Two wheel differential drive

Sunday, 18 June 2023 2:17 PM

Velocity:

$$\xi = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(\omega_L + \omega_R) \\ 0 \\ \frac{r}{2l}(-\omega_L + \omega_R) \end{bmatrix}$$

Position (dead reckoning / odometry):

- Requires constant velocity (or small enough Δt for approximate constant velocity)

Increment

Current pose

Next pose

$$p(t + \Delta t) \approx p(t) + \begin{bmatrix} \Delta s \cdot \cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta s \cdot \sin(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{bmatrix}$$

$$\Delta s \equiv \frac{r \cdot \Delta\theta_L}{2} + \frac{r \cdot \Delta\theta_R}{2}$$ — Incremental linear motion

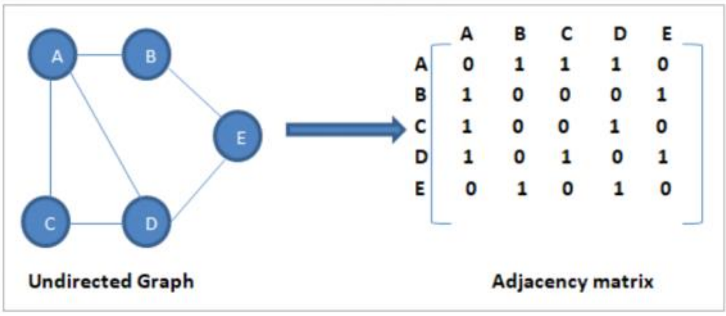
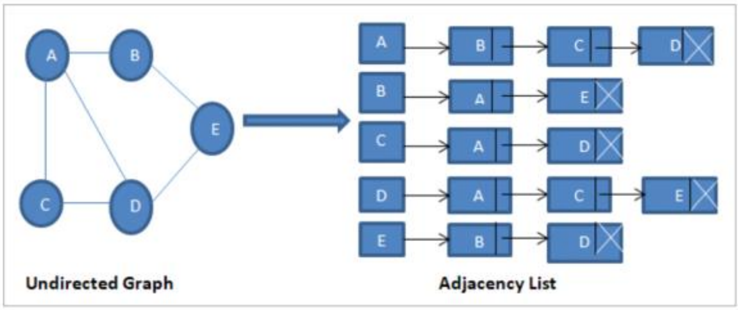
$$\Delta\theta \equiv -\frac{r \cdot \Delta\theta_L}{2l} + \frac{r \cdot \Delta\theta_R}{2l}$$ — Incremental rotation

where $p(t) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$

- note: $\Delta\theta_{L/R}$ is calculated by integrating velocity
- note: the theta used in the pose difference matrix is the old theta

4-Graph Representation

Thursday, 13 July 2023 8:27 AM

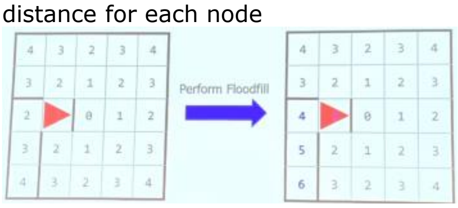
		Desrcription	Pros/cons
Adjacency matrix		- 2D boolean matrix - if matrix[i][j] is true, there is a connection from the i to j	-
Adjacency list		array or linked list of linked lists	

4-Graph search algorithms

Tuesday, 11 July 2023 5:40 PM

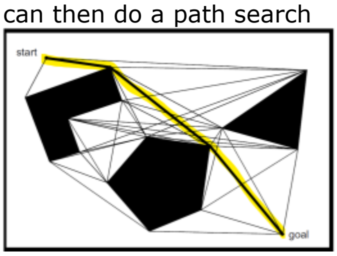
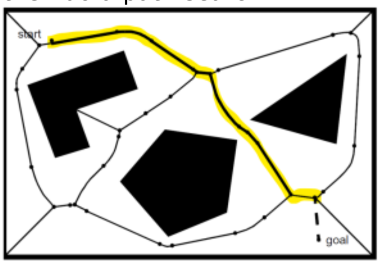
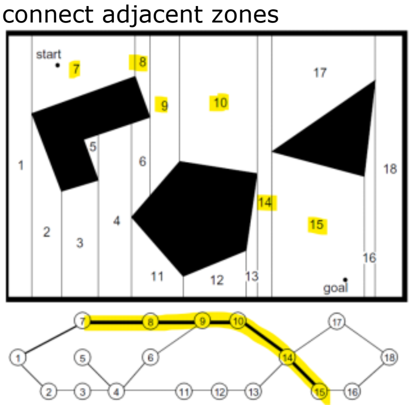
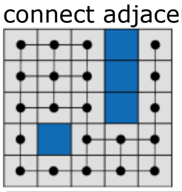
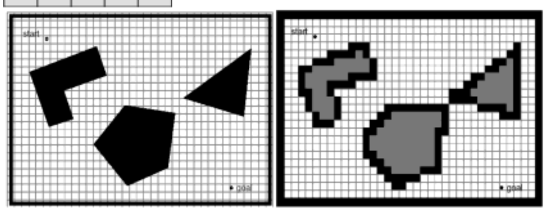
	Explanation	Pros/cons	Pseudocode
BFS	Use queue	- Complete - Optimal if all branch weights are equal - Simple	
DFS	use stack	- Not complete - Not optimal - Simple	
Dijkstra's	BFS, except use a priority queue and keep track of the shortest distance to the current node. To find a path back to the start, keep track of the previous node for each node.	- Complete - Optimal - works with weighted edges	
A*	Same as dijkstra, except use distance cost + heuristic cost	- Complete - Optimal - works with weighted edges - can add a "heuristic" (increase certain edge weights based on some pre-defined rule) to help guide the search towards the goal	

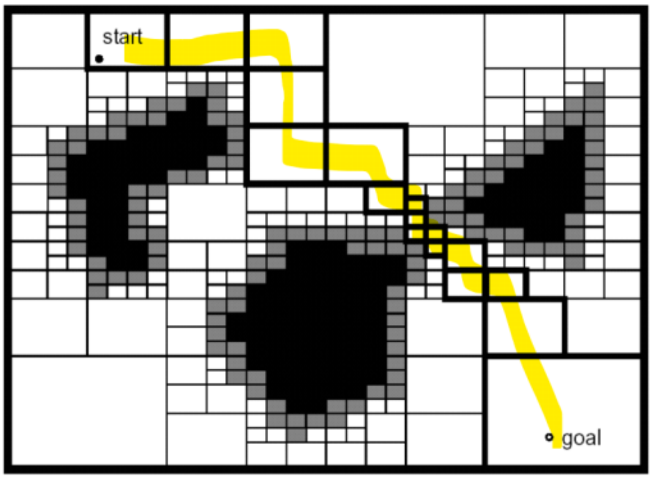
Bellman ford		- Not complete (negative edge cycles)
		-
		- works with negative edge weights
		- Doesn't scale well - slower than dijkstra
		- need to check for negative edge cycles
			set all distances to infinity, except starting node
			while(distance graph has changes)
			    for every edge:
			        update distance to connecting nodes ("relaxing")
			
			check for negative edge cycles
			    return error

flood fill	- useful approach for solving a maze
	- assume maze has no walls
	    • since we know the size, we can construct a graph and calculate a distance for each node
	        ○ go from the goal until every square's distance is updated
	        ○ (on the very first flood fill, set distances to inf before updating distances)
	- as you detect walls, recalculate the distance for each node
	
	"follow the path of numerical least resistance"		

5-Graph construction

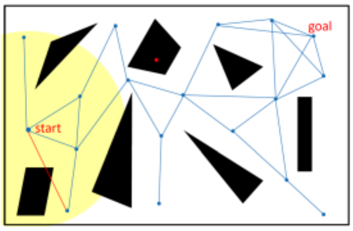
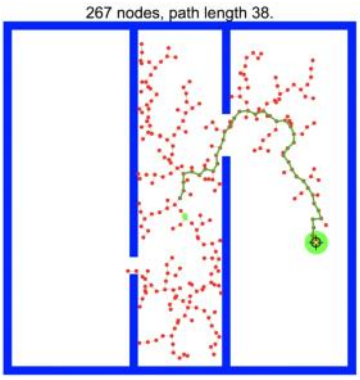
Thursday, 13 July 2023 8:23 AM

	Explanation	Pros	Cons
Visibility Graph	- Connect vertices which are visible to each other - can then do a path search 	- gives the shortest path	- not safe (can hit edges)
Voroni Diagram	- construct vertices at points with equal distance to the nearest two edges - connect vertices - then do a path search 	- safe (doesn't hit edges)	- not always the shortest path
Exact cell decomposition	- split map into "zones" based on polygon vertices of obstacles - connect adjacent zones 	- Efficient for large, sparse environments	- complex implementation
Fixed cell decomposition (occupancy grid	- split map into cells of a grid - connect adjacent cells which don't contain obstacles  	- Easy to implement	- High memory requirements - may lose narrow passages if resolution isn't high enough
Adaptive cell decomposition	- similar to fixed cell decomposition (occupancy grid) - fewer cells/nodes for large areas - more cells/nodes close to obstacles	- solves memory issue	- complex implementation

	start
goal		

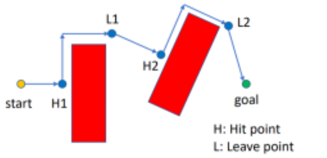
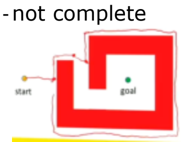
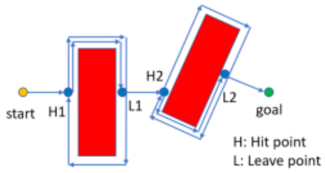
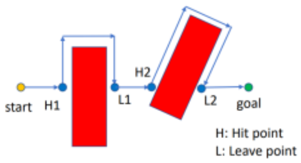
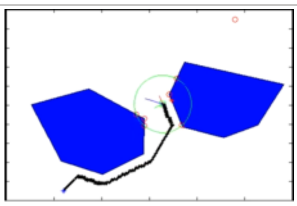
5-Sample based planning

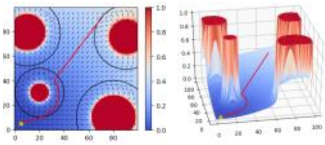
Thursday, 13 July 2023 8:24 AM

	Image	Pros	Cons
PRM		- simple conceptually - can solve high-dimension planning - more points at the start will result in closer to optimal path (i think)	- can lose narrow passages - not for dynamic environments - assumes holonomic motion
RRT		- can apply nonholonomic constraints when adding nodes to the graph • e.g. so that a nonholonomic robot can follow the final path	- doesn't give shortest path (gives a jagged path)

5-Obstacle avoidance

Thursday, 13 July 2023 7:55 AM

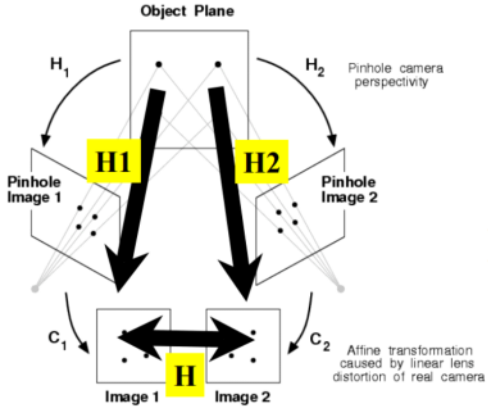
	image	Description	Requirements	Pros/cons
Bug 0		- Moves towards the goal - if it hits an obstacle, follow the boundary until it is no longer in the way	- known direction to goal - tactile sensors	- not complete  - minimal memory and computational power required
Bug 1		Bug 0, except **circumnavigate** the obstacle and **remember how close you get to the goal**, then return and leave obstacle at that point	- known direction to goal - tactile sensors - encoders	
Bug 2		Bug 1, except uses an **m-line** to find the closest point to the goal (don't need to circumnavigate) only leave the obstacle if the m-line encounter is closer to the goal than the first encounter	- known direction to goal - tactile sensors - encoders - extra memory + computing power	Typically more efficient than bug 1, but not in all cases. E.g. 
Tangent Bug		- move towards the goal - if an obstacle is detected which obstructs the path to the goal, move **towards the** obstacle's **corner point closest to the goal**	- known direction to goal - range finding sensors	- don't have to hit the obstacle

Artificial Potential Field (APF)	
	Can get stuck in local minima:
	
		- Attractive force towards the goal, repulsive force away from the goal.
		$\boldsymbol{F}_{att} = -\zeta\left(\boldsymbol{p} - \boldsymbol{p}_{goal}\right)$
		$\boldsymbol{F}_{rep}$
		$= \eta\left(\frac{1}{D(p)} - \frac{1}{r^*}\right) * \frac{1}{\left(D(p)\right)^3}$
		$* \left(\boldsymbol{p} - \boldsymbol{p}_{goal}\right)$
		for $D(p) \leq r^*$, 0 otherwise.
		
		- D(p) = distance to obstacle boundary
		- r* = radius within which repulsion has effect
		- ζ, η, r^* are tuneable parameters
		
		- choose how the bot responds to the force
		  • e.g. more in the direction of the force
			- know position of the goal
			- know position of or can detect obstacles
				Pros:
				- works with dynamic obstacles
				- applicable to non-holonomic planning
				  • (don't need linear motion)
				- applicable to higher order configuration spaces
				
				Cons:
				- can get stuck in local minima (forces = 0, basins "herd" robot)
				  • can modify potential functions to avoid this
				- need to tune parameters

7-Homography Matrix

Tuesday, 1 August 2023 4:22 PM

Used in perspective transforms



$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \sigma \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

- Homography matrices are invertible.
- Product of Homography matrices is also a Homography matrix.

$$H = H_1^{-1} H_2$$

To calculate:

1. set h_33 = 1
2. A*h = b
    a.

$$\begin{array}{l} \textbf{Point 1} \\ \\ \textbf{Point 2} \\ \\ \textbf{Point 3} \\ \\ \textbf{Point 4} \\ \\ \end{array} \overset{\textbf{2N x 8}}{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix}} \overset{\textbf{8 x 1}}{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}} = \overset{\textbf{2N x 1}}{\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}}$$

additional points $\vdots$ $\vdots$

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$