MTRN4110 Robot Design
Week 5 – Planning II

Liao “Leo” Wu, Lecturer

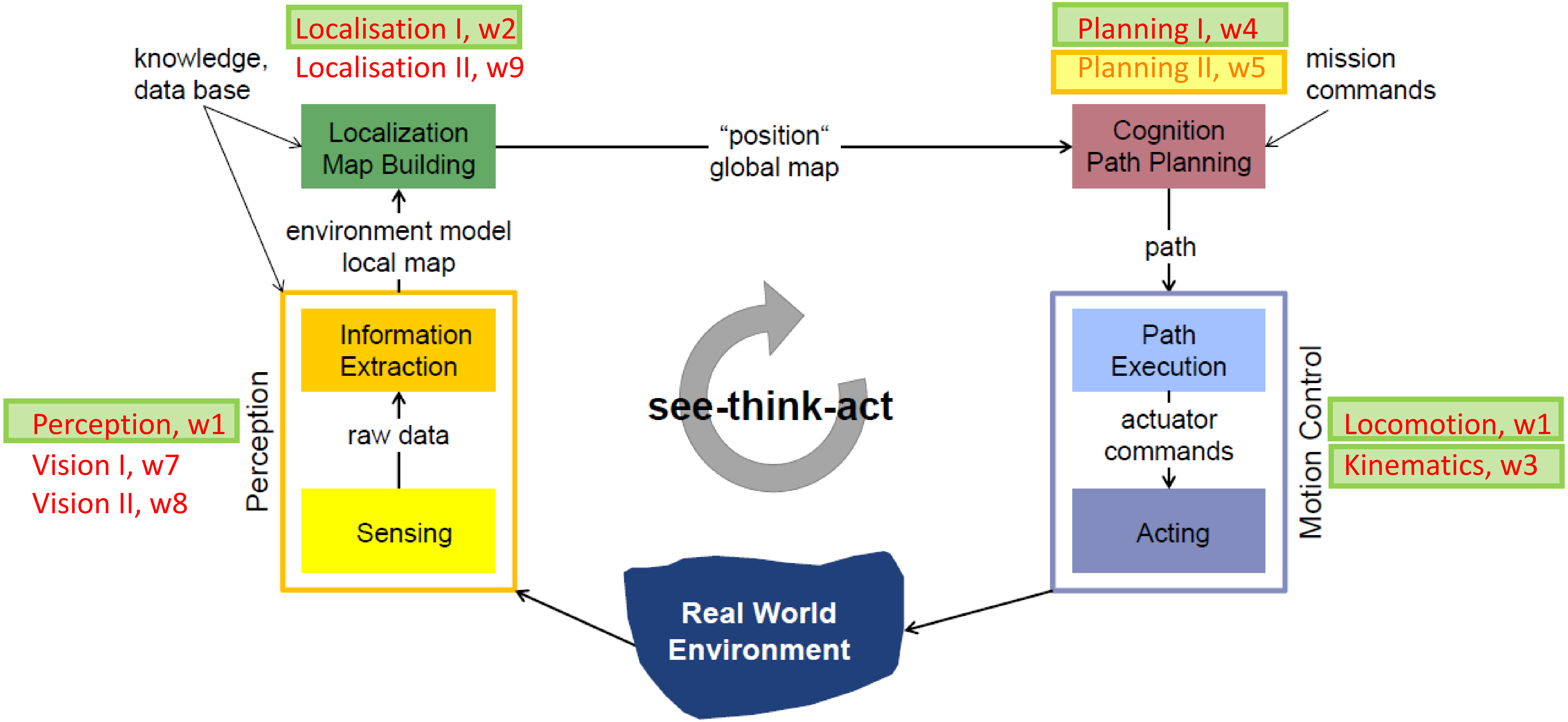
School of Mechanical and Manufacturing Engineering

University of New South Wales, Sydney, Australia

https://sites.google.com/site/wuliaothu/

Kahoot Questions

The See-Think-Act cycle



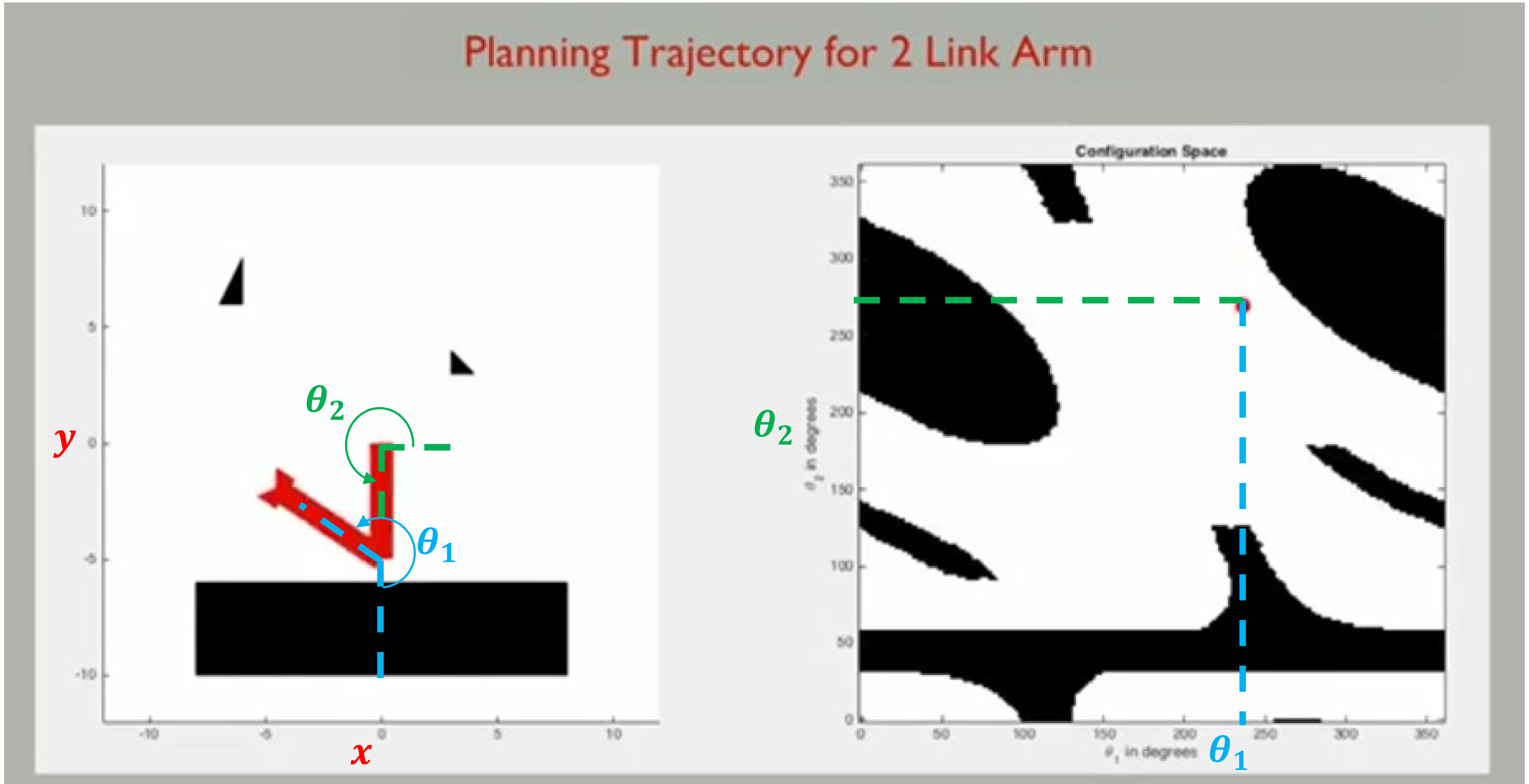
R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to autonomous mobile robots. The MIT Press. Second edition. 2011.

Today's agenda

- Review of Planning I
- Configuration space
- Sampling-based planning
- Obstacle avoidance
- Artificial potential field method
- Planning in practice
- Modified Flood Fill Algorithm

Configuration Space

Workspace and configuration space – Robotic arms

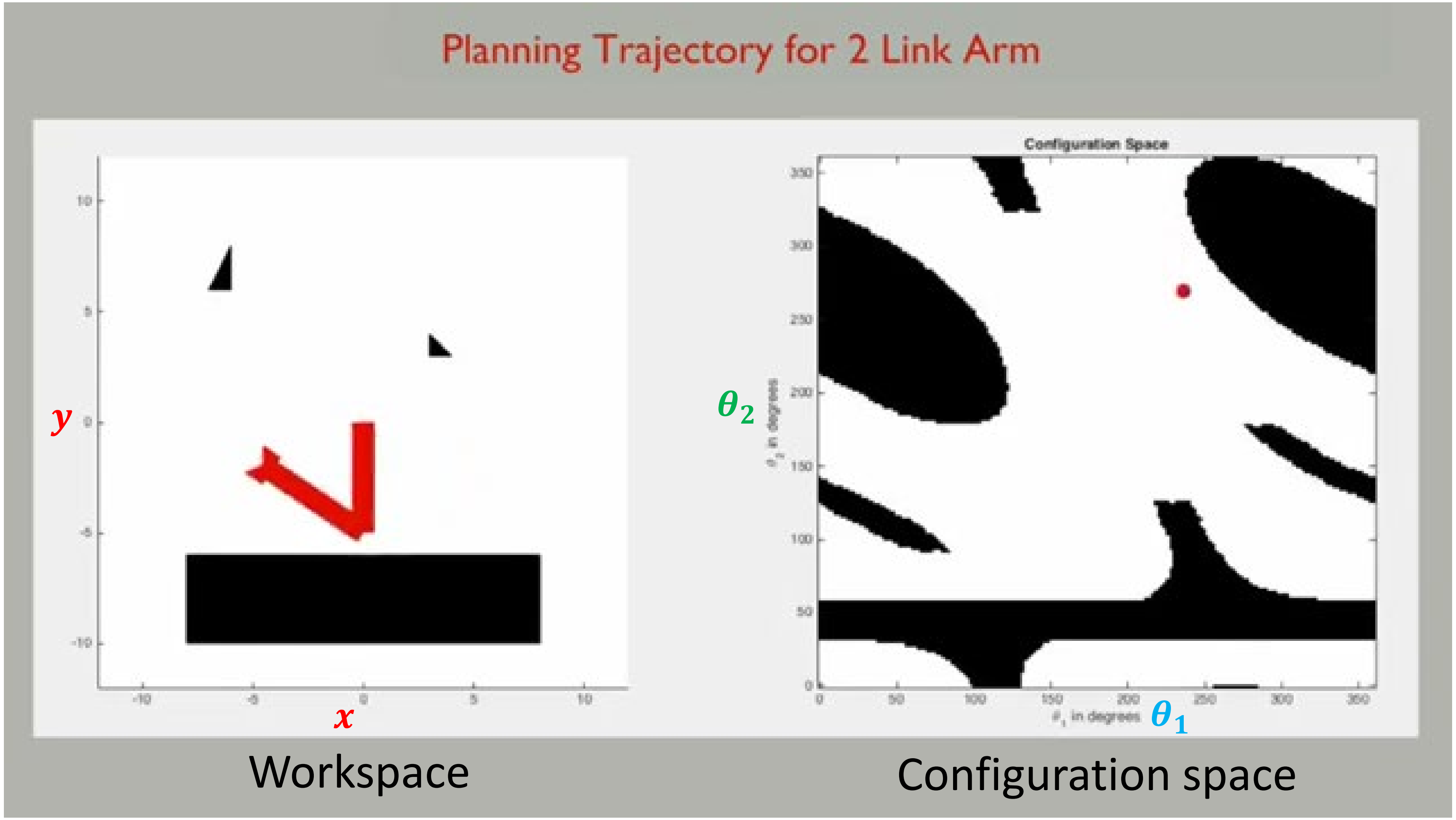


Workspace

Configuration space

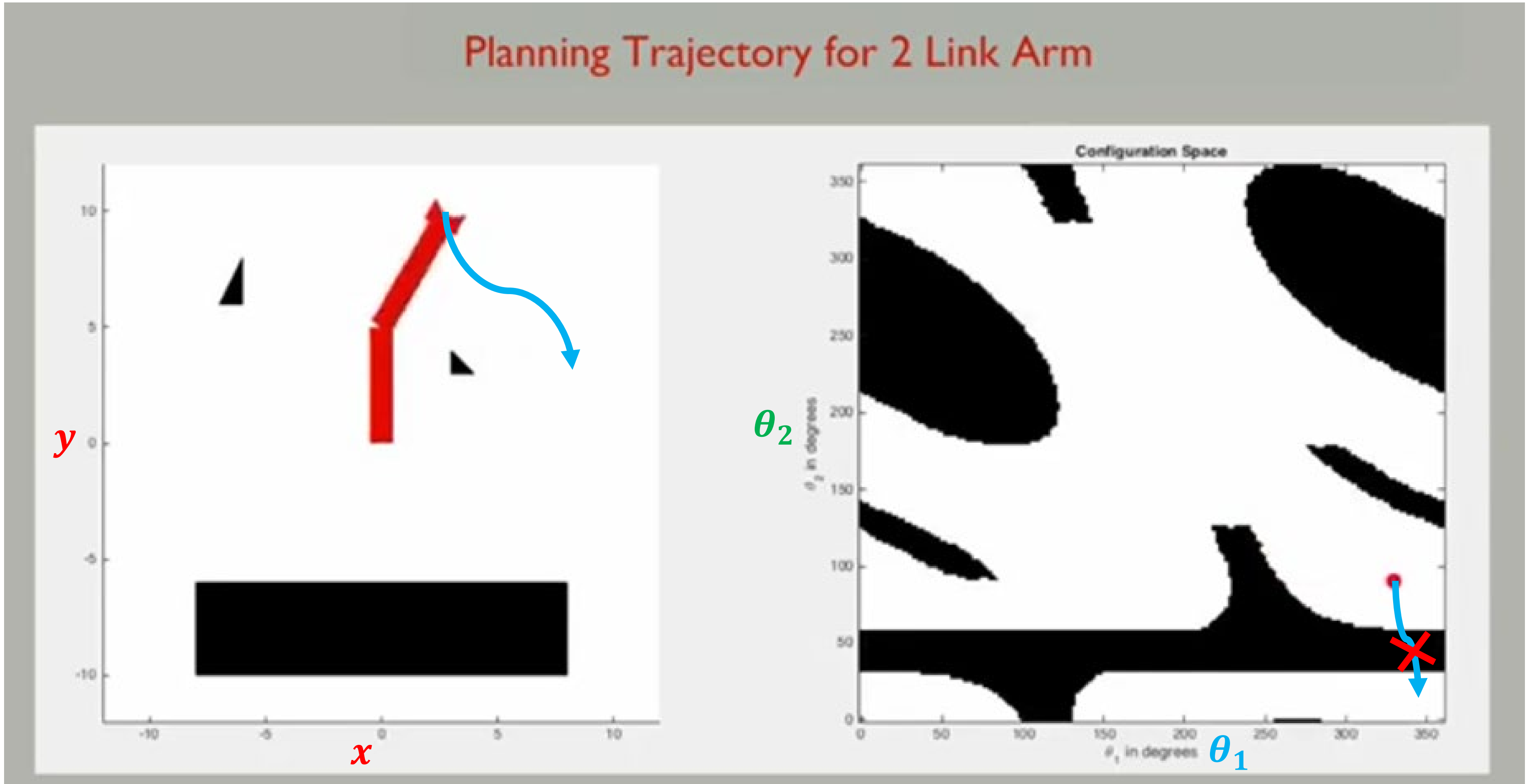
https://www.youtube.com/watch?v=5TS57-AI5LA

Workspace and configuration space – Robotic arms



https://www.youtube.com/watch?v=5TS57-AI5LA

Workspace and configuration space – Robotic arms



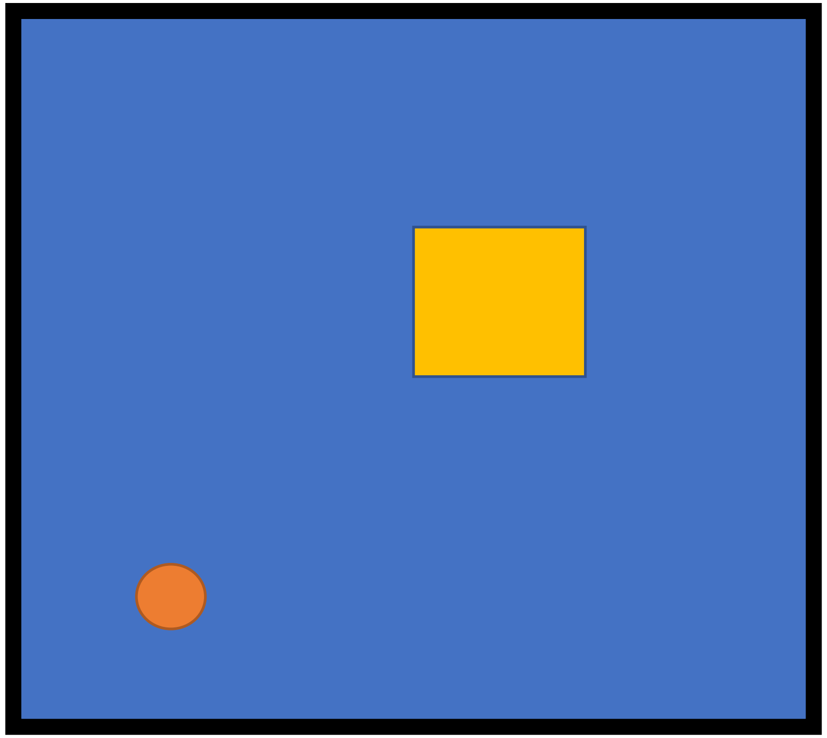
Workspace

Configuration space

https://www.youtube.com/watch?v=5TS57-AI5LA

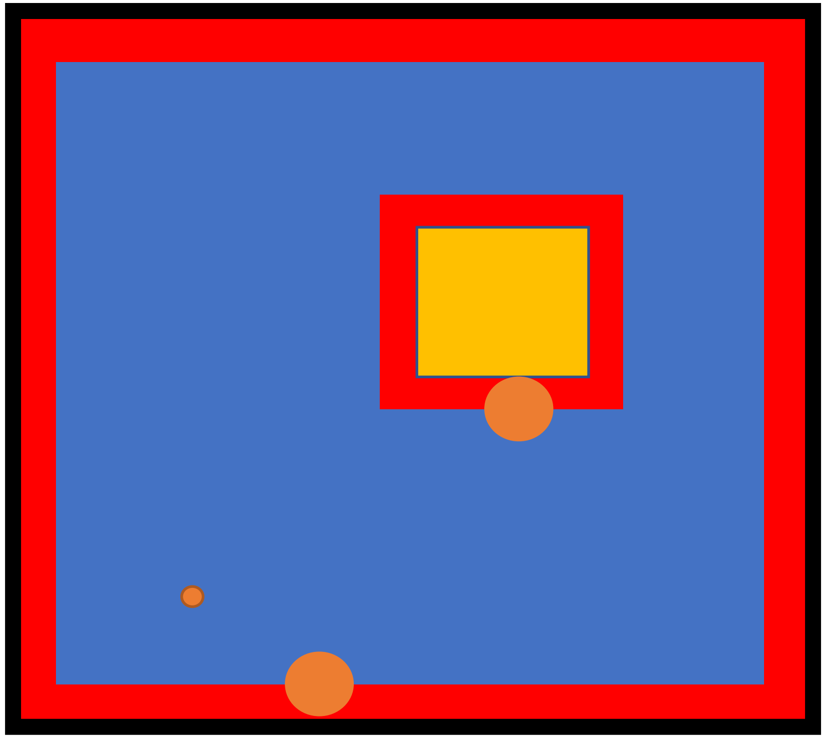
Workspace and configuration space – Planar mobile robots

For omnidirectional and differential-drive robots and for the purpose of path planning, we can approximately simplify the robot as a point moving on a plane (ignoring orientation), and thus simplify the configuration space to be 2D (x and y).



Workspace (3D – x, y, θ)

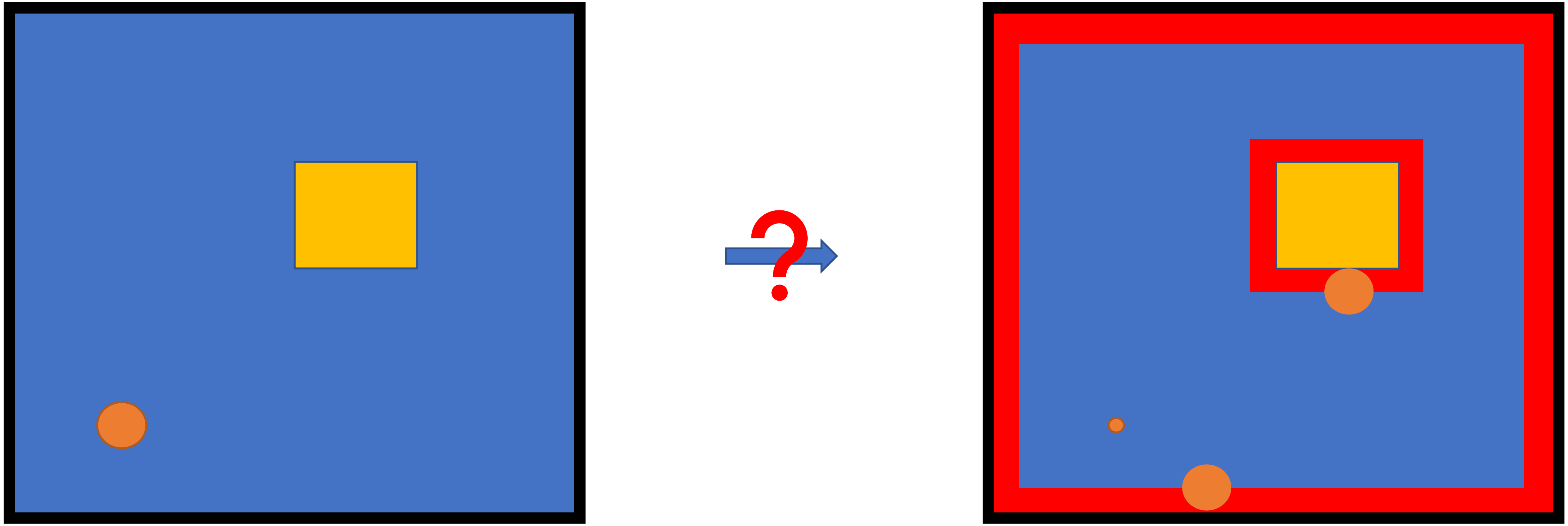
Very similar with shrunk size



Configuration space (2D – x, y)

Workspace and configuration space – Planar mobile robots

For Akerman robot, the configuration space is much more complex as the rotation cannot be decoupled from the linear motions and thus cannot be ignored. More careful considerations are needed for the path planning.



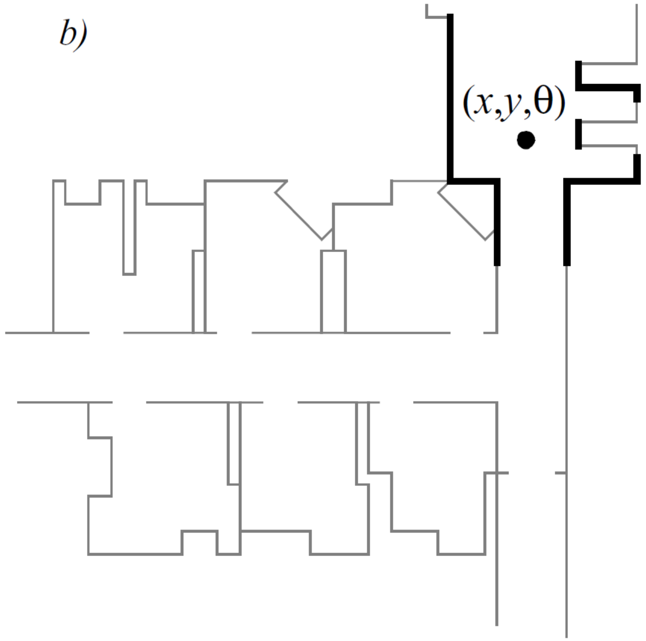
Workspace (3D – x, y, θ)

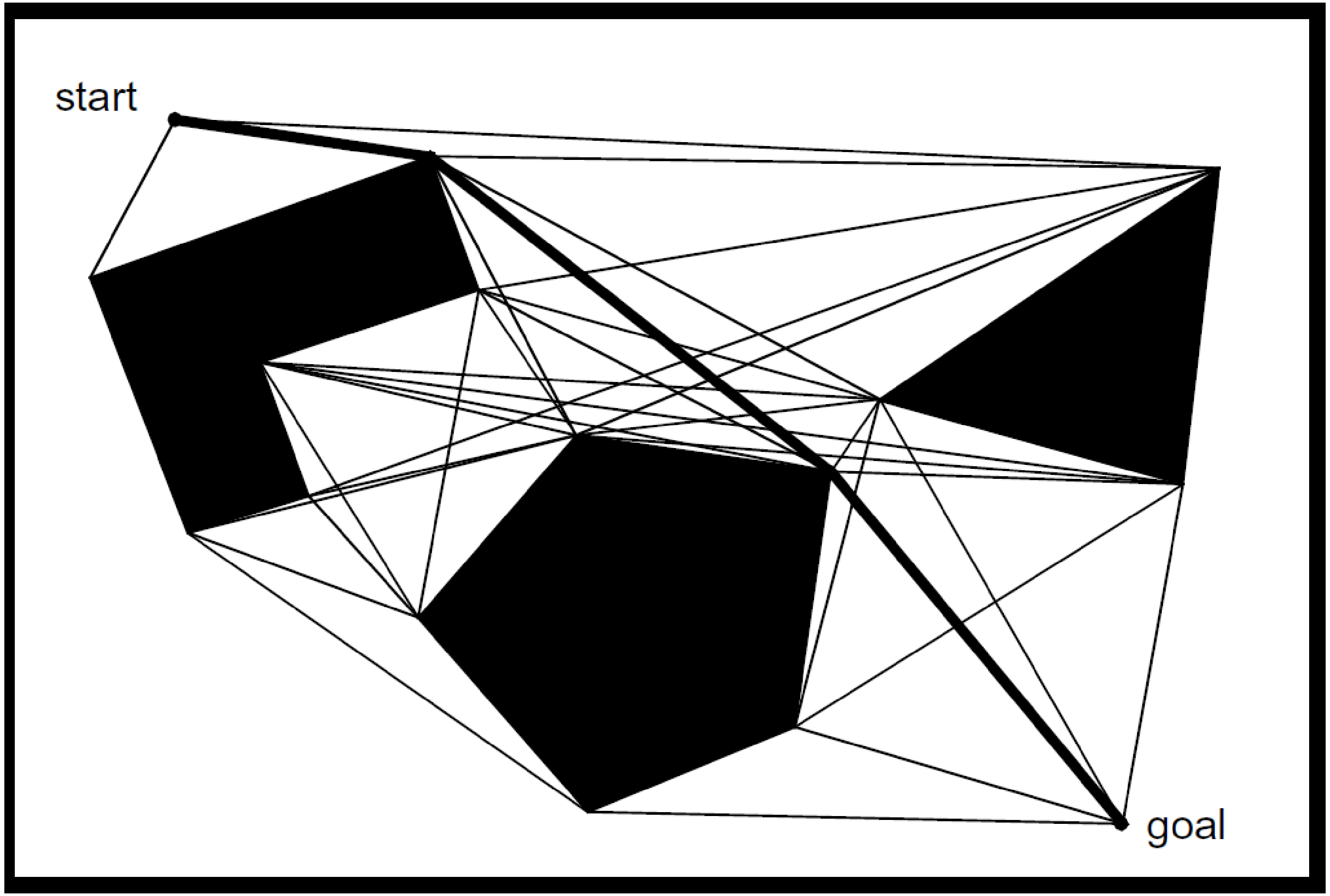
Configuration space (2D – x, y)

Visibility graph – Connect all the vertices visible to each other

Short but not safe

Continuous Map





Latombe, J.-C., *Robot Motion Planning*. Norwood, MA, Kluwer Academic Publishers, 1991.
R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to autonomous mobile robots. The MIT Press. Second edition. 2011.

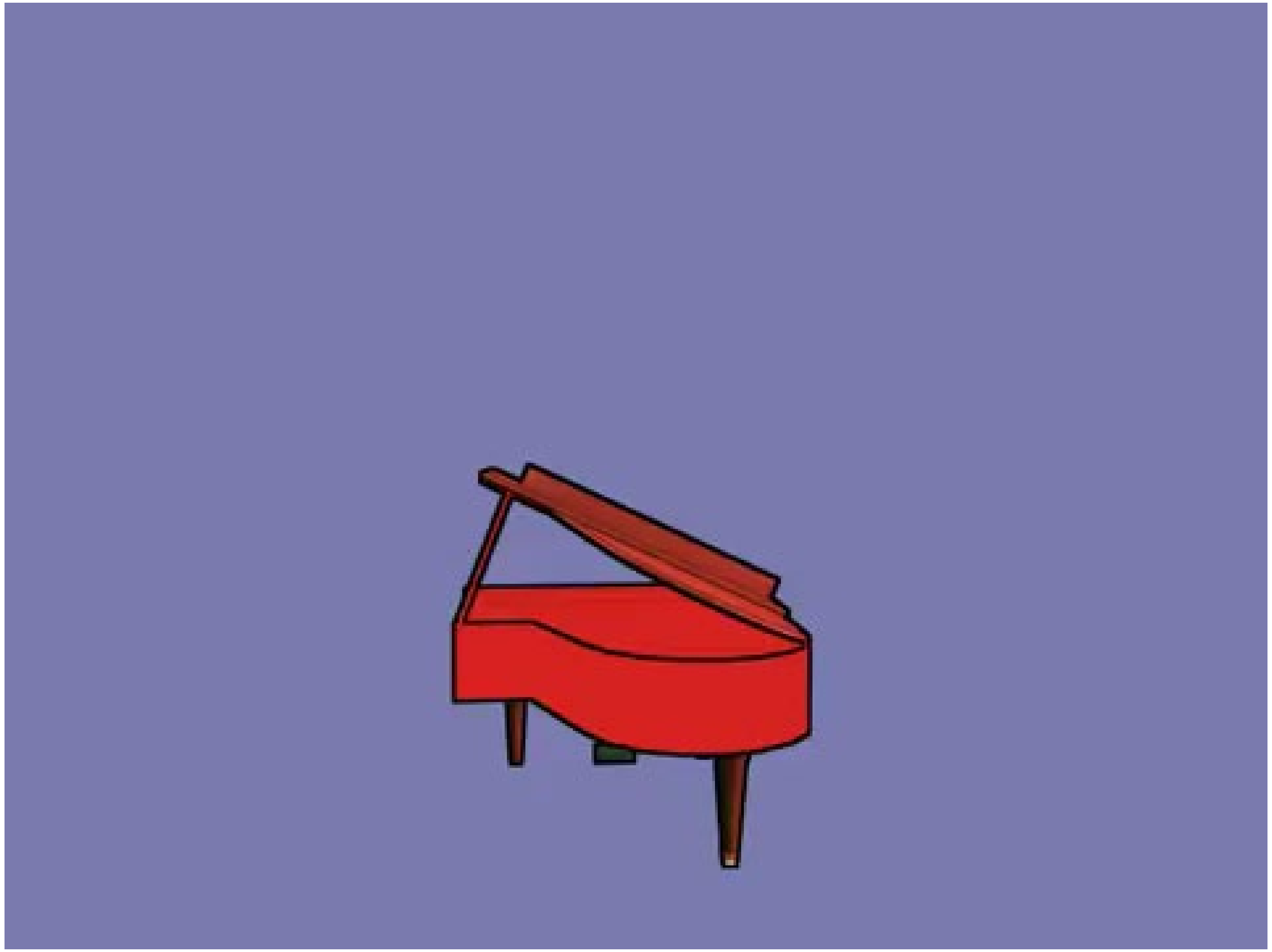
Sampling-based planning

Motivation for sampling-based planning

- Graph search methods rely on an explicit representation of the obstacles in the configuration space

- This may result in an excessive computational burden
  - High-dimensions
  - Environments with a large number of obstacles

Piano Mover's Problem



https://www.youtube.com/watch?v=cXm3WW-geD8

Motivation for sampling-based planning

- Graph search methods rely on an explicit representation of the obstacles in the configuration space

- This may result in an excessive computational burden
  - High-dimensions
  - Environments with a large number of obstacles

- Sampling-based planning – Avoid using an explicit representation of the environment by involving a collision-checking module

- The two most influential sampling-based motion planning algorithms
  - Probabilistic Roadmaps (PRM, 1996)
  - Rapidly-exploring Random Trees (RRT, 1998)

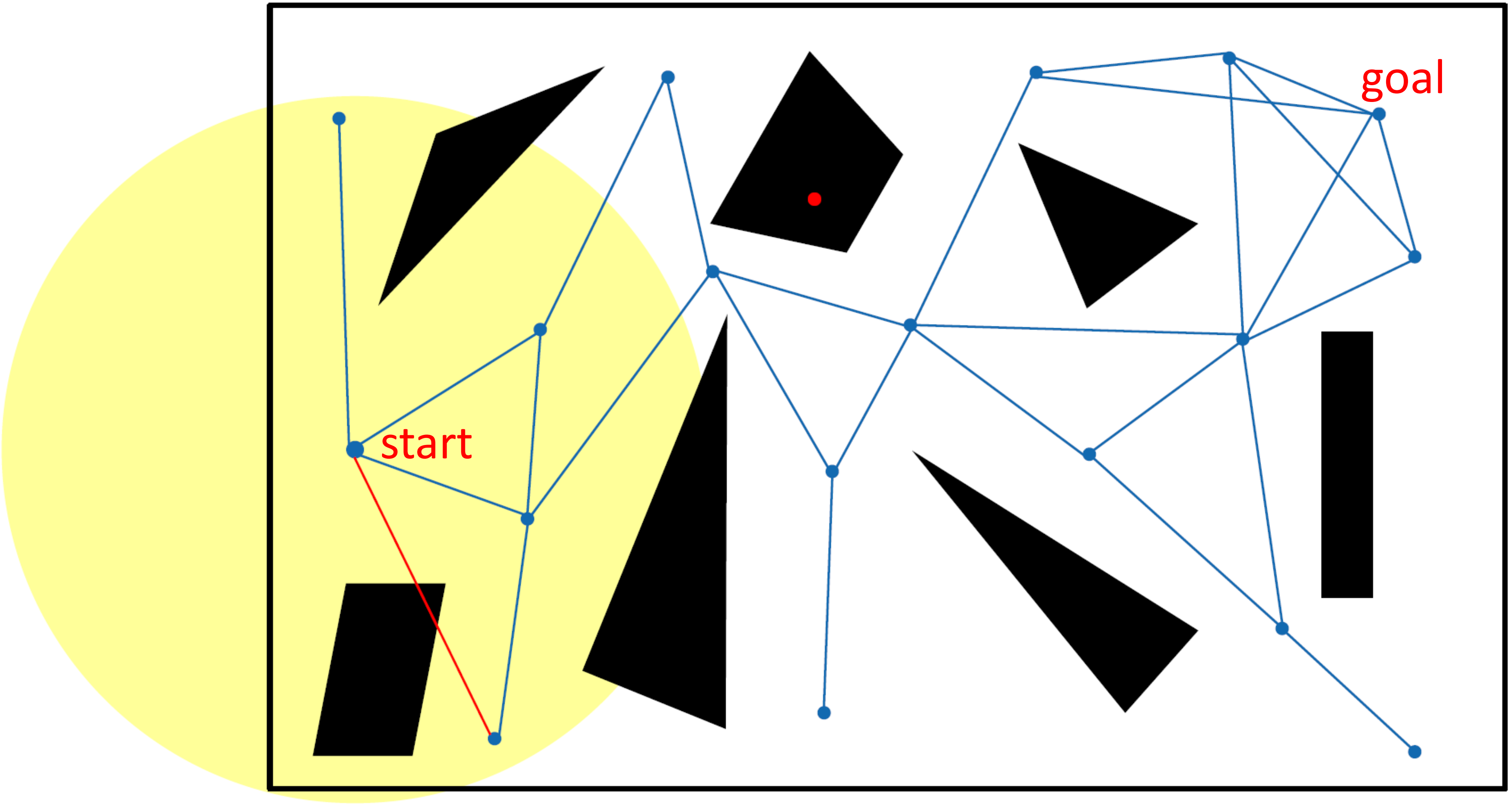
PRM

Probabilistic roadmaps (PRM)

- Principle: Sample free-space configurations

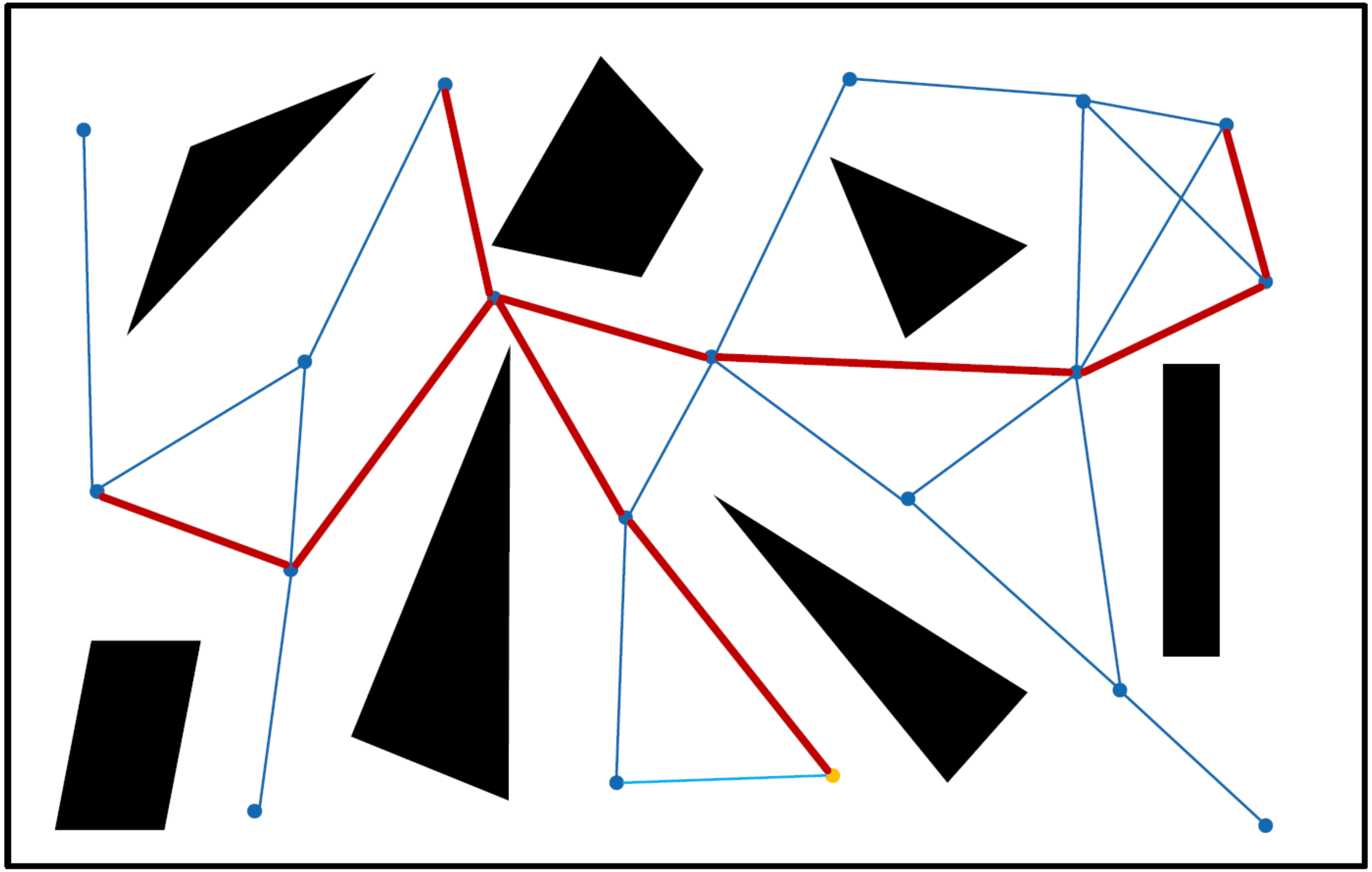
- Two steps:
  - 1. Building a roadmap by connecting nearby (sampled) configurations using simple planners to construct a graph of valid path segments
  - 2. Query: Search the graph using a graph search technique (e.g., A*)

PRM Step 1 – Building the roadmap in the configuration space

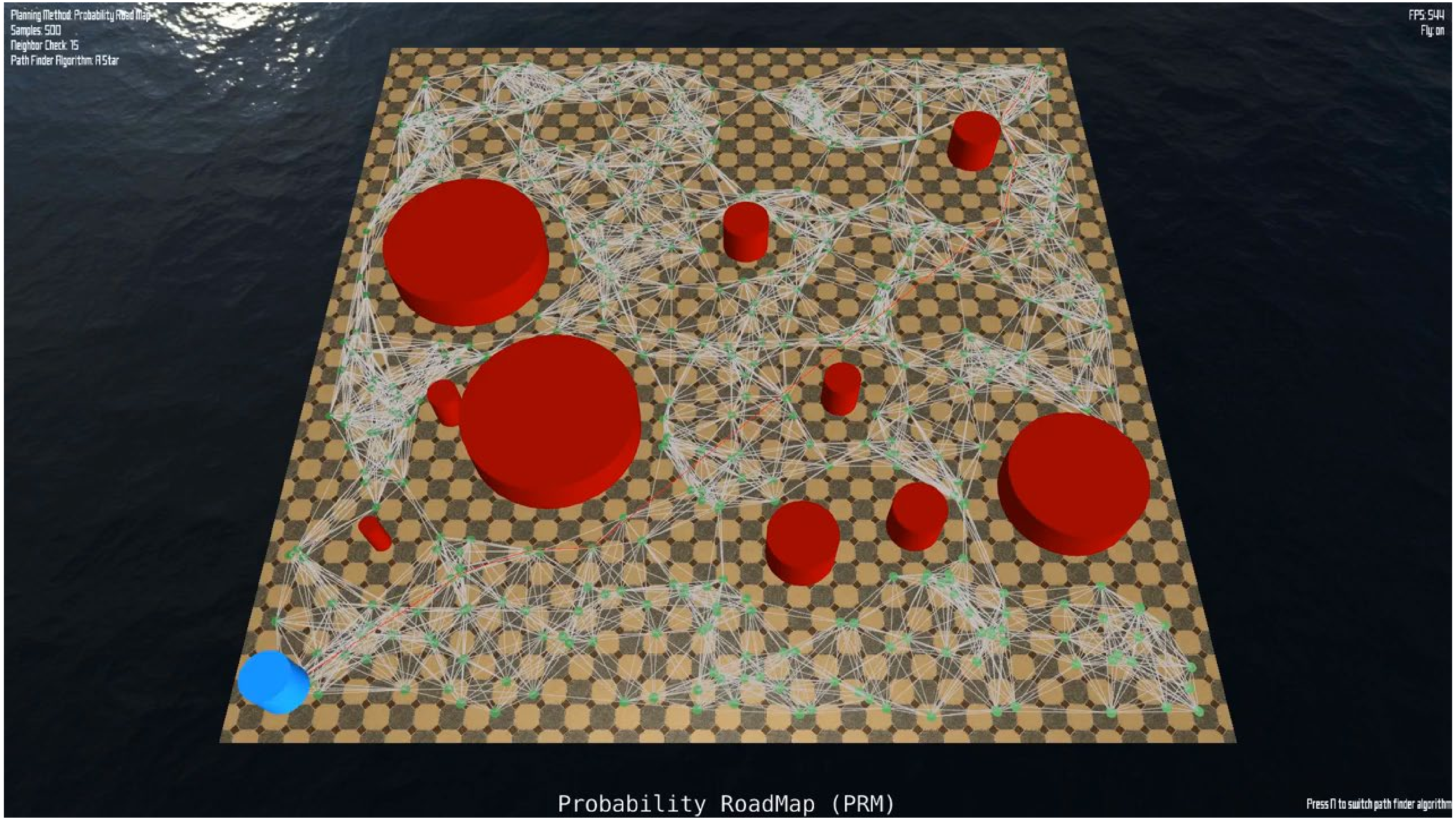


1. Random sampling
2. Check if sample is in free space
3. Connect neighbouring samples and check if collision happens

PRM Step 2 – Graph search



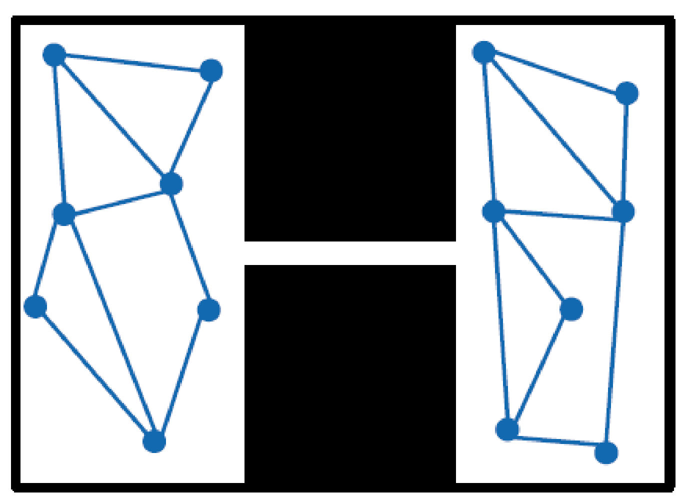
PRM - Example



https://www.youtube.com/watch?v=WgelhGE1KLg

PRM - Summary

Advantages	Disadvantages
• Conceptually very simple • Able to solve high-dimension planning	• Can suffer from "narrow passage problem" • Not suited for dynamic environments • Assumes holonomic motion



PRM assumes holonomic motion

https://www.youtube.com/watch?v=Tdmm3i52WBc

RRT

Rapidly-exploring Random Trees (RRT)

- Instead of constructing a graph with all points sampled and then converting it to a tree for graph search,
- Incrementally construct a search tree that gradually improves the resolution.

- Steps:
  - 1. Start with a root node
  - 2. Employs an expansion heuristic toward the goal state
  - 3. Connect node to tree if path is collision free
  - 4. Graph search

RRT - Algorithm

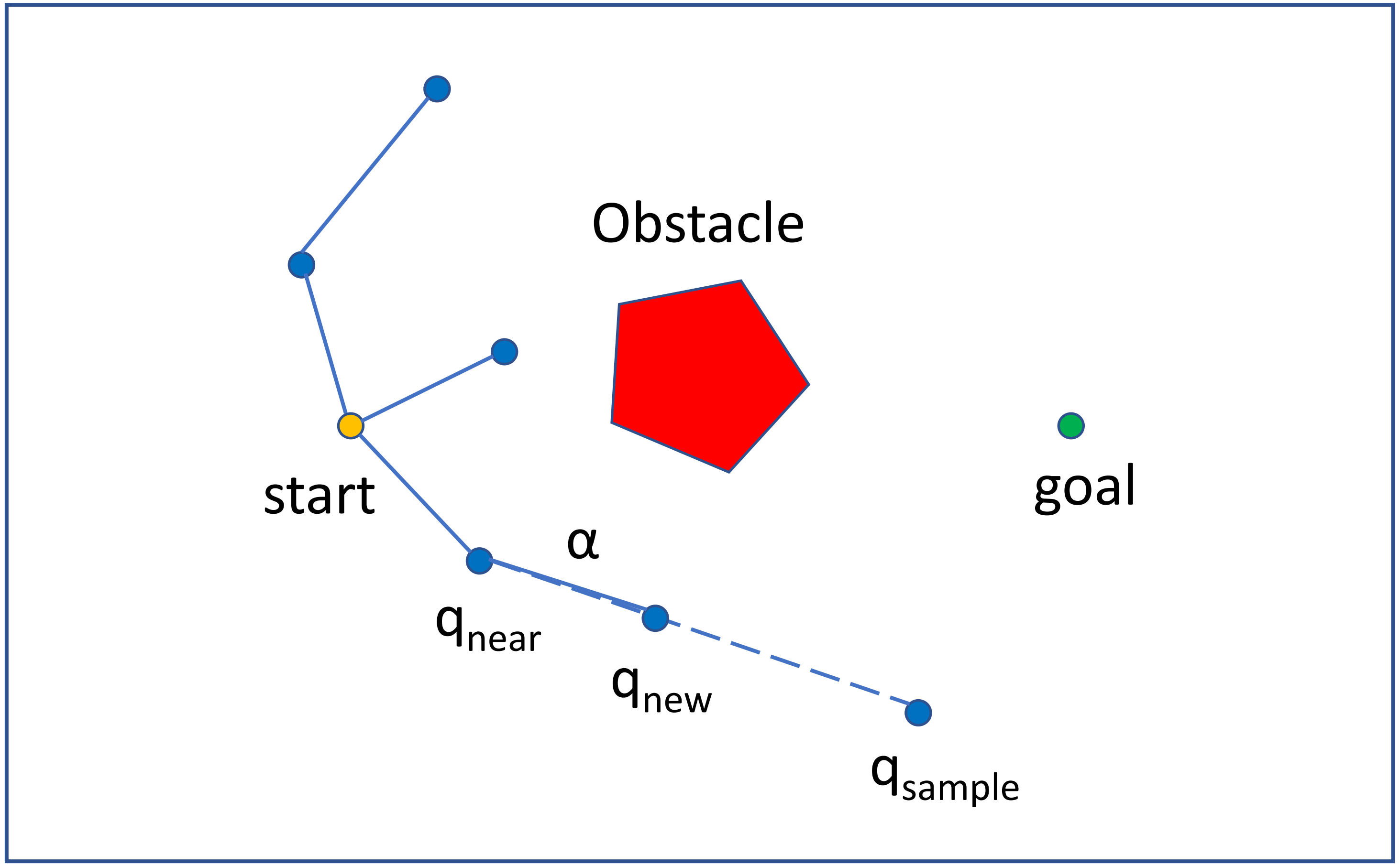
Input: q_{start}, q_{goal}, number n of nodes, stepsize α, β

Output: tree $T = (V, E)$

initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
for $i = 0 : n$ **do**
    if rand$(0, 1) < \beta$ **then** $q_{\text{target}} \leftarrow q_{\text{goal}}$
    else $q_{\text{target}} \leftarrow$ random sample from Q
    $q_{\text{near}} \leftarrow$ nearest neighbor of q_{target} in V
    $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|}(q_{\text{target}} - q_{\text{near}})$
    if $q_{\text{new}} \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q_{\text{new}}\}$, $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
end for

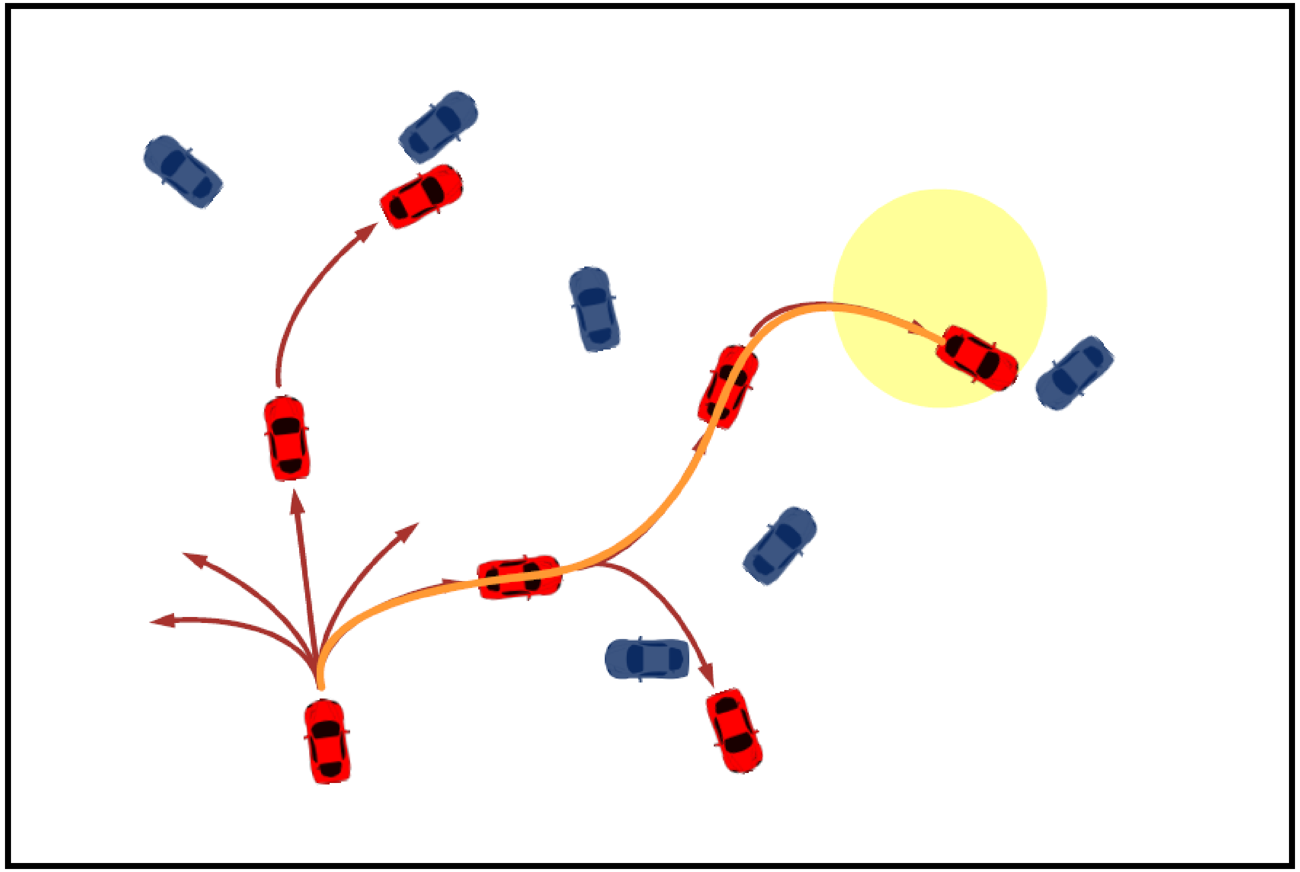
RRT - Illustration

What if using a nonlinear interpolation when adding a new node?



RRT – For nonholonomic motions

- Instead of linear interpolation, use nonholonomic constraints for growing the tree



Nick Lawrance, https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/asl-dam/documents/lectures/autonomous_mobile_robots/spring-2019/Planning_I_2019.pdf

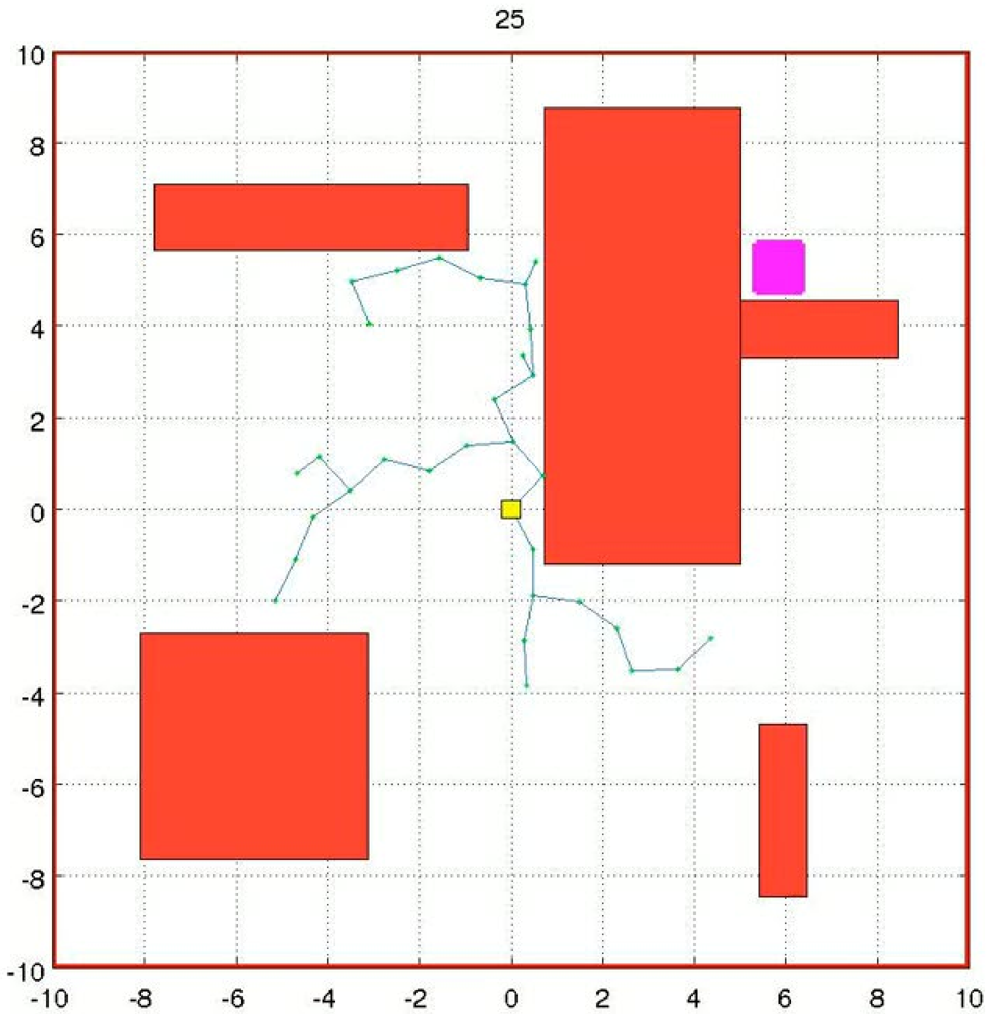
RRT - Example

- After many iterations, the tree will be dense and eventually get to any point in the space

45 iterations

2345 iterations

RRT – Example



RRT*

Asymptotically Optimal Rapidly-exploring Random Tree (RRT*)

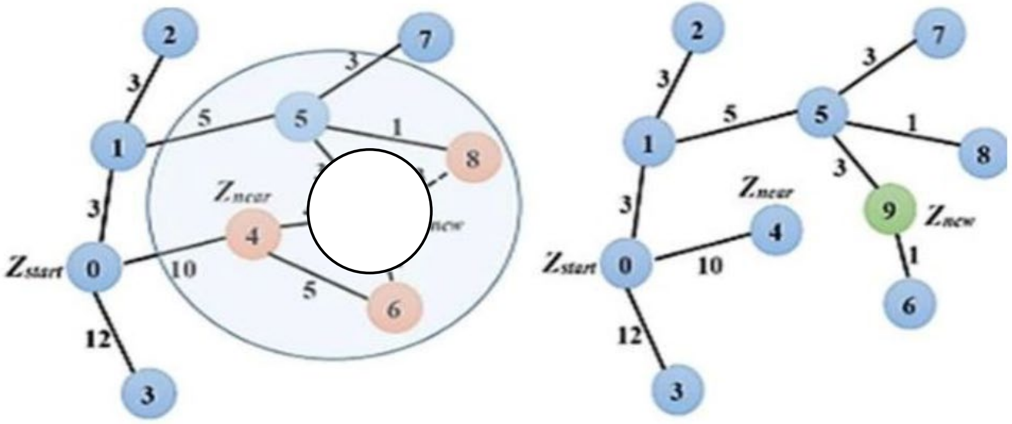
- Asymptotically optimal
  - The cost of the returned solution will approach the optimal as the number of samples -> ∞
- RRT* = An "optimal" version of RRT
- BTW, why is A* called "A*"?
  - *"In 1964 Nils Nilsson invented a heuristic based approach to increase the speed of Dijkstra's algorithm. This algorithm was called A1. In 1967 Bertram Raphael made dramatic improvements upon this algorithm, but failed to show optimality. He called this algorithm A2. Then in 1968 Peter E. Hart introduced an argument that proved A2 was optimal when using a consistent heuristic with only minor changes. His proof of the algorithm also included a section that showed that the new A2 algorithm was the best algorithm possible given the conditions.*
  - *He thus named the new algorithm in Kleene star syntax to be the algorithm that starts with A and includes all possible version numbers or A*."*

Kleene star syntax: $V^* = \bigcup_{i\geq 0} V^i = V^0 \cup V^1 \cup V^2 \cup V^3 \cup V^4 \cup \cdots$

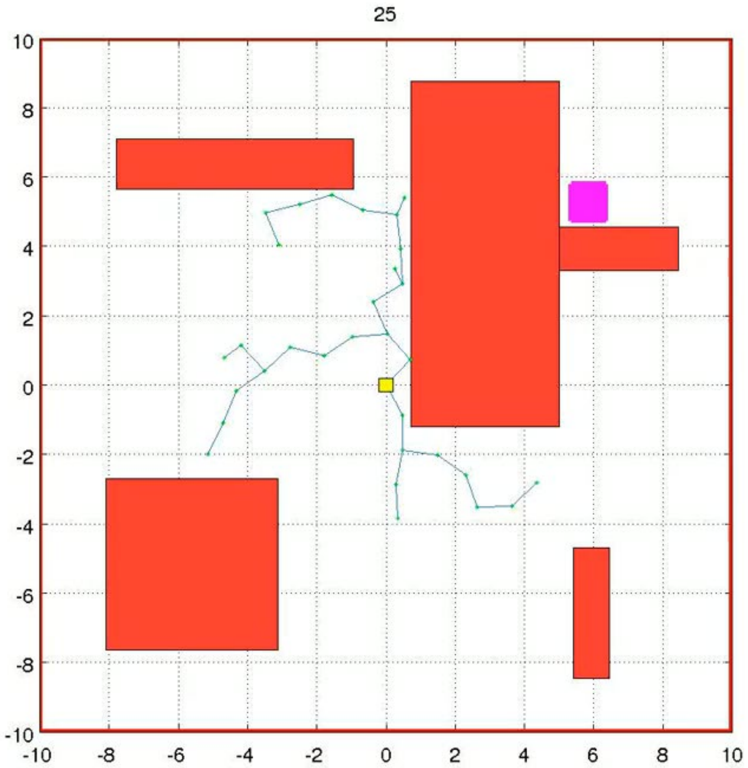
Nosrati, Masoud, Ronak Karimi, and Hojat Allah Hasanvand. "Investigation of the (star) search algorithms: Characteristics, methods and approaches." World Applied Programming 2.4 (2012): 251-256.
https://en.wikipedia.org/wiki/Kleene_star

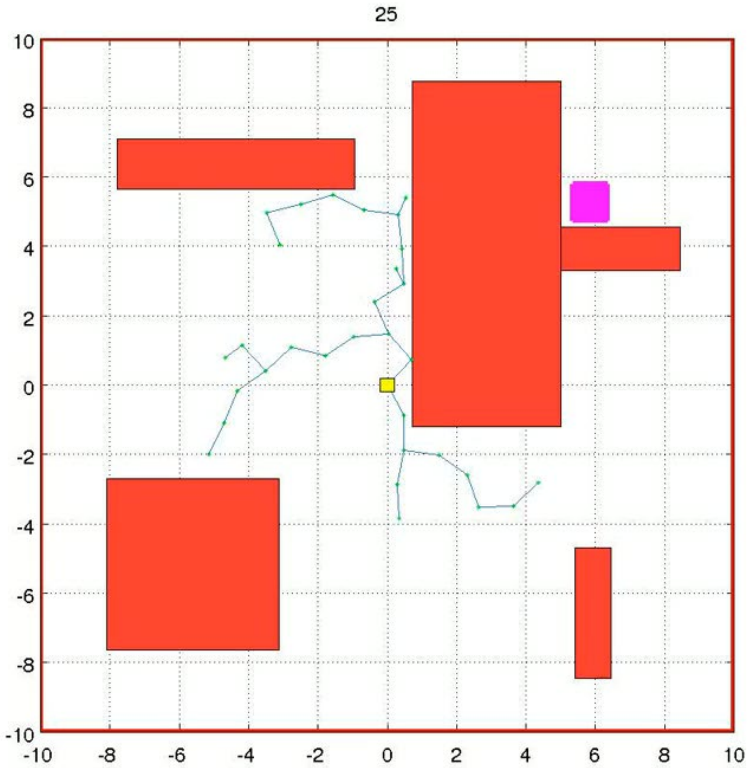
RRT*

- Inherits all the properties of RRT and works similar to RRT.
- However, it introduced two promising features
  - Nearest neighbour search
    - Finds the best (lowest cost) parent node for the new node before its insertion in tree
  - Rewiring
    - Rebuilds the tree within this radius of area k to maintain the tree with minimal cost between tree connections



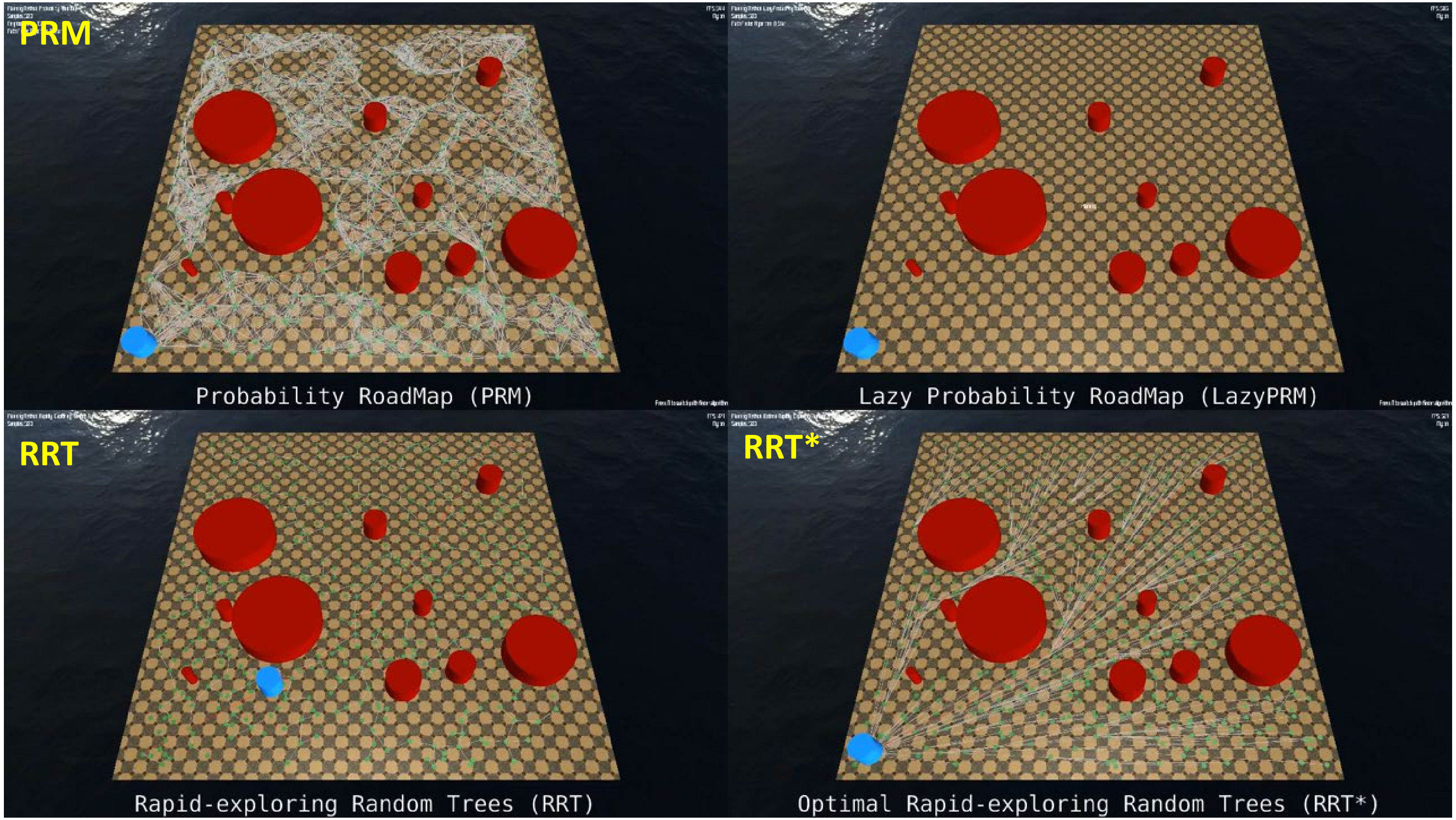
RRT* vs RRT





https://www.youtube.com/watch?v=YKiQTJpPFkA

PRM vs RRT vs RRT*

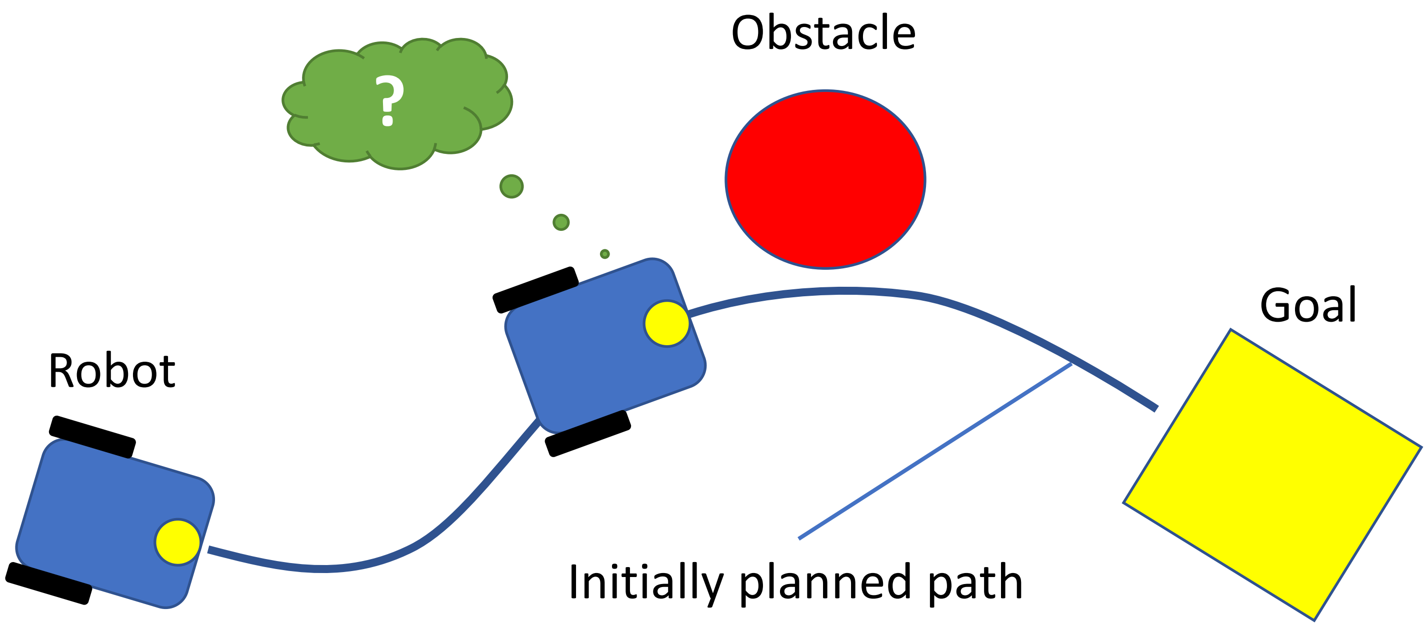


https://www.youtube.com/watch?v=WgelhGE1KLg

Obstacle Avoidance

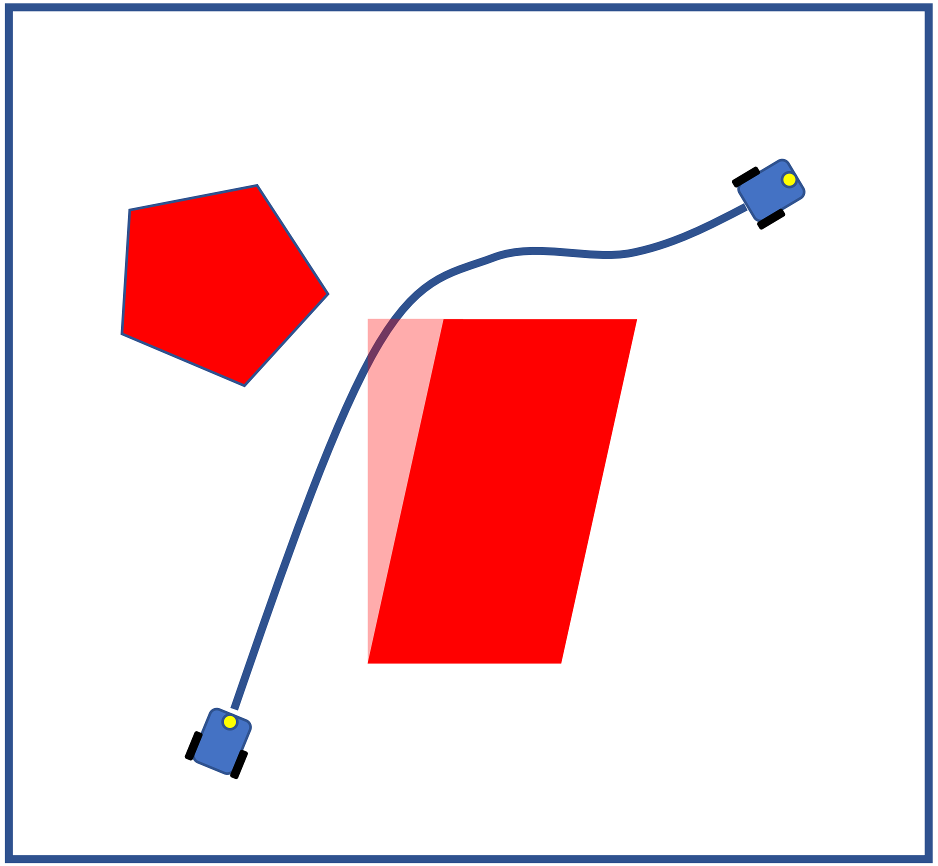
Obstacle avoidance

- The ability to replan the path when encountering an obstacle

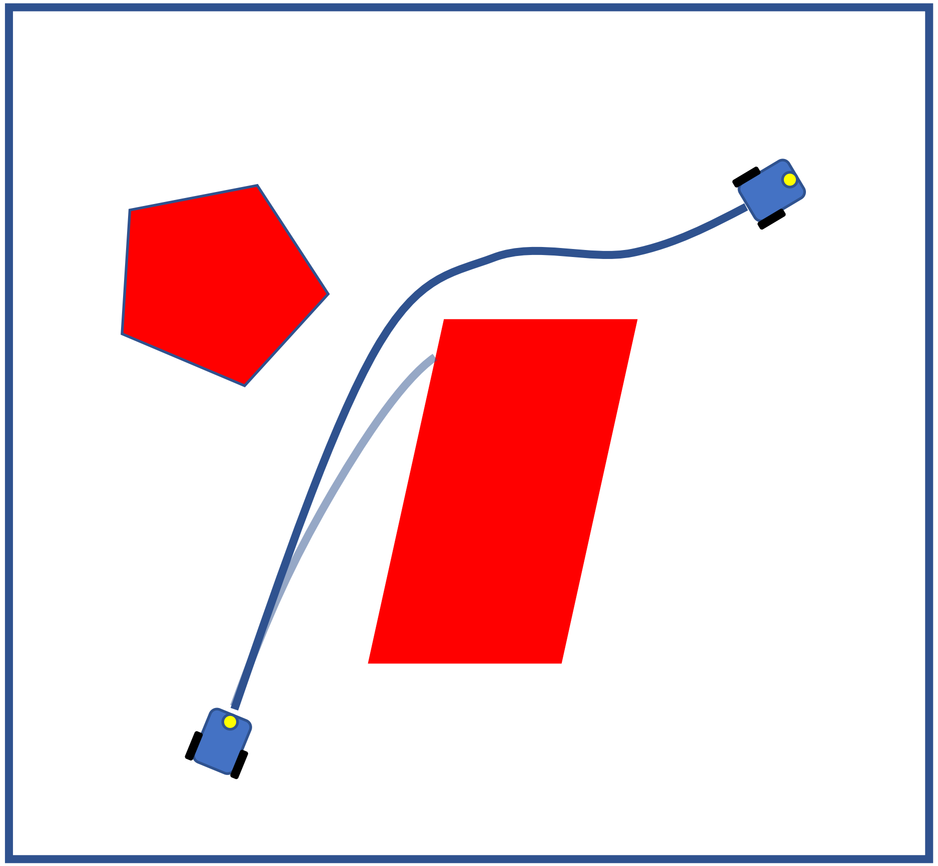


Why obstacle avoidance is needed?

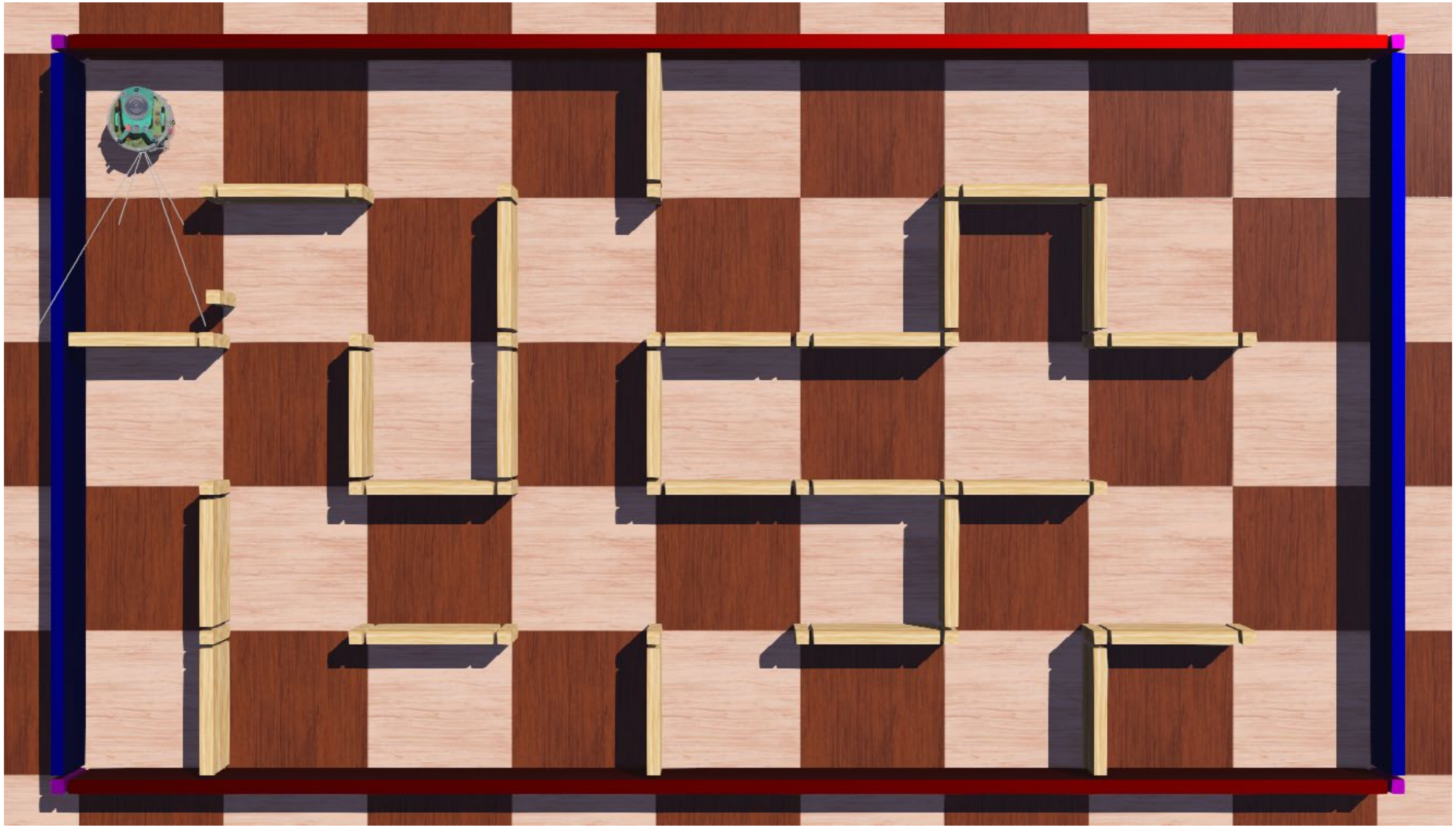
- Environment uncertainty



- Motion uncertainty

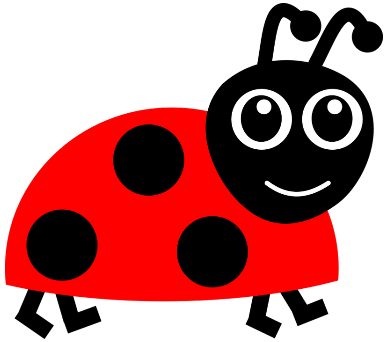


Why obstacle avoidance is needed? – Another example



Obstacle avoidance – The Bug algorithms

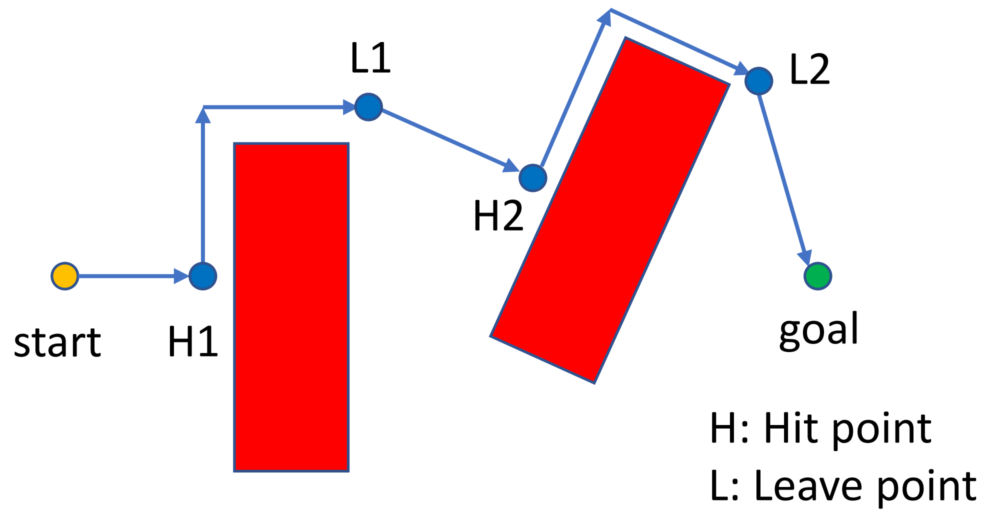
- Bug 0
- Bug 1
- Bug 2
- Tangent Bug
- …



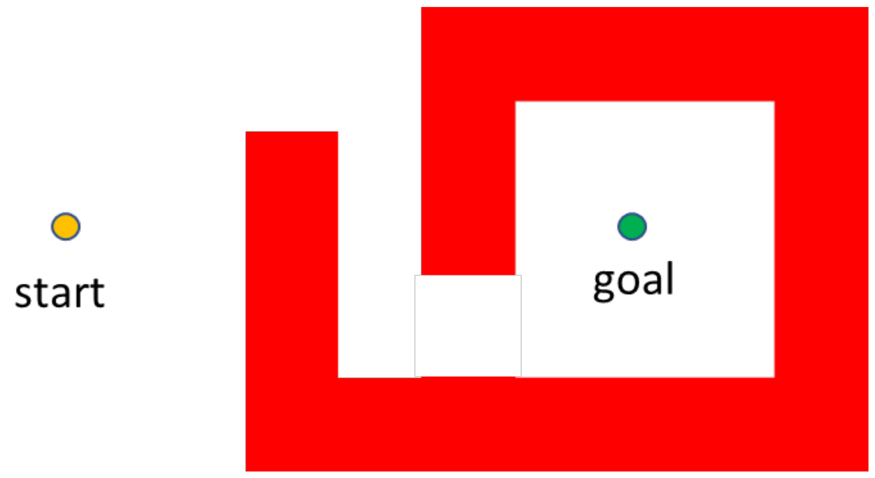
https://pixabay.com/vectors/ladybird-ladybug-insect-luck-red-297875/

Bug 0 Algorithm

- Known direction to goal
- Only tactile sensors (or equivalent)
- Repeat until goal is reached
  - Head towards goal
  - If sensor reports contact with an obstacle then
    - Follow obstacle boundary
      until can head towards goal again

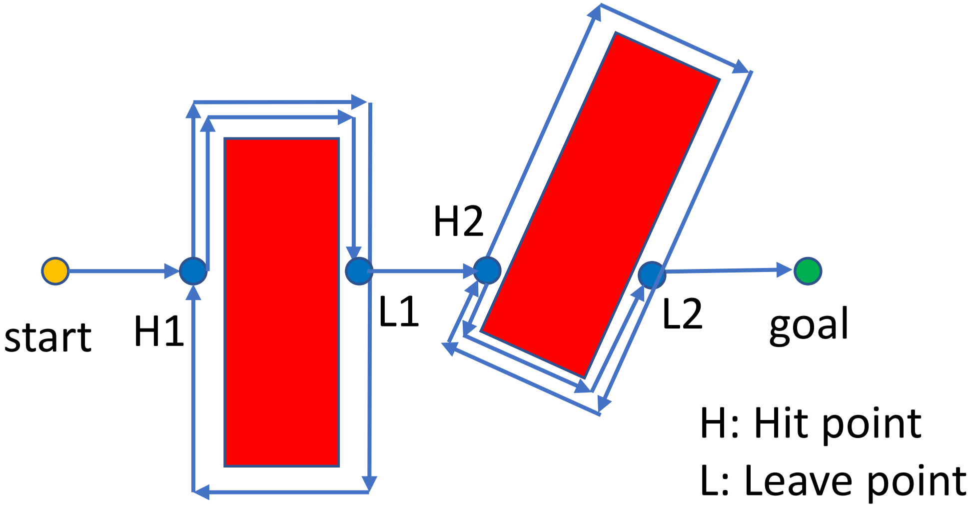


- Is Bug 0 complete?
  - Complete: Always find a valid path if there exists one



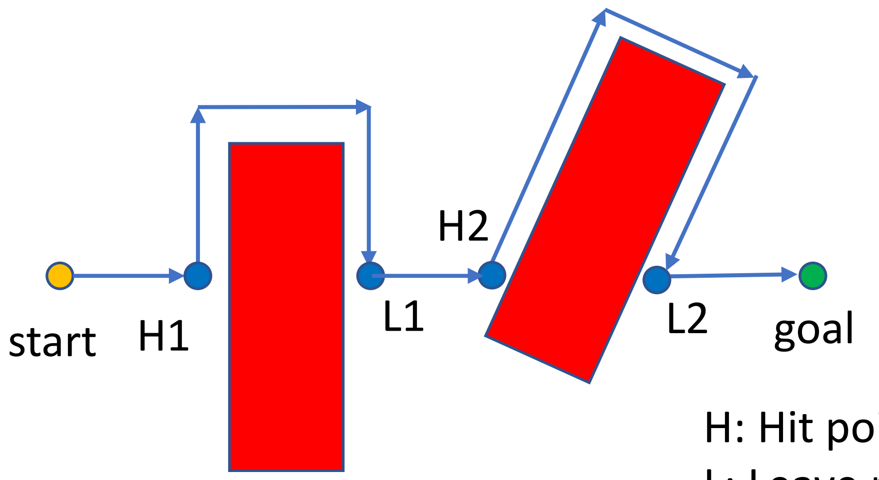
Bug 1 Algorithm

- Known direction to goal
- Tactile sensors (or equivalent)
- Encoders (or equivalent)
- With some additional memory and computing power
- Repeat until goal is reached
  - Head towards goal
  - If sensor reports contact with an obstacle then
    - Circumnavigate the obstacle and remember how close you get to the goal
    - Return to that closest point (by wall following) and continue towards goal



Bug 2 Algorithm

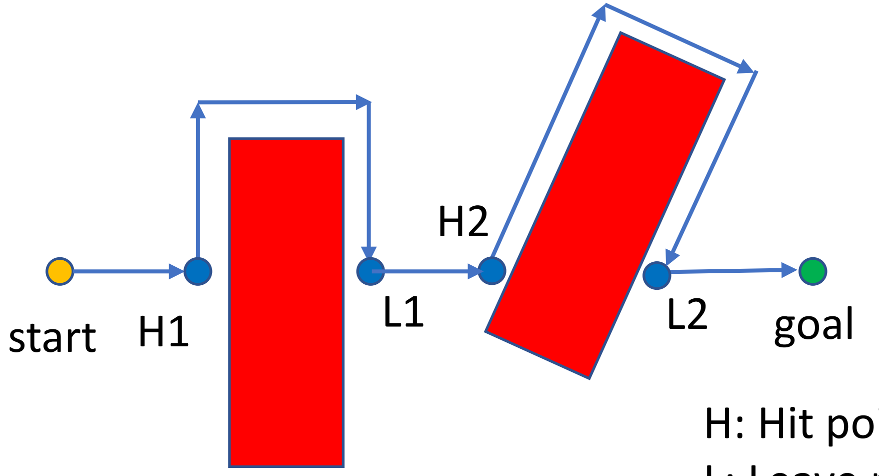
- Known direction to goal
- Tactile sensors (or equivalent)
- Encoders (or equivalent)
- With some additional memory and computing power
- Repeat until goal is reached
  - Head towards goal
  - If sensor reports contact with an obstacle then
    - Follow the obstacle until it encounters the line from start to goal (*m-line*) again
    - Leave the obstacle and continue straight toward goal



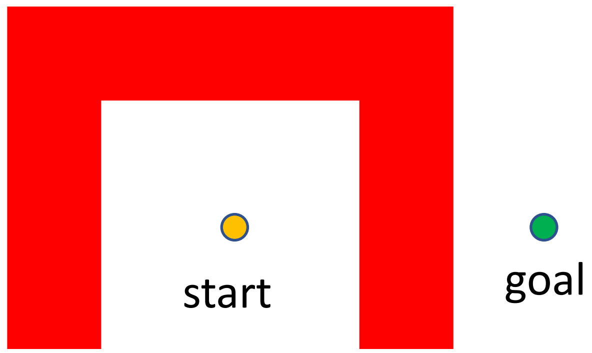
H: Hit point
L: Leave point

Bug 2 Algorithm

- Known direction to goal
- Tactile sensors (or equivalent)
- Encoders (or equivalent)
- With some additional memory and computing power
- Repeat until goal is reached
  - Head towards goal
  - If sensor reports contact with an obstacle then
    - Follow the obstacle until it encounters the line from start to goal (*m-line*) again
    - Leave the obstacle and continue straight toward goal

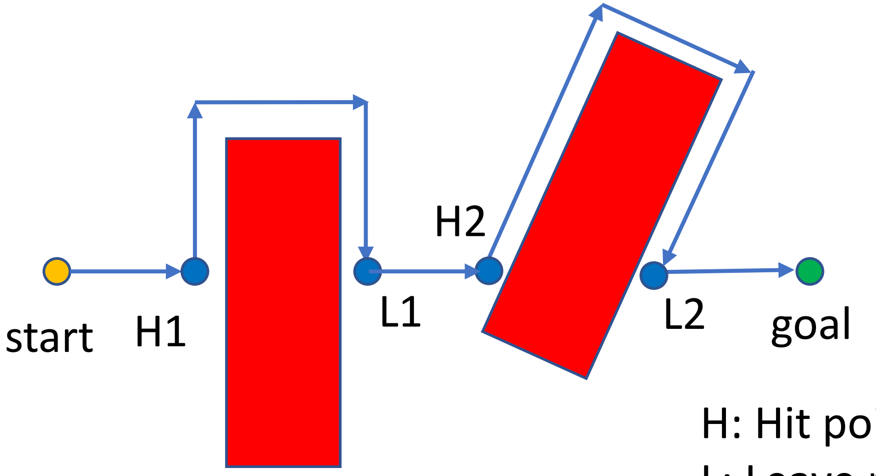


H: Hit point
L: Leave point

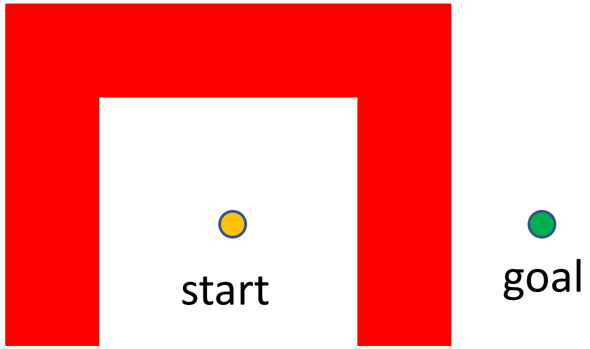


Bug 2 Algorithm

- Known direction to goal
- Tactile sensors (or equivalent)
- Encoders (or equivalent)
- With some additional memory and computing power
- Repeat until goal is reached
  - Head towards goal
  - If sensor reports contact with an obstacle then
    - Follow the obstacle until it encounters the line from start to goal (*m-line*) again closer to the goal
    - Leave the obstacle and continue straight toward goal

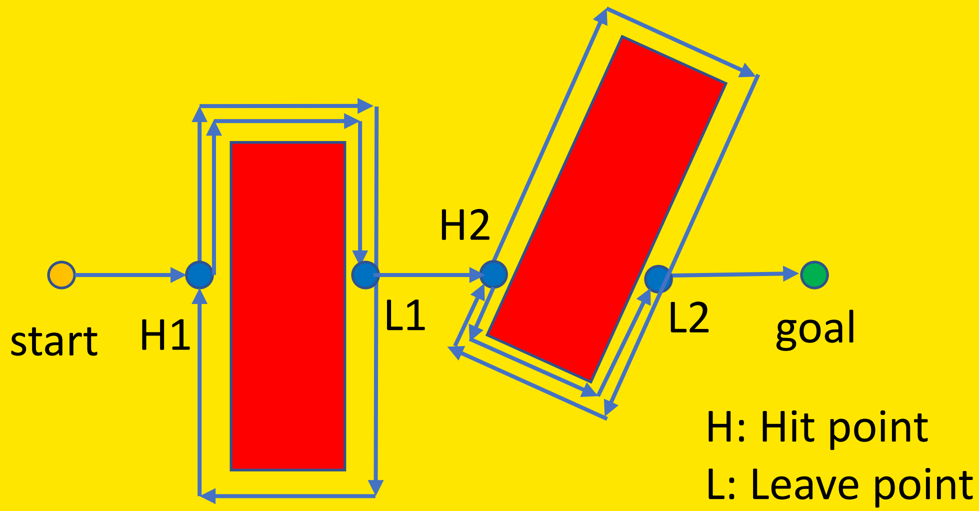


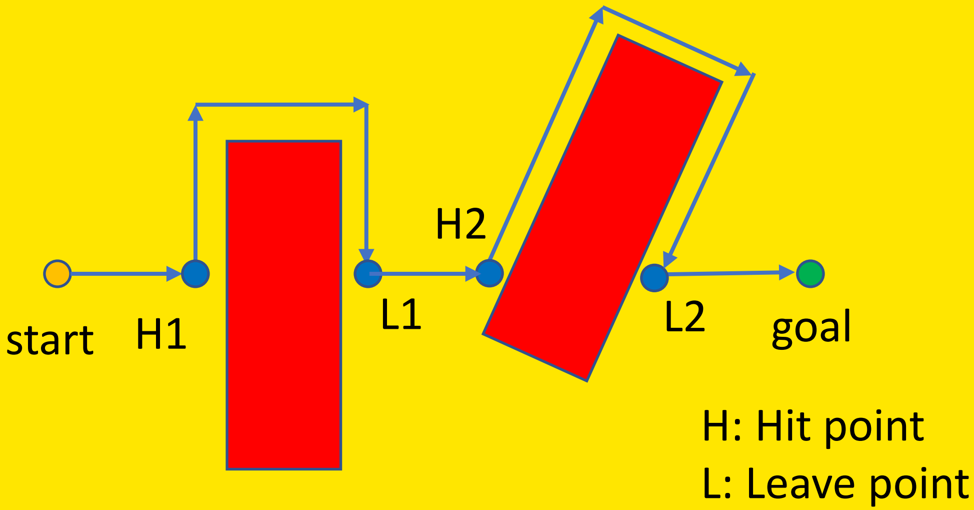
H: Hit point
L: Leave point



https://www.sli.do/ #4110

Which algorithm is better? Bug 1 or Bug 2?





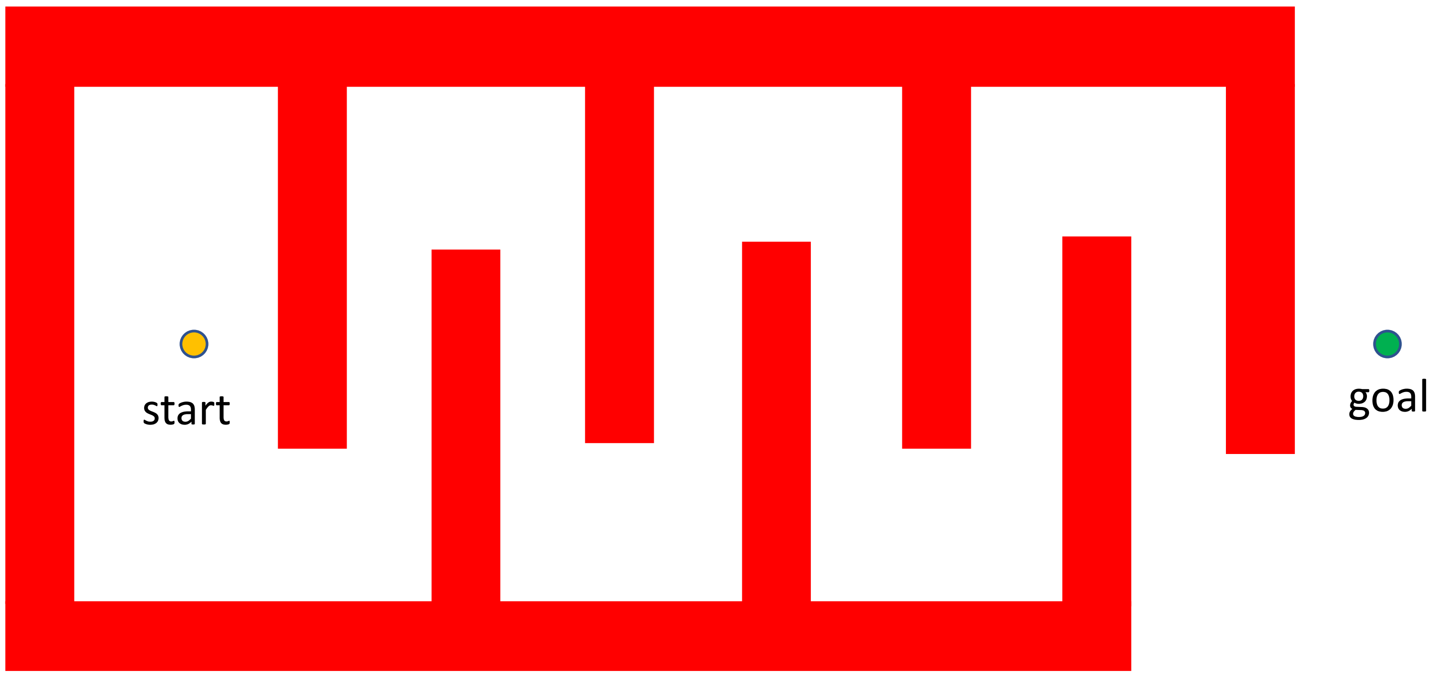
slido

Which algorithm is better? Bug 1 or Bug 2?

ⓘ Start presenting to display the poll results on this slide.

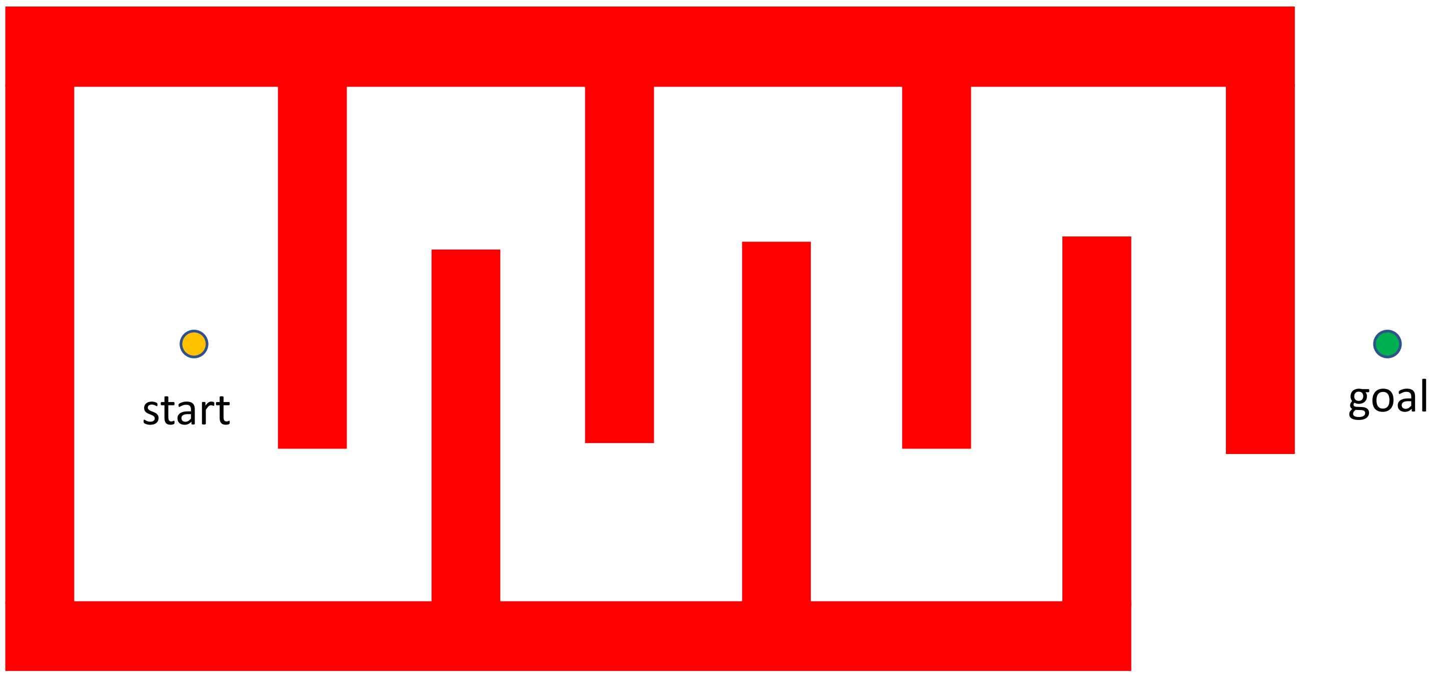
Is Bug 2 always better than Bug 1?

- Bug 1:
  - Circumnavigate the obstacle and remember how close you get to the goal
  - Return to that closest point (by wall following) and continue towards goal
- Bug 2:
  - Follow the obstacle until it encounters the line from start to goal (*m-line*) again closer to the goal
  - Leave the obstacle and continue straight toward goal



Is Bug 2 always better than Bug 1?

- Bug 1:
  - Circumnavigate the obstacle and remember how close you get to the goal
  - Return to that closest point (by wall following) and continue towards goal
- Bug 2:
  - Follow the obstacle until it encounters the line from start to goal (*m-line*) again closer to the goal
  - Leave the obstacle and continue straight toward goal

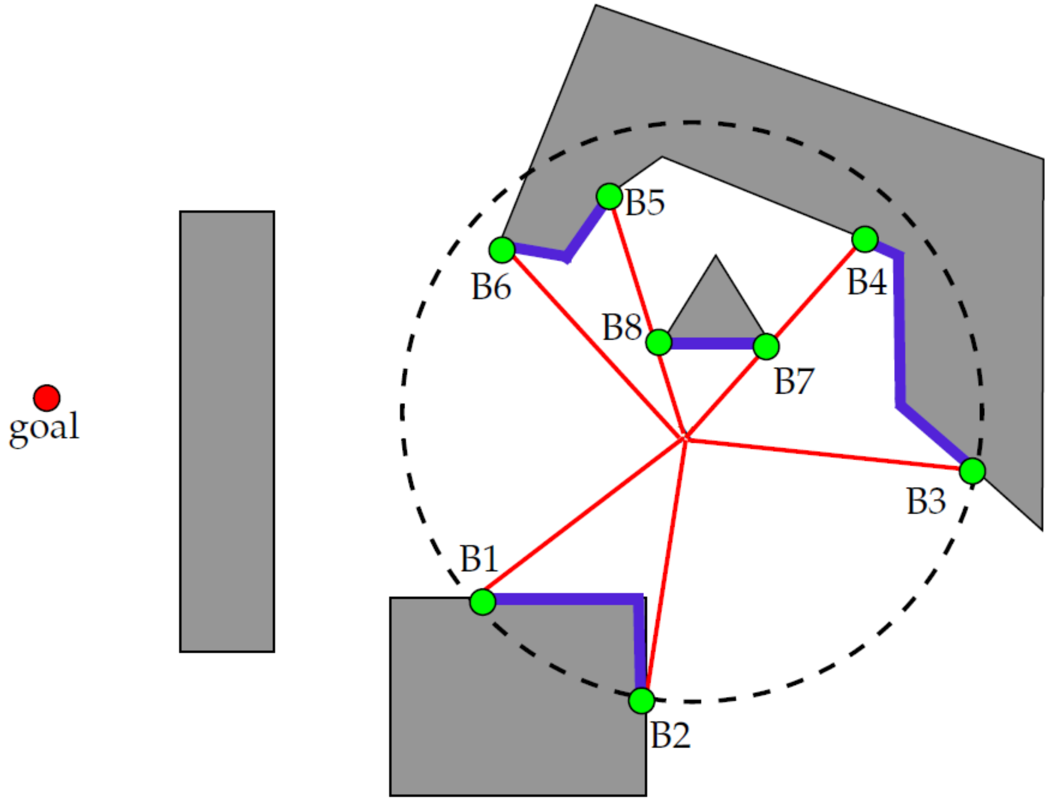


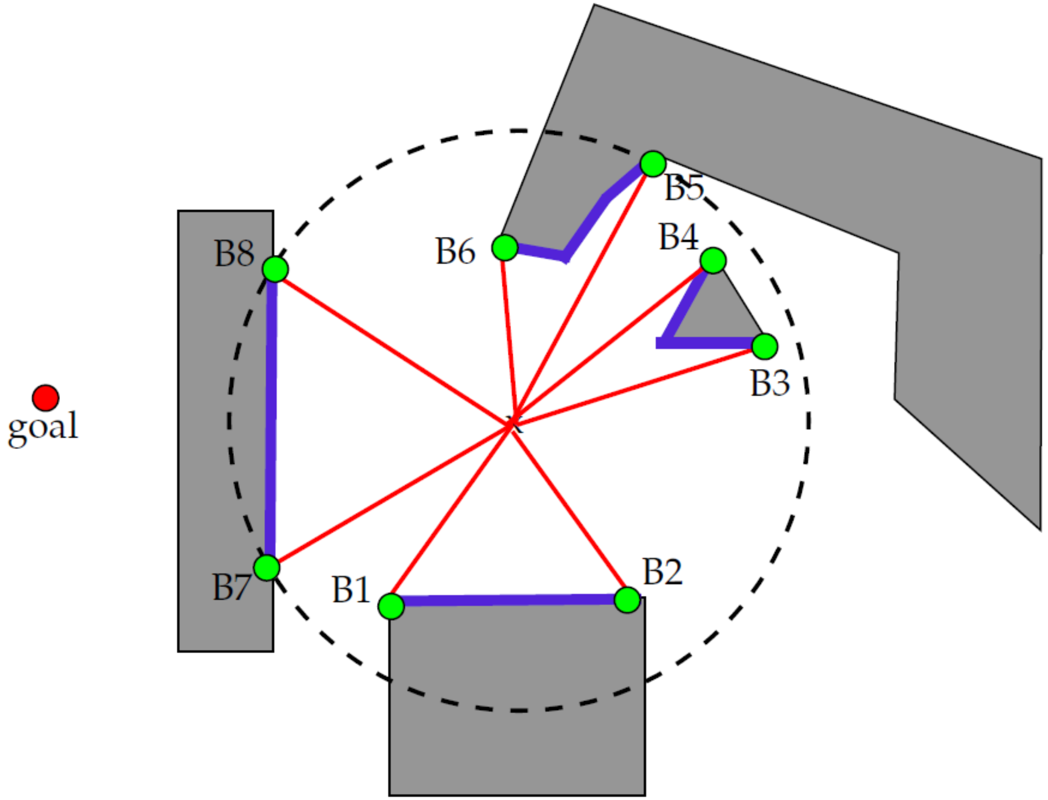
Which algorithm is used here?

https://www.youtube.com/watch?v=GPOe6uDOS_Q

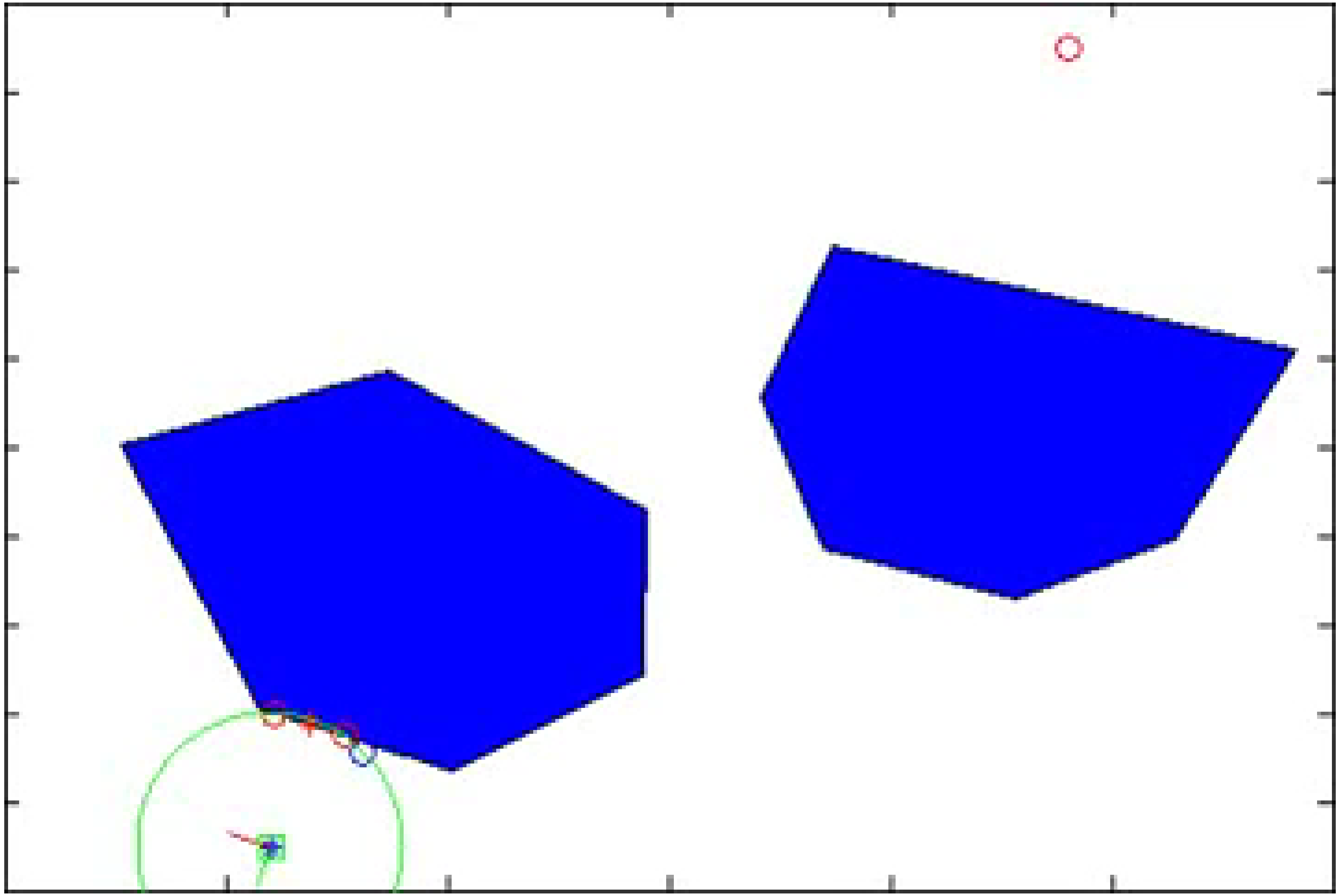
Tangent Bug Algorithm

- Robot equipped with range finding sensors
- Choose direction that minimises the distance to the goal while in motion





Tangent Bug Algorithm - Example



Tangent Bug Algorithm – KUKA YouBot

https://www.youtube.com/watch?v=SV5GNDj4JZA

Many other methods…

method		model fidelity			view	other requisites			sensors	tested robots	performance		remarks
		shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	remarks
Bug	Bug1 [101, 102]	point			local				tactile				very inefficient, robust
	Bug2 [101, 102]	point			local				tactile				inefficient, robust
	Tangent Bug [82]	point			local	local tangent graph			range				efficient in many cases, robust

R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to autonomous mobile robots. The MIT Press. Second edition. 2011.

Many other methods…

method		model fidelity			view	other requisites			sensors	tested robots	performance		remarks
		shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	
Vector Field Histogram (VFH)	VFH [43]	simplistic			local	histogram grid			range	synchro-drive (hexagonal)	27 ms	20 MHz, 386 AT	local minima, oscillating trajectories
	VFH+ [92, 150]	circle	basic	simplistic	local	histogram grid			sonars	nonholonomic (GuideCane)	6 ms	66 MHz, 486 PC	local minima
	VFH* [149]	circle	basic	simplistic	essentially local	histogram grid			sonars	nonholonomic (GuideCane)	6 … 242 ms	66 MHz, 486 PC	fewer local minima
Bubble band	Elastic band [86]	C-space			global		polygonal	required		various			
	Bubble band [85]	C-space	exact		local		polygonal	required		various			

Many other methods…

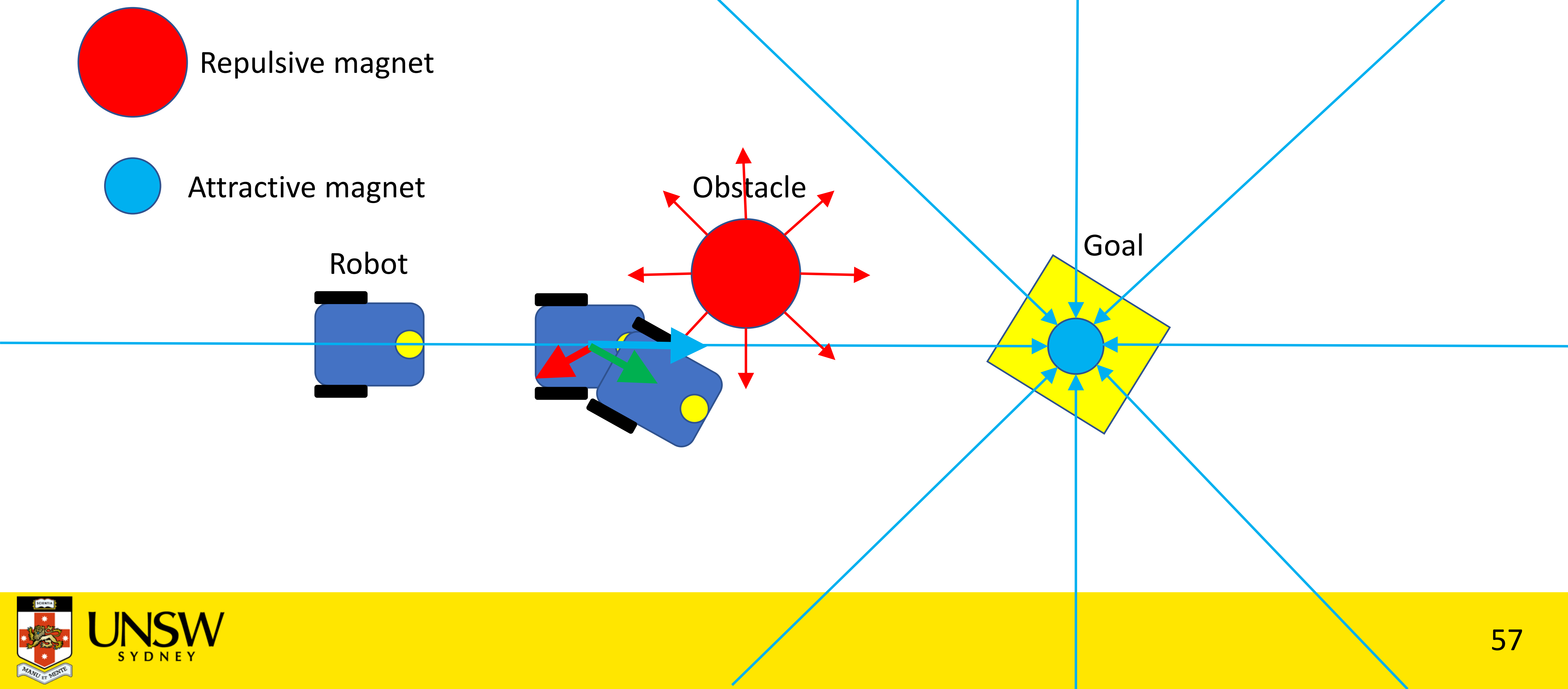
method		model fidelity			view	other requisites			sensors	tested robots	performance		remarks
		shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	
Curvature velocity	Curvature velocity method [135]	circle	exact	basic	local	histogram grid			24 sonars ring, 30° FOV laser	synchro-drive (circular)	125 ms	66 MHz, 486 PC	local minima, turning into corridors
Curvature velocity	Lane curvature method [87]	circle	exact	basic	local	histogram grid			24 sonars ring, 30° FOV laser	synchro-drive (circular)	125 ms	200 MHz, Pentium	local minima
Dynamic window	Dynamic window approach [69]	circle	exact	basic	local	obstacle line field			24 sonars ring, 56 infrared ring, stereo camera	synchro-drive (circular)	250 ms	486 PC	local minima
Dynamic window	Global dynamic window [44]	circle	(holonomic)	basic	global		C-space grid	NF1	180° FOV SCK laser scanner	holonomic (circular)	6.7 ms	450 MHz, PC	turning into corridors

R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to autonomous mobile robots. The MIT Press. Second edition. 2011.

Artificial Potential Field

Artificial Potential Field

- Combines global planning and local planning

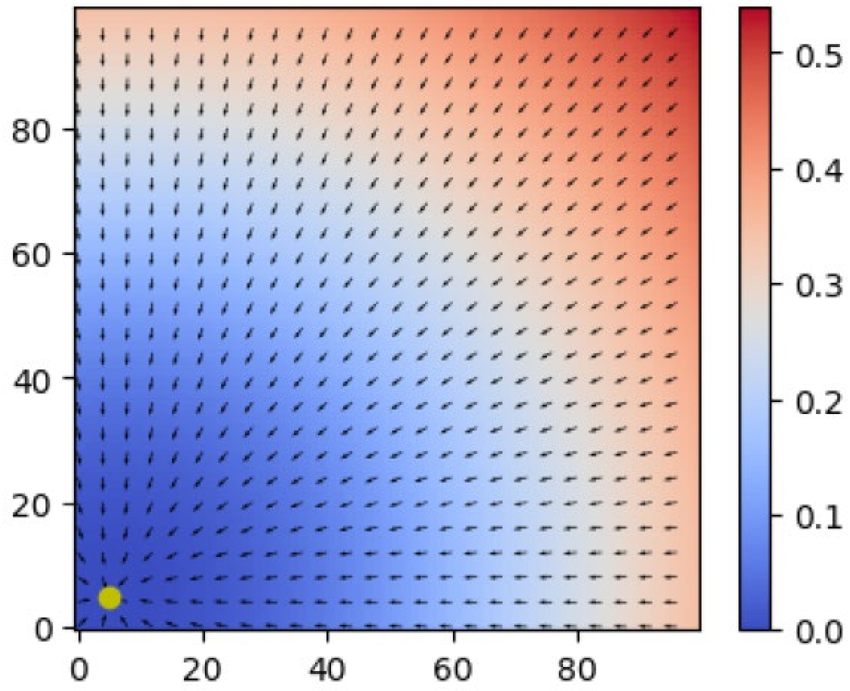


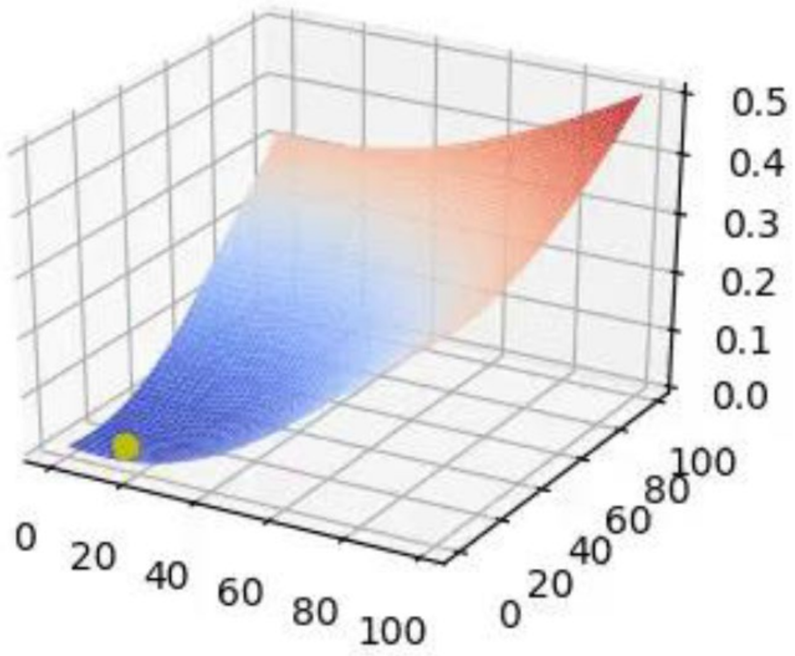
Artificial Potential Field – Attractive potential for goal

- A smooth, differentiable function so that it is easy to calculate the target vector

$$U_{att}(p) = \frac{1}{2}\zeta\left\|p - p_{goal}\right\|^2$$

$$F_{att}(p) = -\nabla U_{att}(p)$$





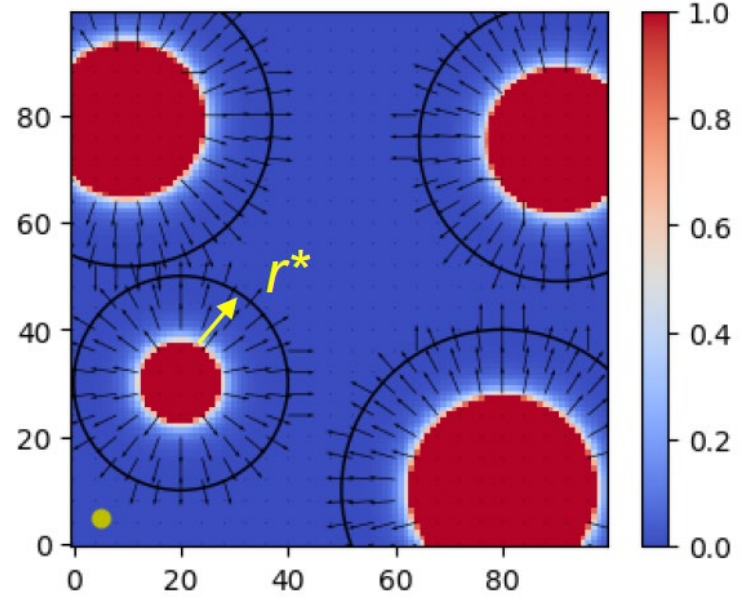
Artificial Potential Field – Repulsive potential for obstacles

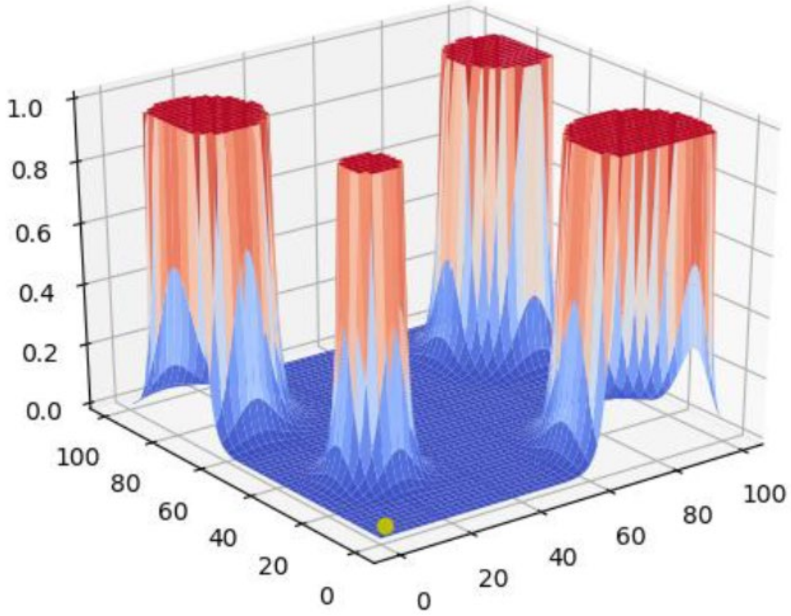
- A smooth, differentiable function to repel the obstacles within an effective range

$D(p)$ is the distance to the nearest obstacle boundary

$$U_{rep}(p) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(p)} - \frac{1}{r^*}\right)^2, & D(p) \leq r^* \\ 0, & otherwise \end{cases}$$

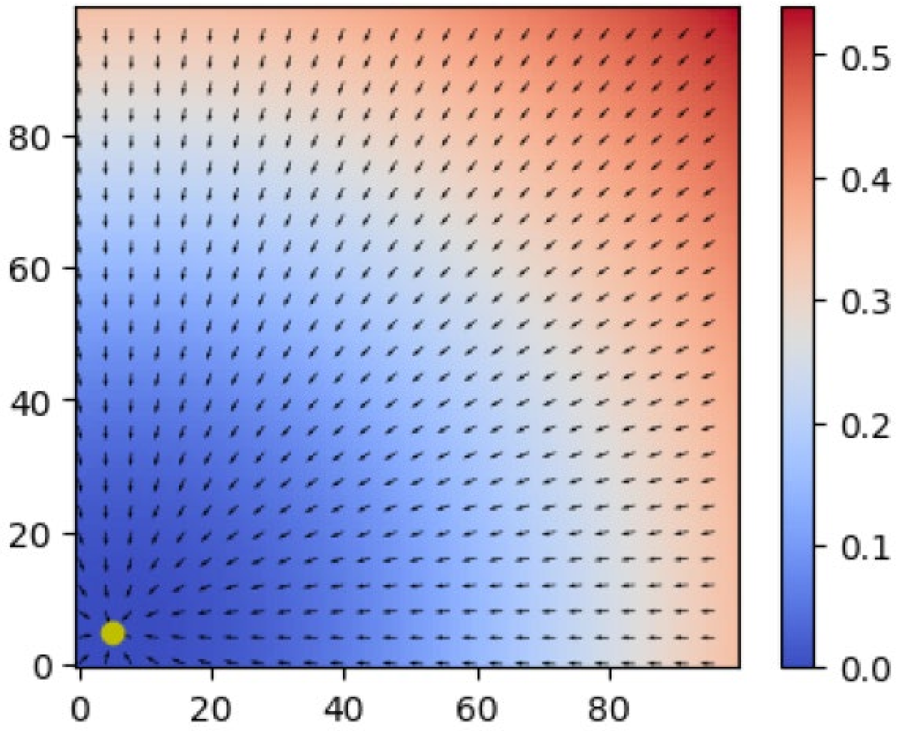
$$F_{rep}(p) = -\nabla U_{rep}(p)$$





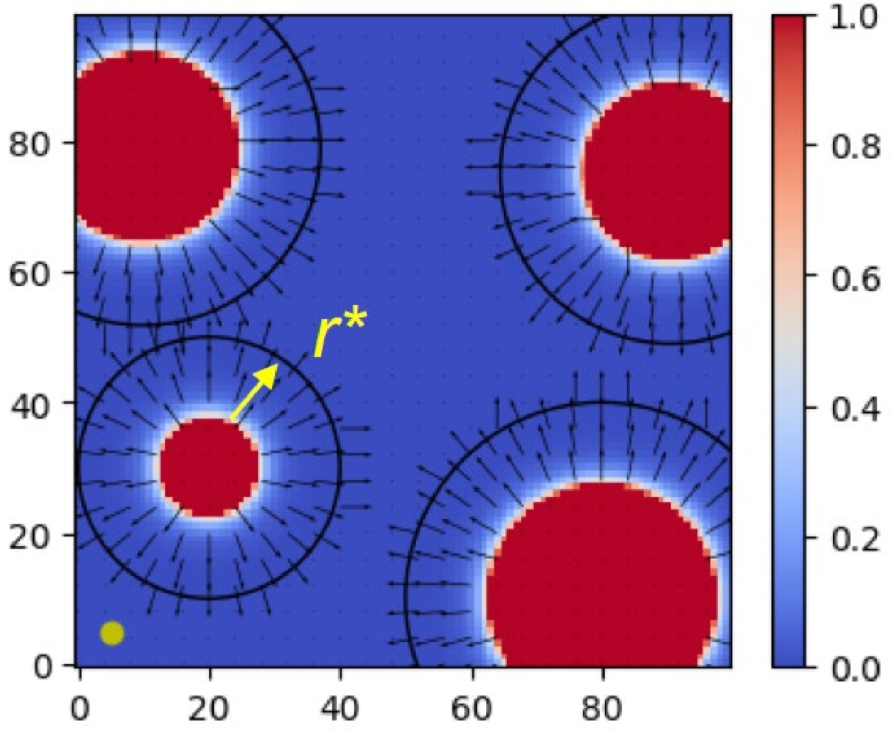
Artificial Potential Field – Overall potential

Attractive potential

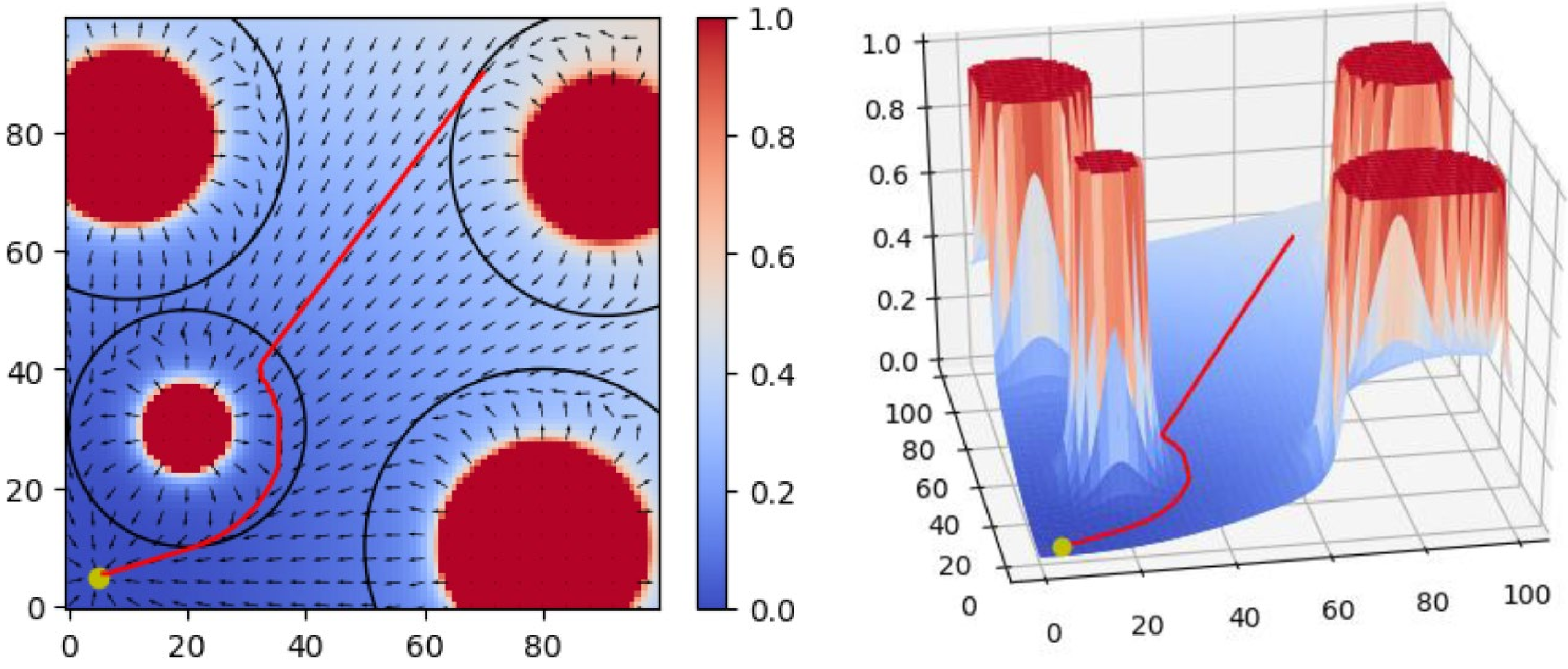


+

Repulsive potential



Artificial Potential Field – Overall potential

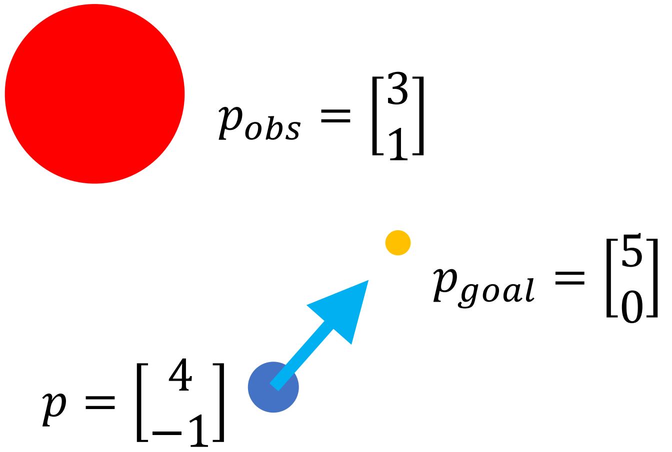


Nick Lawrance, https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/asl-dam/documents/lectures/autonomous_mobile_robots/spring-2019/Planning_I_2019.pdf

UNSW SYDNEY

Artificial Potential Field - Example

- Calculate the net potential field force when the robot is at (4, -1). Use $\zeta = 1$, $\eta = 3$, and $r^* = 5$.



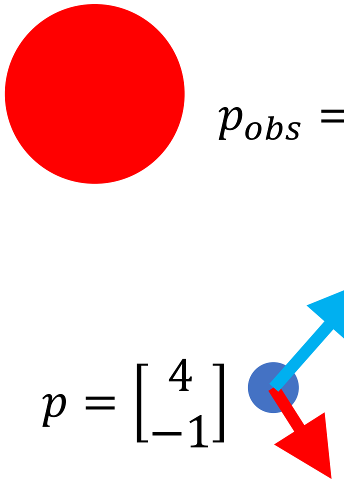
Attractive force:

$$U_{att}(p) = \frac{1}{2}\zeta \left\| p - p_{goal} \right\|^2$$

$$F_{att}(p) = -\nabla U_{att}(p) = -\zeta\left(p - p_{goal}\right)$$

Artificial Potential Field - Example

- Calculate the net potential field force when the robot is at (4, -1). Use ζ = 1, η = 3, and r^*= 5.



$p_{obs} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$

$p_{goal} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$

$p = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$

Repulsive force:

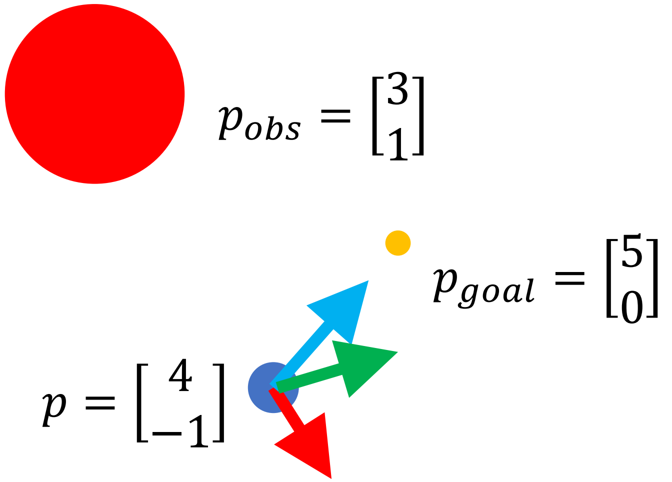
$$D(p) = \sqrt{(p - p_{obs})^T (p - p_{obs})}$$

$$U_{rep}(p) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(p)} - \frac{1}{r^*}\right)^2, & D(p) \leq r^* \\ 0, & otherwise \end{cases}$$

$$F_{rep}(p) = -\nabla U_{rep}(p) = \begin{cases} \eta\left(\frac{1}{D(p)} - \frac{1}{r^*}\right)\frac{1}{(D(p))^3}(p - p_{obs}), & D(p) \leq r^* \\ 0, & otherwise \end{cases}$$

Artificial Potential Field - Example

- Calculate the net potential field force when the robot is at (4, -1). Use ζ = 1, η = 3, and r^*= 5.



Net field force:

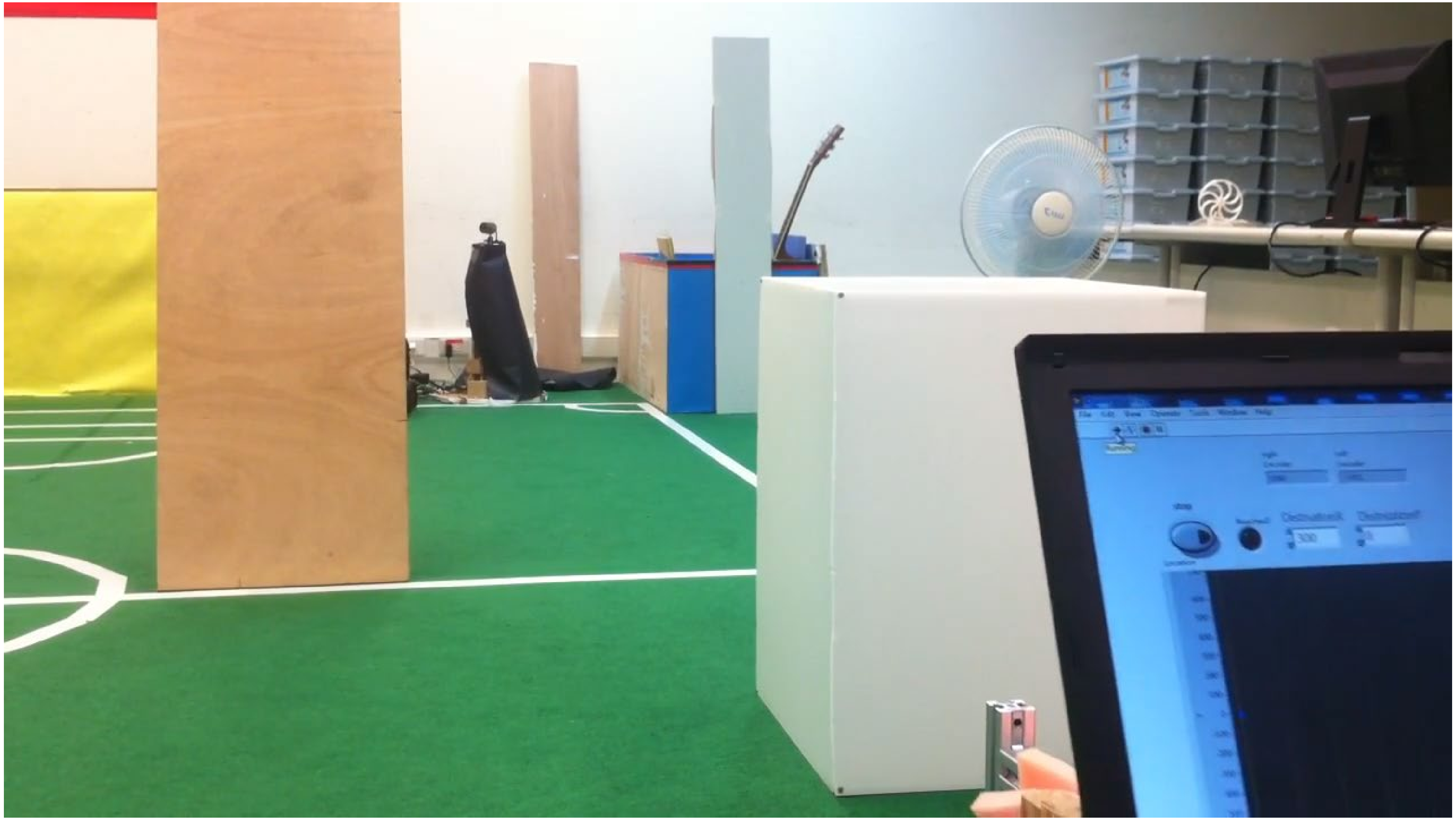
$$F_{net}(p) = F_{att}(p) + F_{rep}(p)$$

<u>Homework:</u>

Solve this problem and write a program in MATLAB (or any other language) for the calculation.

Hints: The solution is: (1.0663, 0.8673).

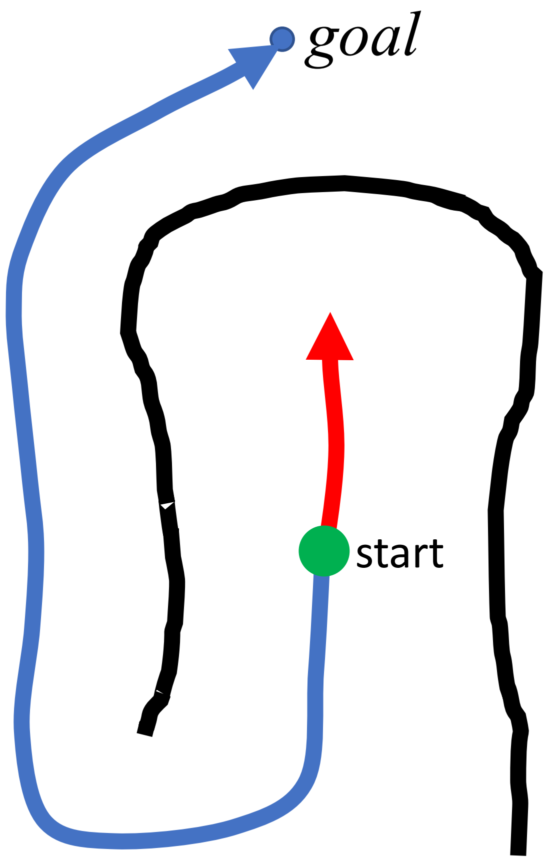
Artificial Potential Field - In Practice



https://www.youtube.com/watch?v=QpuO9OyQ7Nw

Artificial Potential Field - Summary

- Useful for planning paths around obstacles
  - Works with dynamic obstacles too
- Won’t land precisely at goal
  - Use another trajectory planner once close to goal
- Can get stuck in local minima
  - Places where attractive and repulsive forces cancel to zero
  - Ratio between attractive and repulsive forces matters
  - Basins where obstacles “herd” robot to centre
- Modifying the potential functions can avoid local minima (e.g., harmonic potentials)
- Applicable to nonholonomic planning
- Applicable to higher-order configuration spaces



Planning In Practice

Planning in practice

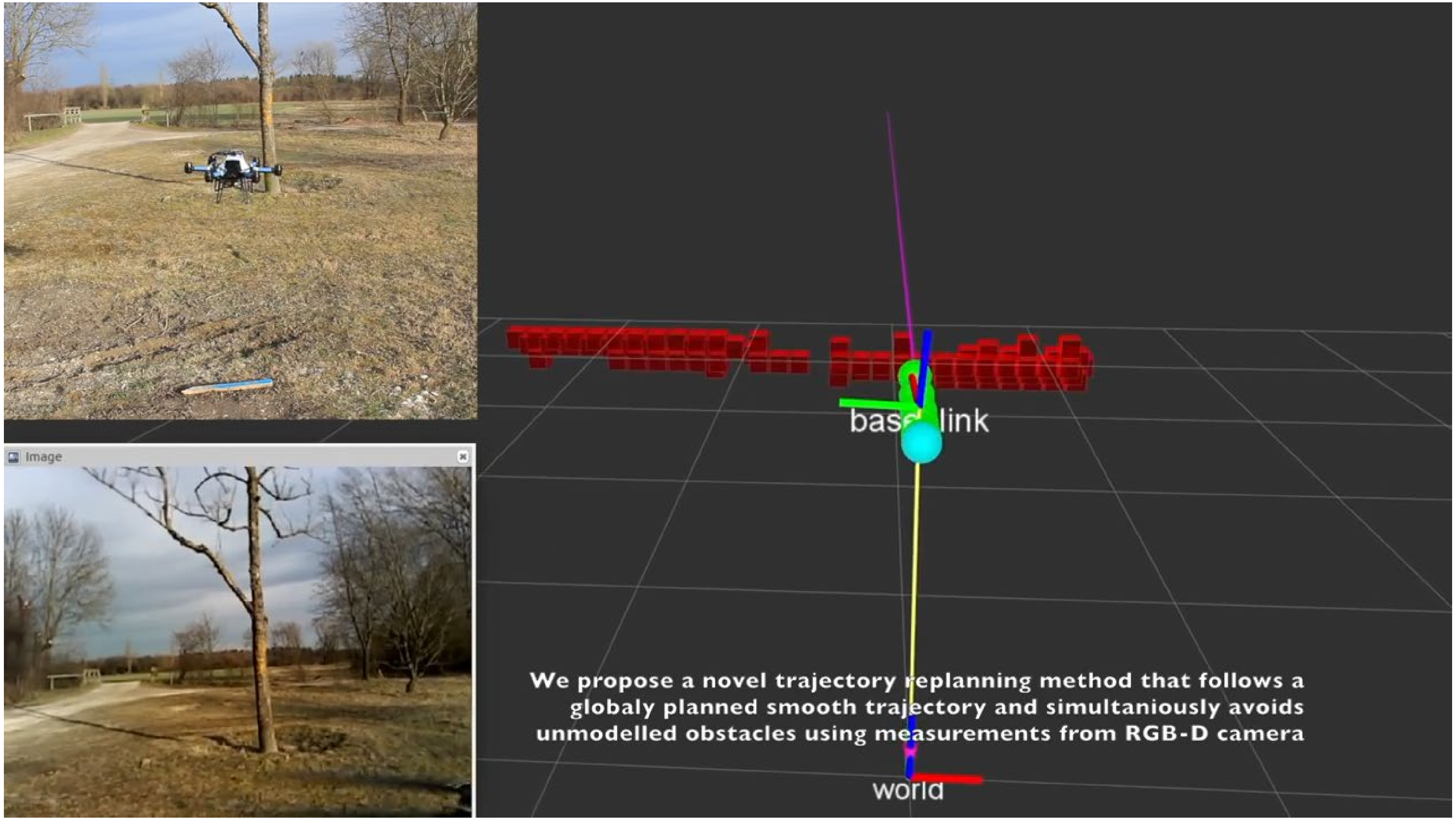
- In general planning is done in a hierarchical manner

- Global planner
  - Construct a path from initial position to the goal
  - A*, RRT, etc.

- Local planning
  - Continuously run to adapt the planned global path to changes
  - Avoids the need to compute the entire global path repeatedly

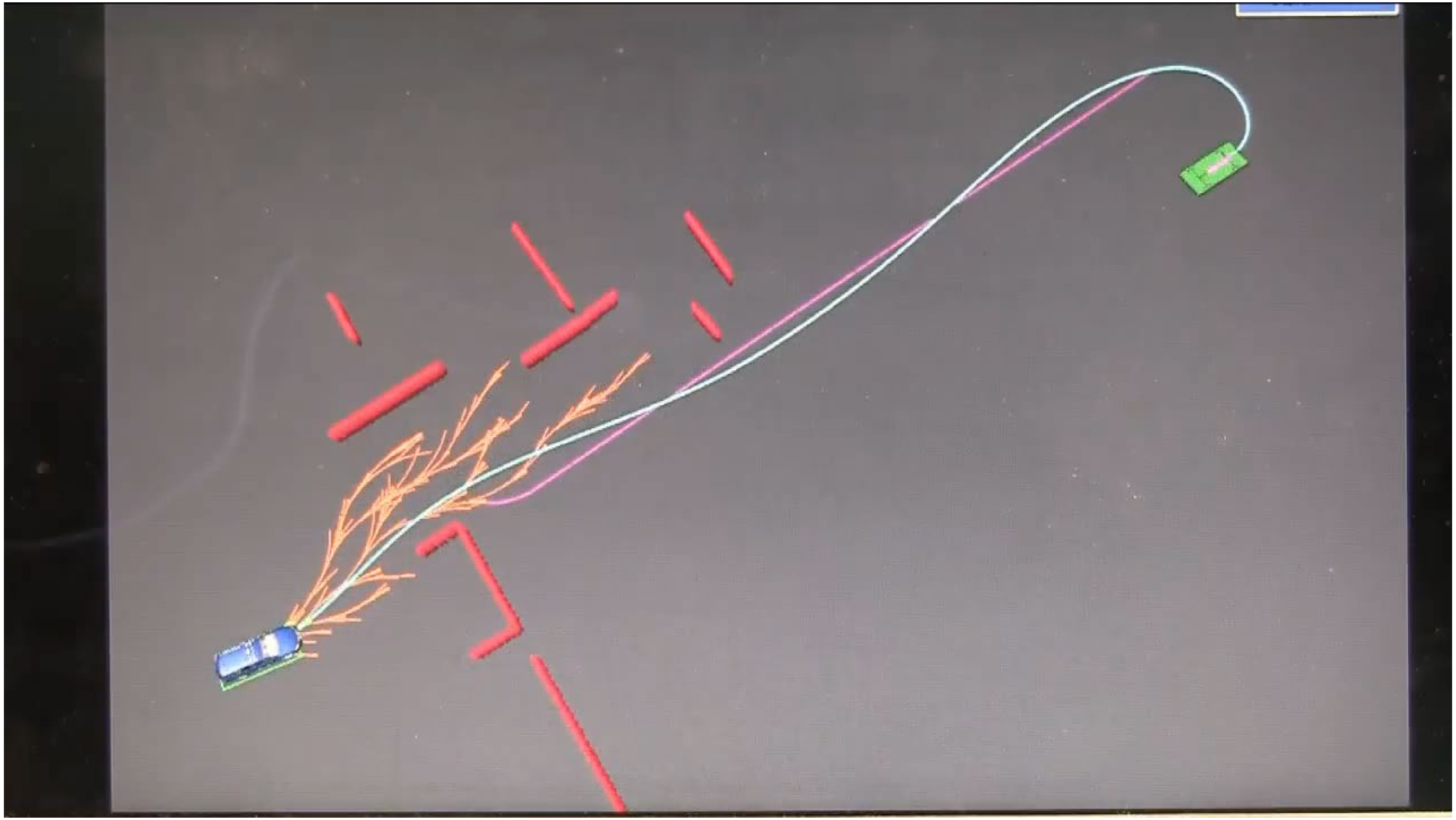
- Reactive
  - For collision avoidance in case of fast dynamic objects

Real-Time Trajectory Replanning using B-Splines



https://www.youtube.com/watch?v=jh6tMHjxHSY&feature=youtu.be

Real-Time Trajectory Replanning using Hybrid-state A*



https://www.youtube.com/watch?v=zS3st_7og3A&feature=youtu.be

Dolgovet al, Practical Search Techniques in Path Planning for Autonomous Driving, 2008 slido

Which algorithm are you using/planning to use for Assignment Phase B?

ⓘ Start presenting to display the poll results on this slide.

Modified Flood Fill Algorithm

Planning in Micromouse

Search Run
- Exploration
- Mapping

Speed Run
- Planning
- Execution

Modified Flood Fill – For exploration

- Once a new wall is found, instead of flood-filling the whole maze again, just update the cells that could be potentially affected

- Based on this observation:
  - After updating, every cell should have a value equal to the lowest value of the neighbouring open cells plus 1, except the central cell

4	3	2	3	4
3	2	1	2	3
4	5	0	1	2
3	2	1	2	3
4	3	2	3	4

UNSW SYDNEY

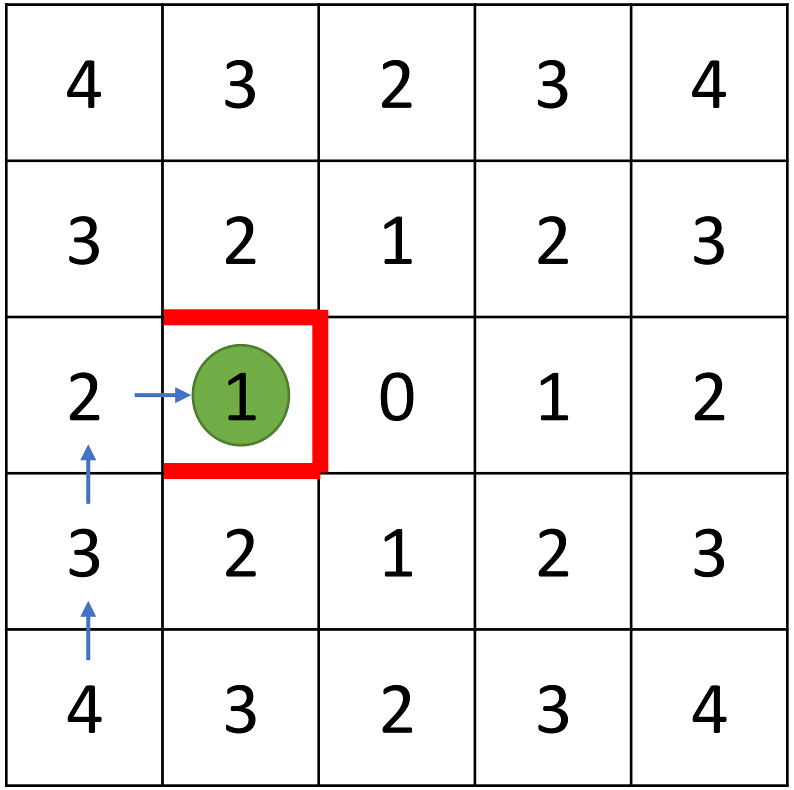
Modified Flood Fill – For exploration

- At the start, initialise the maze using the standard flood fill algorithm (or any equivalent initialisation method)

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

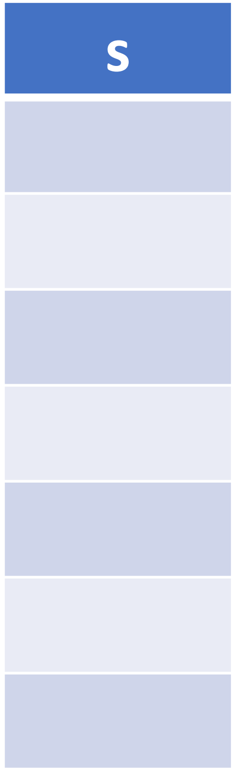
Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell



Modified Flood Fill – For exploration

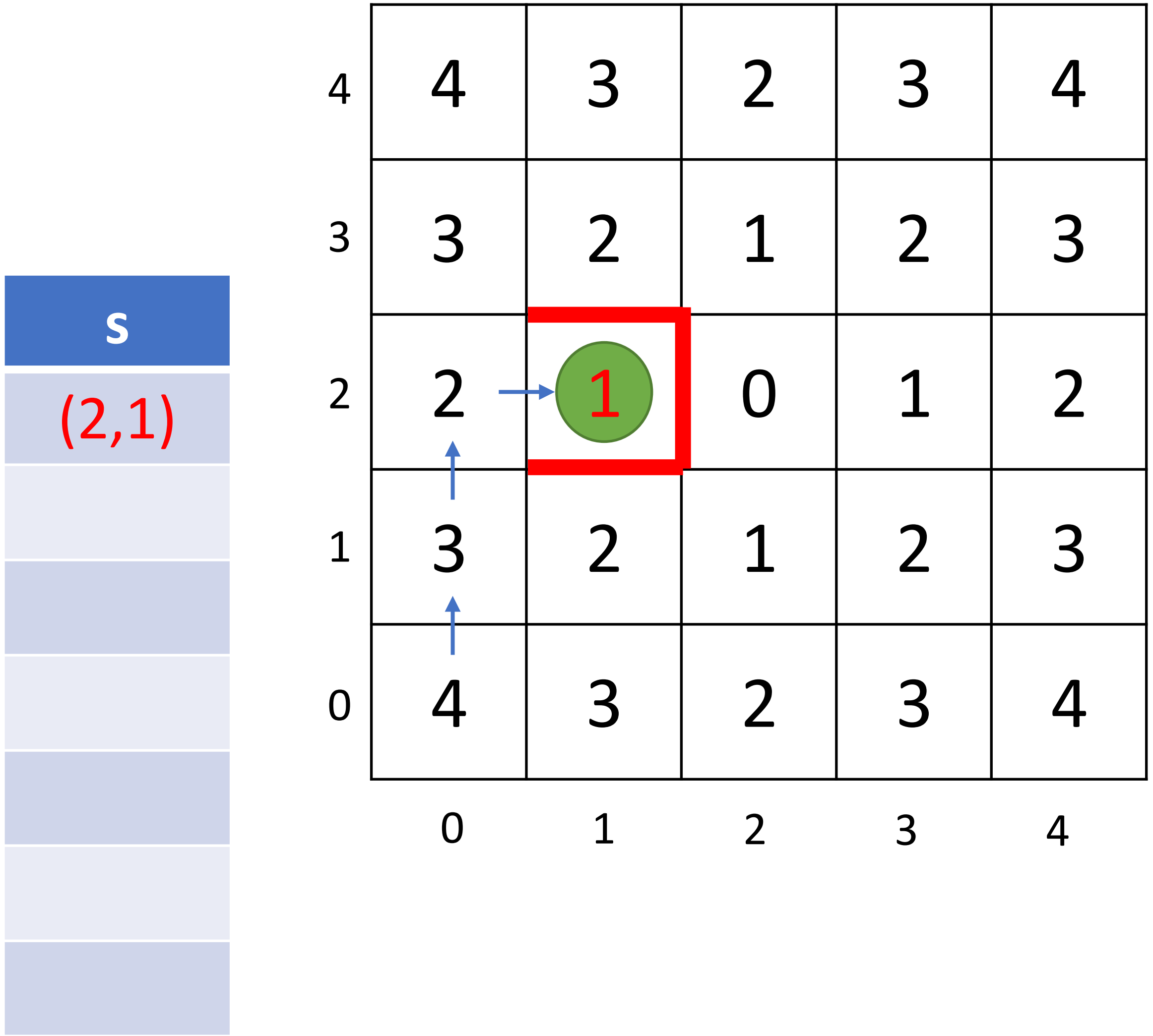
- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell



4	4	3	2	3	4
3	3	2	1	2	3
2	2	1	0	1	2
1	3	2	1	2	3
0	4	3	2	3	4
	0	1	2	3	4

Modified Flood Fill – For exploration

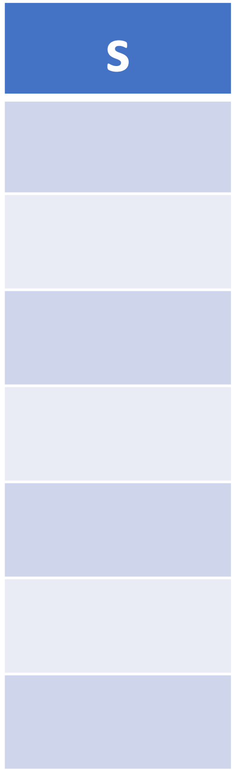
- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

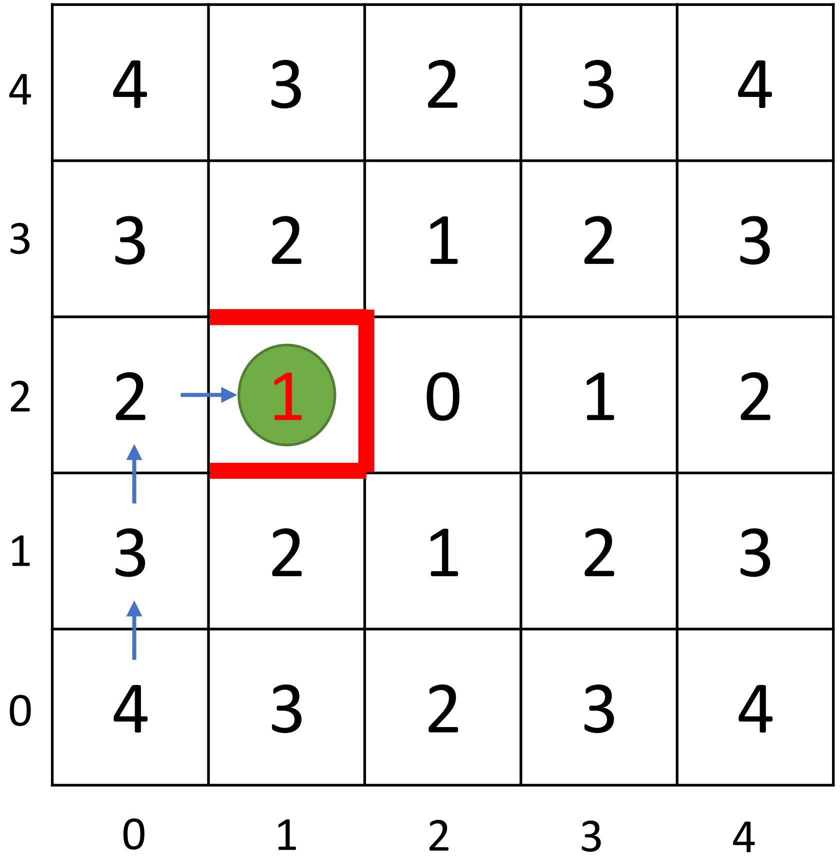


Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (2,1)
(present cell distance) pcd = 1
(minimum distance) md = 2
md != pcd - 1





Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (2,1)
(present cell distance) pcd = 1
(minimum distance) md = 2
md != pcd - 1

s
(3,1)
(2,0)
(1,1)

4	4	3	2	3	4
3	3	2	1	2	3
2	2	3	0	1	2
1	3	2	1	2	3
0	4	3	2	3	4
	0	1	2	3	4

Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (3,1)
(present cell distance) pcd = 2
(minimum distance) md = 1
md == pcd - 1

s
(2,0)
(1,1)

4	4	3	2	3	4
3	3	2	1	2	3
2	2	3	0	1	2
1	3	2	1	2	3
0	4	3	2	3	4
	0	1	2	3	4

Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (2,0)
(present cell distance) pcd = 2
(minimum distance) md = 3
md != pcd - 1

S
(1,1)

4	4	3	2	3	4
3	3	2	1	2	3
2	2	3	0	1	2
1	3	2	1	2	3
0	4	3	2	3	4
	0	1	2	3	4

Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (2,0)

(present cell distance) pcd = 2

(minimum distance) md = 3

md != pcd - 1

S
(3,0)
(1,0)
(2,1)
(1,1)

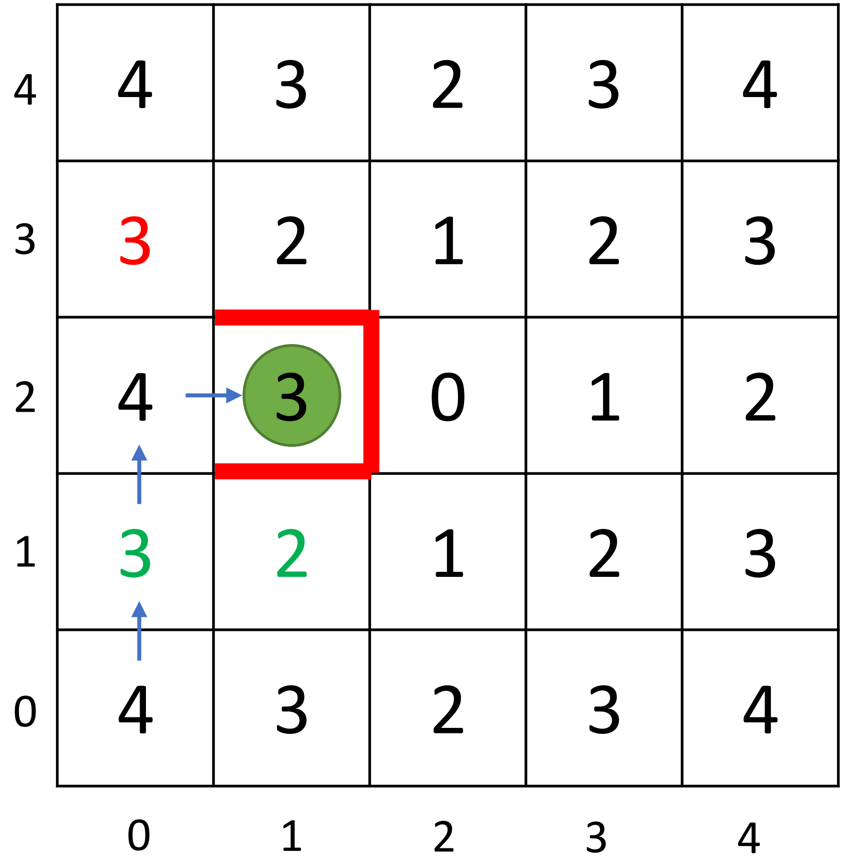
4	4	3	2	3	4
3	3	2	1	2	3
2	4	3	0	1	2
1	3	2	1	2	3
0	4	3	2	3	4
	0	1	2	3	4

Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (3,0)
(present cell distance) pcd = 3
(minimum distance) md = 2
md == pcd - 1

S
(1,0)
(2,1)
(1,1)



Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

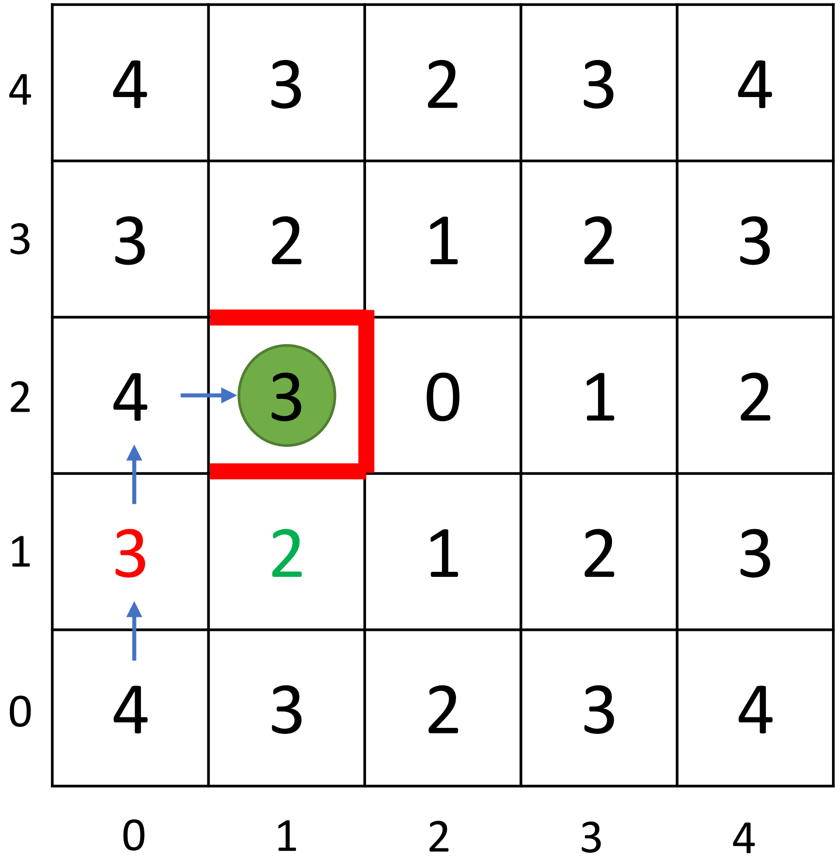
present cell = (1,0)

(present cell distance) pcd = 3

(minimum distance) md = 2

md == pcd - 1

s
(2,1)
(1,1)

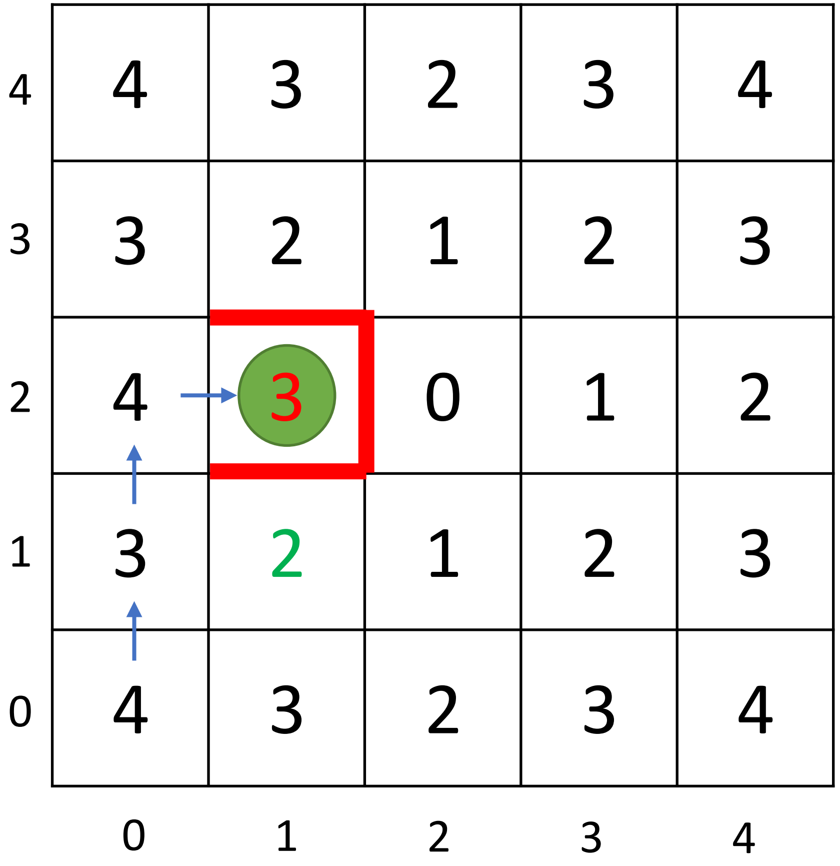


Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (2,1)
(present cell distance) pcd = 3
(minimum distance) md = 4
md != pcd - 1

S
(1,1)



Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (2,1)
(present cell distance) pcd = 3
(minimum distance) md = 4
md != pcd - 1

s
(3,1)
(2,0)
(1,1)
(1,1)

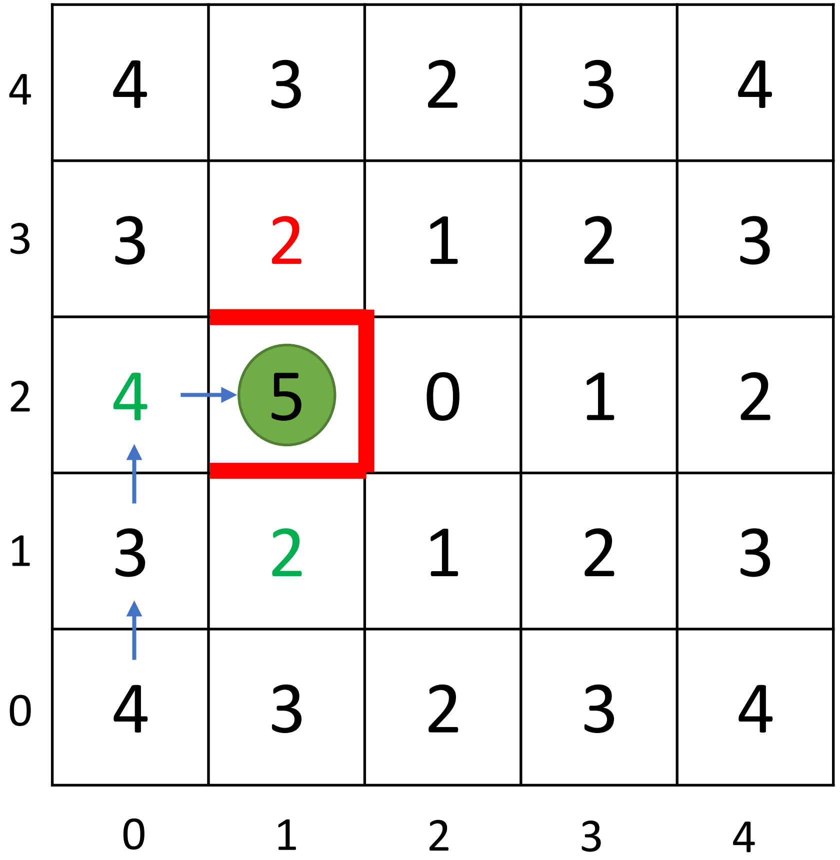
4	4	3	2	3	4
3	3	2	1	2	3
2	4	5	0	1	2
1	3	2	1	2	3
0	4	3	2	3	4
	0	1	2	3	4

Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (3,1)
(present cell distance) pcd = 2
(minimum distance) md = 1
md == pcd - 1

S
(2,0)
(1,1)
(1,1)

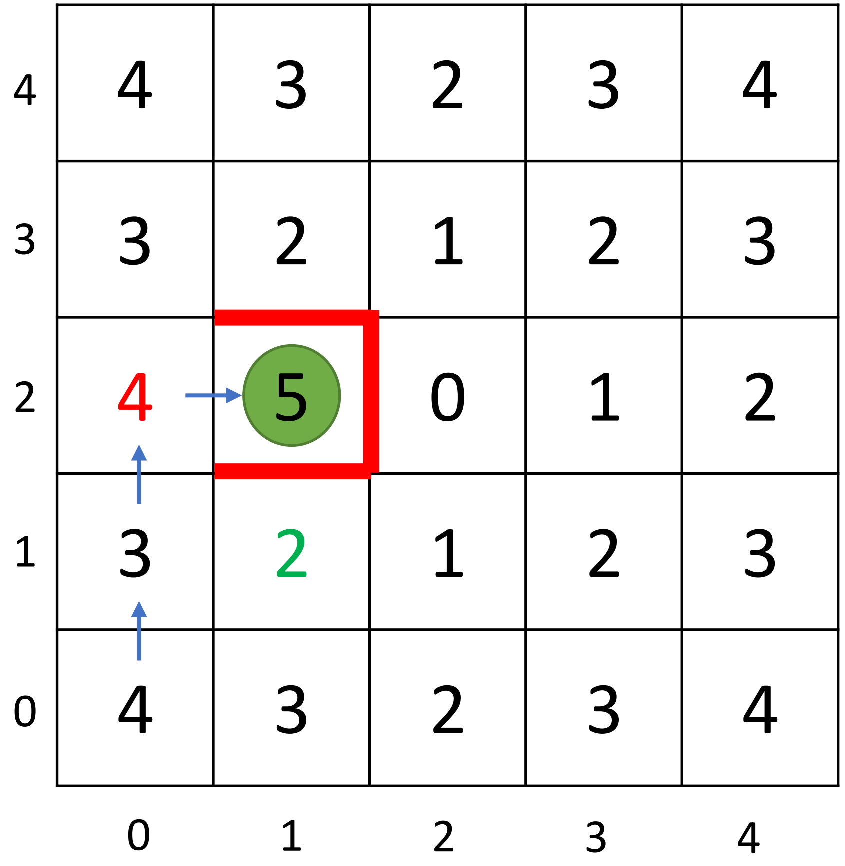


Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (2,0)
(present cell distance) pcd = 4
(minimum distance) md = 3
md == pcd - 1

S
(1,1)
(1,1)



Modified Flood Fill – For exploration

- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (1,1)
(present cell distance) pcd = 2
(minimum distance) md = 1
md == pcd - 1

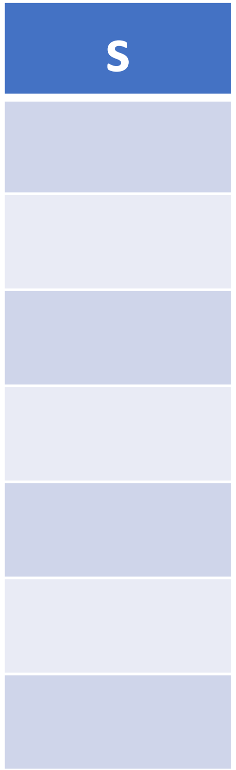
s
(1,1)

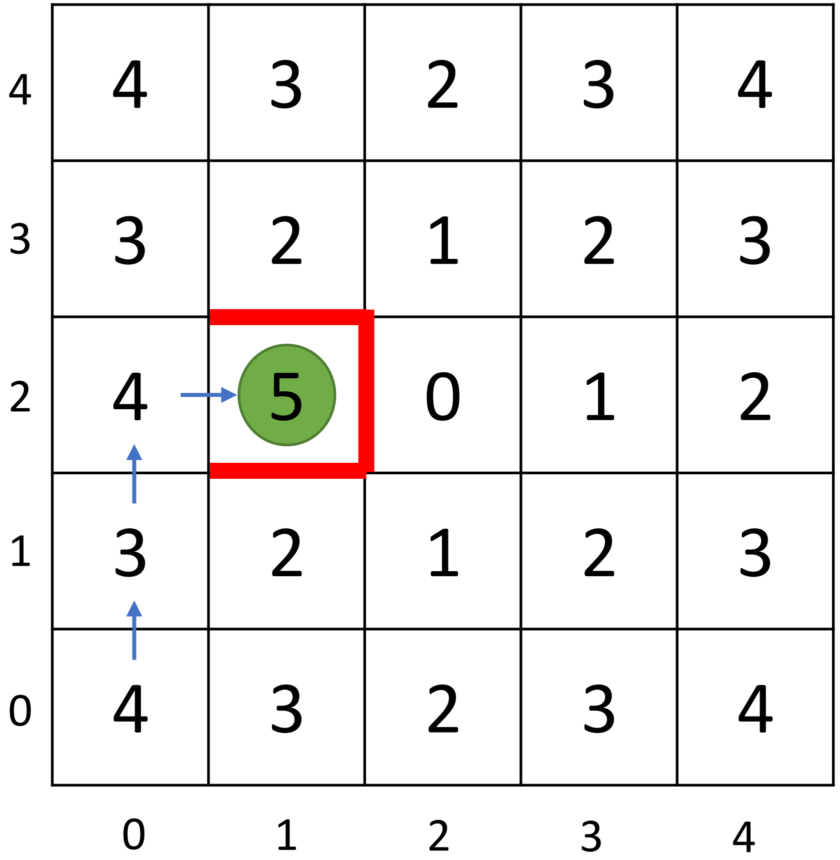
4	4	3	2	3	4
3	3	2	1	2	3
2	4	5	0	1	2
1	3	2	1	2	3
0	4	3	2	3	4
	0	1	2	3	4

Modified Flood Fill – For exploration

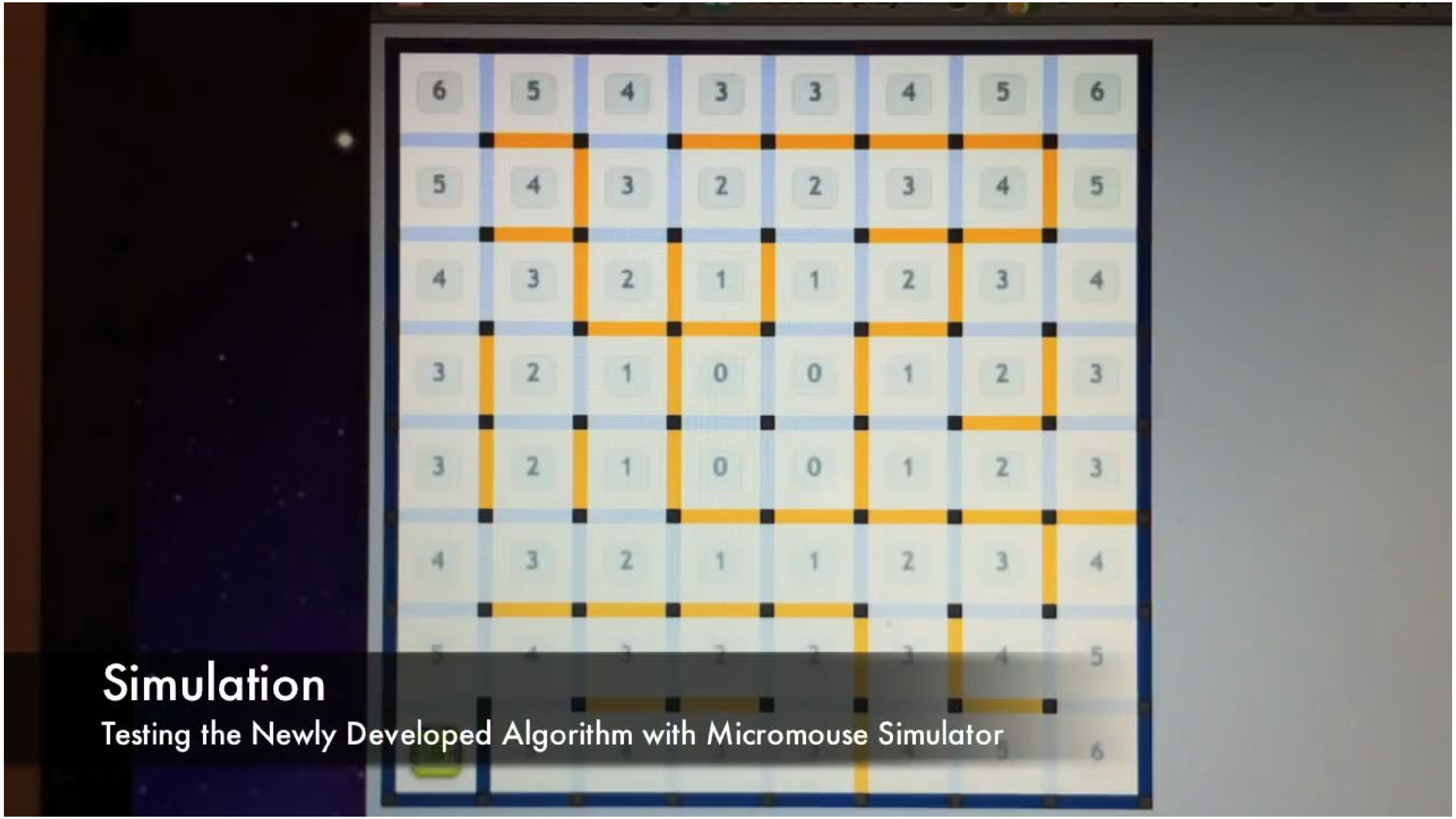
- When new walls are found
  - Run modified flood fill
    - 1. Create an empty stack
    - 2. Push the current cell location (x,y) onto the stack
    - 3. Repeat the following steps while the stack is not empty
      - 3.1 Pull the cell location (x,y) from the stack
      - 3.2 If the minimum distance of the neighbouring open cells, *md*, is not equal to the present cell's distance (*pcd*) – 1, replace the present cell's distance with *md* + 1, and push all neighbour locations onto the stack except the central cell

present cell = (?,?)
(present cell distance) pcd = ?
(minimum distance) md = ?
md == pcd – 1?





Modified Flood Fill – For exploration



Expectations for Learning of Planning I & II

- Understand how the BFS, DFS, Dijkstra's Algorithm, A*, and Bellman-Ford Algorithm work for graph search and the pros and cons of each graph search algorithm
- Understand the concepts of "complete" and "optimal" and be able to apply the two concepts to analyse the graph search algorithms
- Understand how visibility graph and Voronoi graph are constructed
- Understand the difference between workspace and configuration space
- Understand the principles, pros, and cons of PRM, RRT, RRT*
- Understand how the Bug Algorithms work for obstacle avoidance
- Understand how the Artificial Potential Field works for planning

Week 7: Vision I