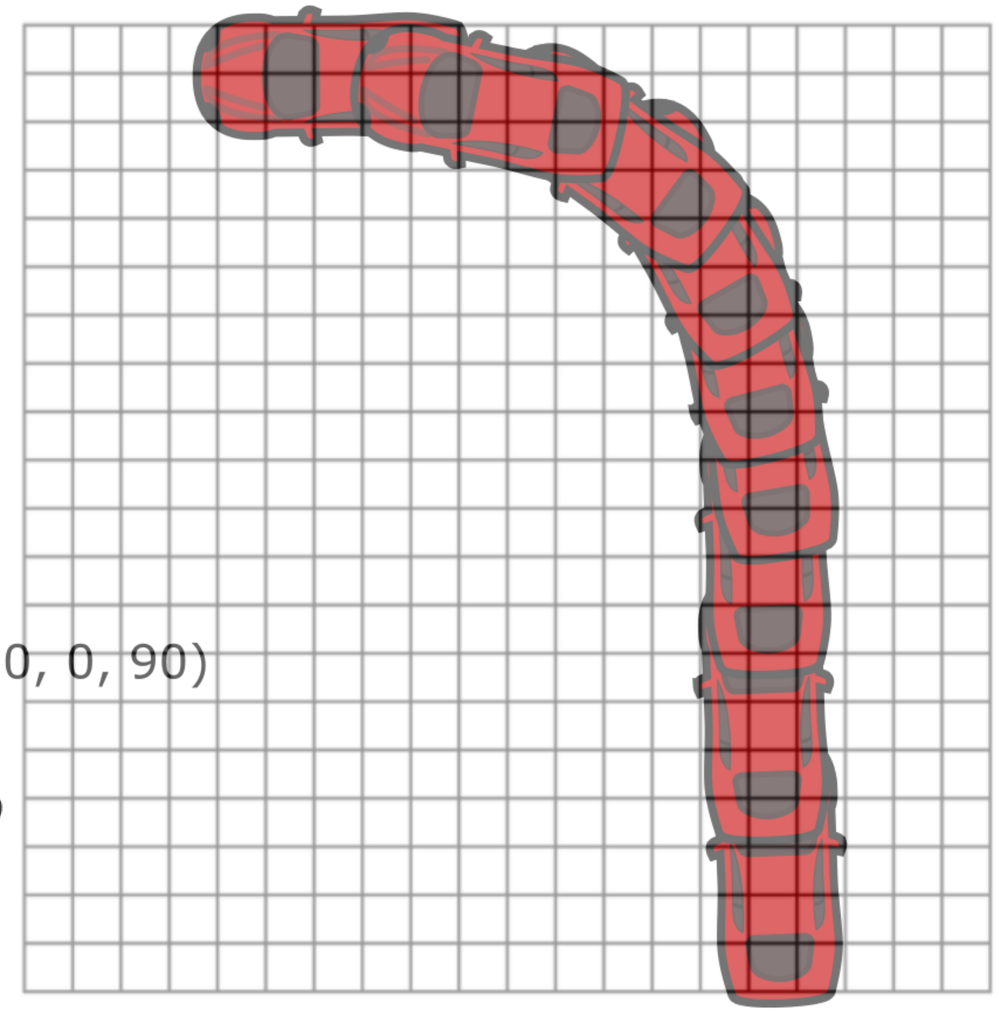


Are these local plans safe?

Collision-checking

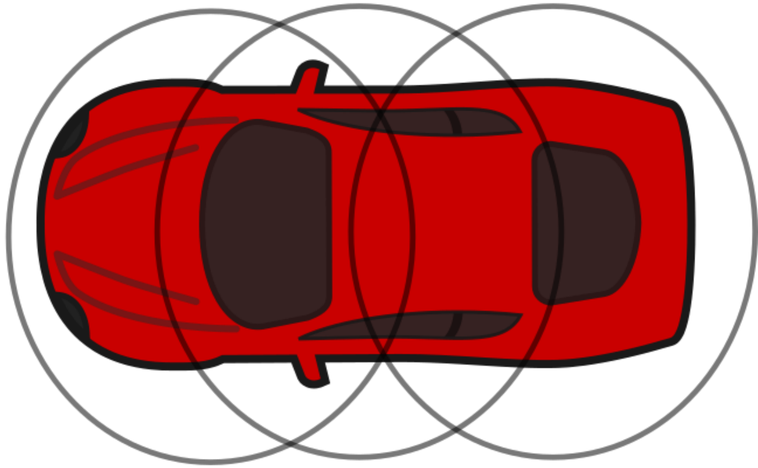


Vehicle is about to take a left turn
Anticipated vehicle trajectory



Actual vehicle check for collision along anticipated trajectory

Vehicle approximated as circles

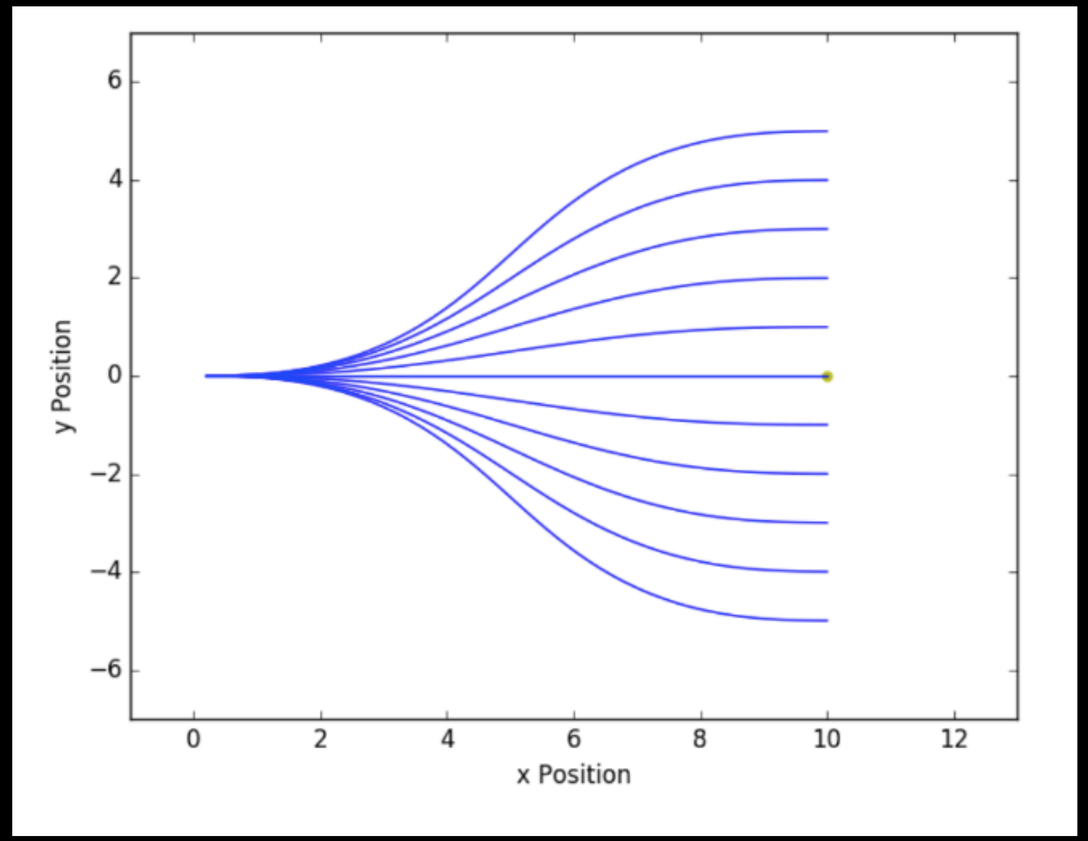


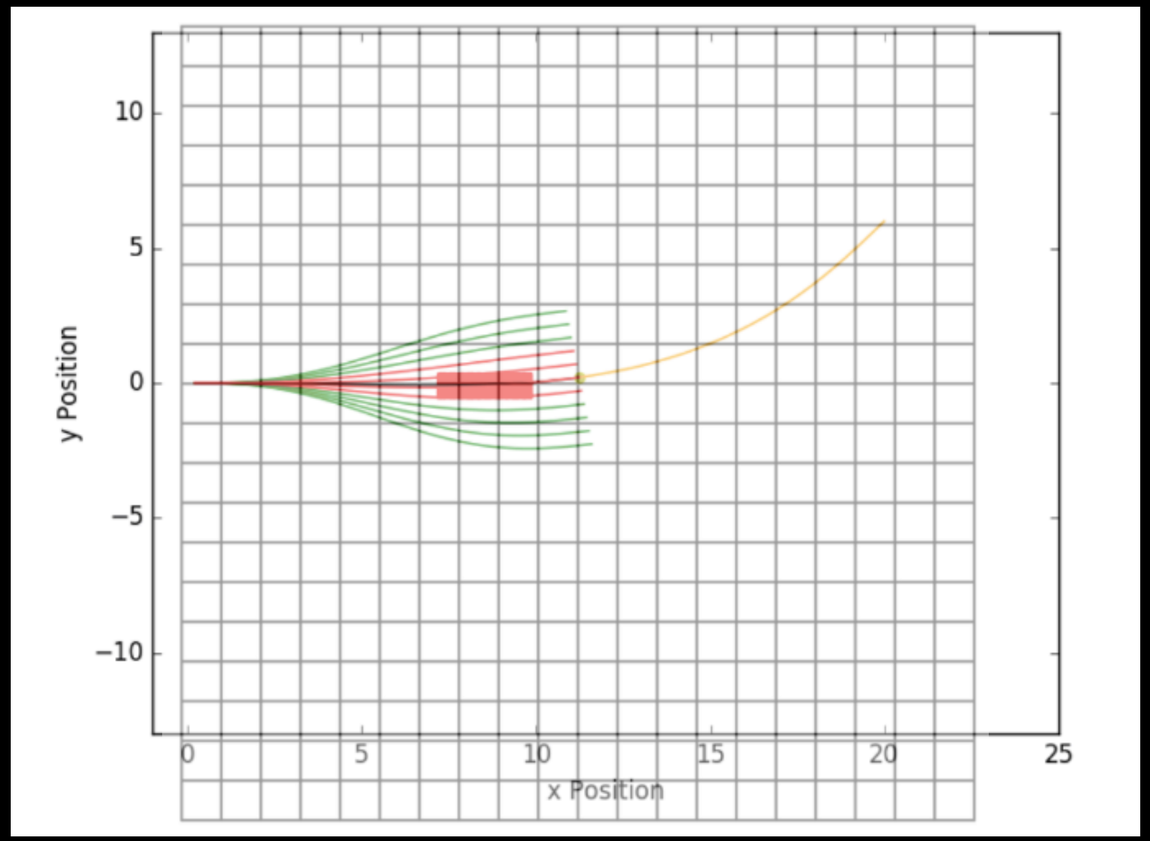
Low-compute collision check

For points in occupancy grid, check

$$\sqrt{(x_c - x_i)^2 + (y_c - y_i)^2} \leq r$$

Lattice planner



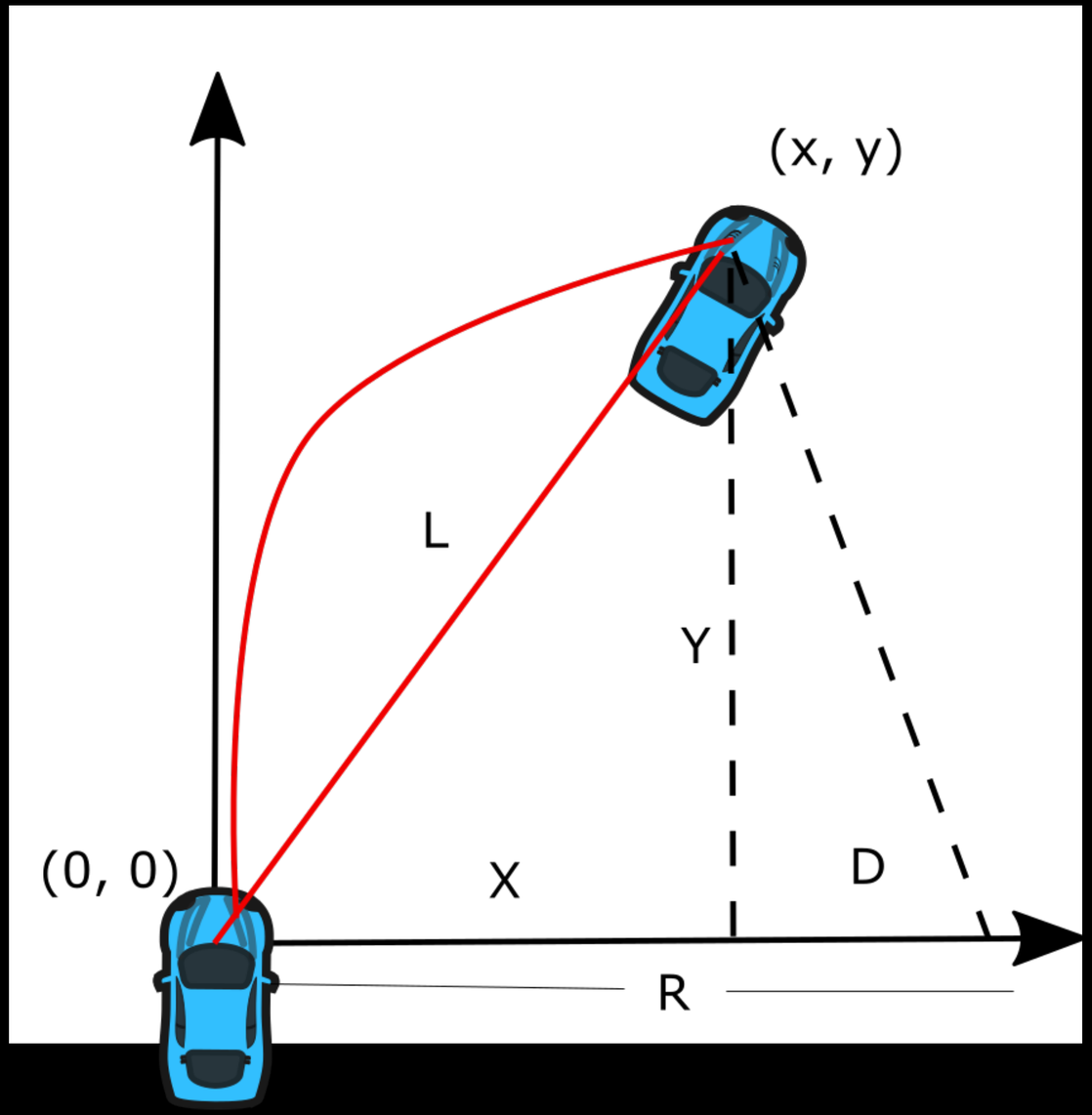


- Entire mission broken into several local plans
  - Intermediate goal point based on a lookahead distance
  - Multiple candidate paths parallel to the goal point
  - Imagine 800m athletic events with runners on parallel lanes
  - The vehicle can shift to any parallel lane which is obstacle-free
  - Lowest cost is assigned to the central lane

Need to actively track

- Local planner
  - Uses vehicle kinematics model and sensor observations - can be noisy/ imperfect
  - Assumed v, ω may not be actuated because of non-ideal motor response
  - Vehicle will deviate from planned local trajectory
  - Need to actively course-correct

Pure pursuit tracker



- Car is at (0, 0). Needs to be at (x, y)
- Not concerned with the orientation
- Assume the vehicle is going to take a circular arc to reach (x, y)

$$X + D = R$$

Pythagoras theorem

$$D^2 + Y^2 = R^2$$

$$X^2 + Y^2 = L^2$$

By eliminating X, Y

$$R = \frac{L^2}{2X}$$

For constant linear velocity V

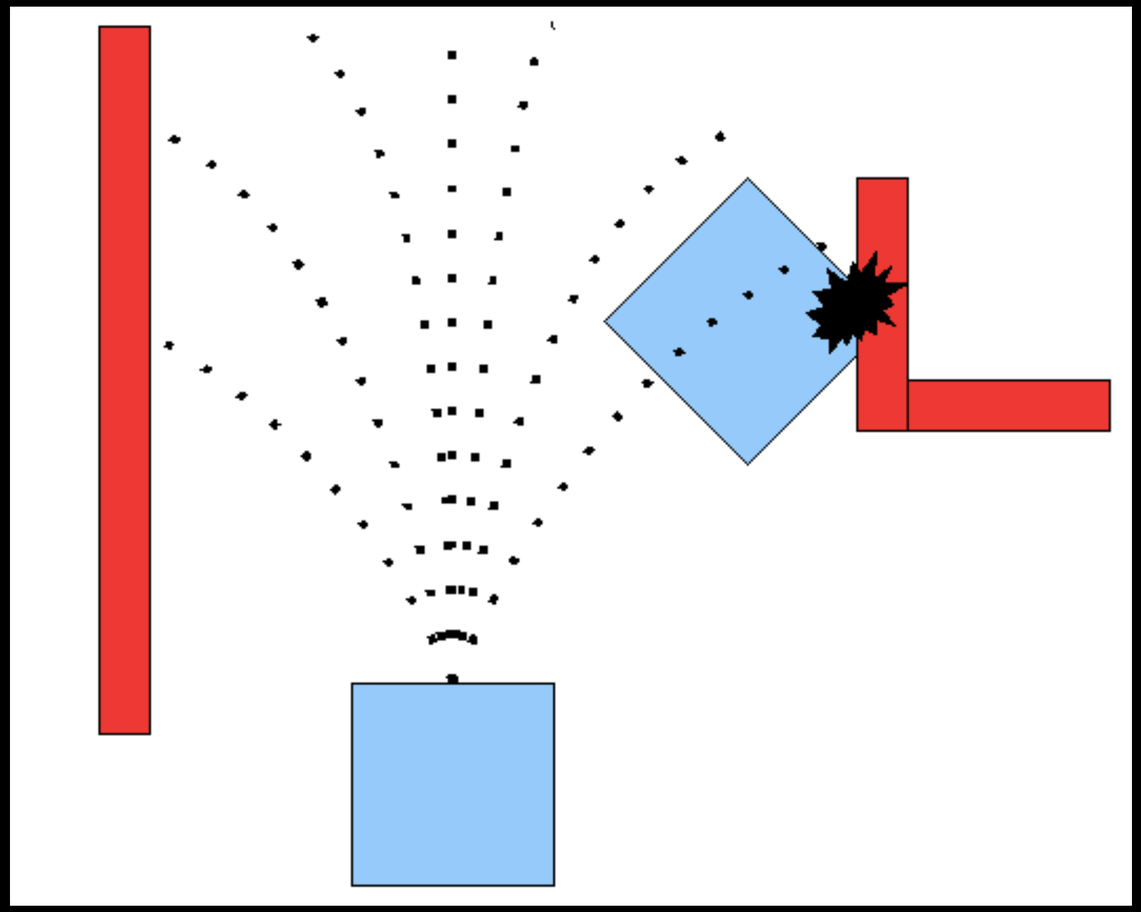
$$\omega = \frac{2V}{L^2}X$$

Pure pursuit: mechanics

- Receding time-horizon planner
- L is the distance horizon for tracker
- v, ω calculated for this horizon
- Only a fraction of this plan executed
- Converges to destination point
  - Closed-loop feedback

- Think of X as cross-track error
- Vehicle is to turn if there is a cross-track error
- Careful choice of look ahead distance L to stabilize the controller
- Trade-off between stability and accuracy

Dynamic window avoidance



- Enumerate all possible control actions
  - Say, (v_t, ω_t) is velocity at t
  - (α, β) are the linear/ angular accelerations
  - $(v_t - \alpha dt, v_t + \alpha dt)$ and $(\omega_t - \beta dt, \omega_t + \beta dt)$ are the range of control actions
  - Ex: (0.9, 1.1) m/s and (-0.1, 0.1) rad/s
- Using every sampled (v, ω) rollout a trajectory as shown
- Trajectories that collide are discarded
- Of the safe trajectories pick the one with the least-cost
  - Eg: trajectory closest to center