PATH PLANNING AND OBSTACLE AVOIDANCE

Dr. Teena Hassan
Robotics Innovation Center
DFKI Bremen

Prof. Dr. Dr. h.c. Frank Kirchner
Arbeitsgruppe Robotik, Universität Bremen
https://robotik.dfki-bremen.de/
robotik@dfki.de

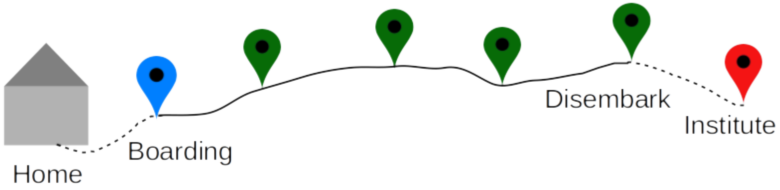
28th November, 2021 – Bremen, Deutschland

Contents

Path Planning and Obstacle Avoidance

Introduction to Path Planning

- **Path**: A sequence of connected locations in an environment.
- **Path planning**: The process of finding a path from an initial location to a goal location in the given environment.
- **Waypoints**: The intermediate locations that lie on the path in between the initial location and the goal location.
- Very often, path planning deals with finding an **optimal path**, e.g. shortest path, least-cost path, etc.



We use path planning in our everyday life. For example,

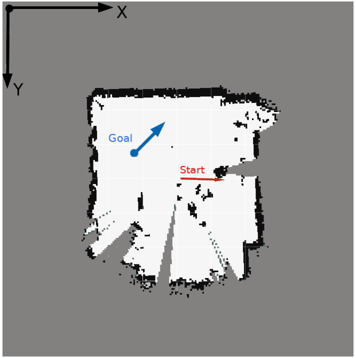
- We ask the navigation system of our cars to recommend routes with the **least traffic congestion** to reach our destination.
- To get from our homes to the university, we search for the **fastest** public transport connections.
- When traveling to a far-away holiday destination, we look for the **cheapest** flight connections.
- On the university campus, we look at floor maps to find our way from one lecture hall to another **via an intermediate location**, such as the cafeteria or the library.

Path Planning in Mobile Robots

German Research Center for Artificial Intelligence

Pre-requisites for path planning:

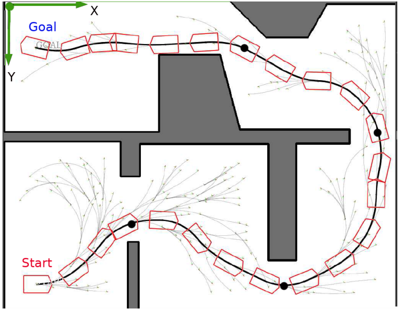
- A representation of the environment (**Map**).
- The definition of the **world coordinate system** in the map.
- Knowledge about the current position and orientation of the robot in the map (i.e. the current robot pose) (**Localization**).
- Description of the goal position and orientation (i.e. **goal pose**).



Universität Bremen

German Research Center for Artificial Intelligence

Path planning in mobile robots refers to the process of **autonomously** finding a sequence of robot poses that takes the robot from its initial pose to the desired goal pose **without colliding with obstacles** in the environment.

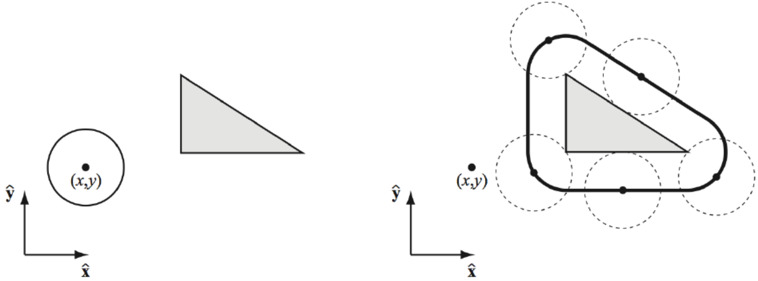


Universität Bremen

For simplicity,

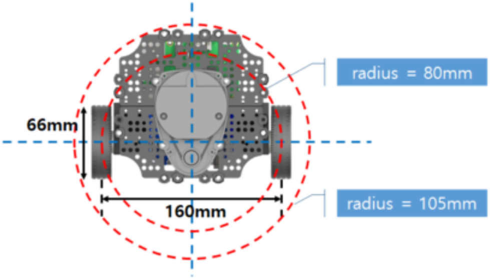
- We reduce the robot to a point object, and
- We grow or enlarge the obstacles in the map by the original size of the robot.

As a result, any pose outside these enlarged boundaries does not collide with the obstacles.



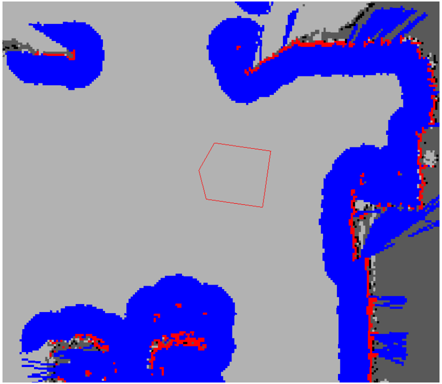
(a) (b)

Inflating Obstacles in an Occupancy Grid Map



Robot radius to inflate obstacles

https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications

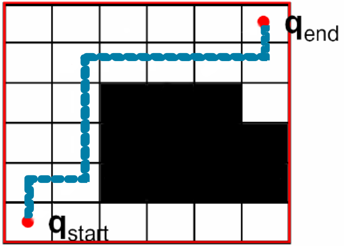


Occupancy grid map with inflated obstacles

http://library.isr.ist.utl.pt/docs/roswiki/costmap_2d.html

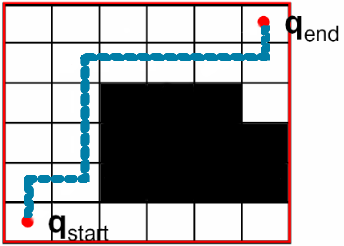
- In this lecture, we examine path planning algorithms for **fully known** environments with **static obstacles**. That is,
  - The entire environment has been explored.
  - The locations of all obstacles have been identified.
  - The obstacles never change their location.
  - The mobile robot is the only moving object in the environment.

Path Planning Algorithms



A grid map with obstacles

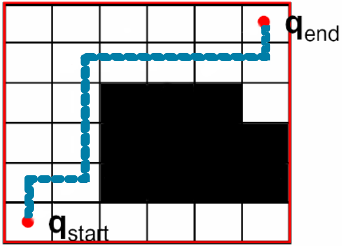
- **Path**: A sequence of free cells in the grid map.
  - First cell on path $\rightarrow$ *start* cell
  - Last cell on path $\rightarrow$ *end* cell
- **Path Length**: No. of cells to be traversed in order to reach the *end* cell from the *start* cell.



A grid map with obstacles

- **Path**: A sequence of free cells in the grid map.
  - First cell on path $\rightarrow$ *start* cell
  - Last cell on path $\rightarrow$ *end* cell
- **Path Length**: No. of cells to be traversed in order to reach the *end* cell from the *start* cell. In the example on the left, the length of the path marked in blue is 10.

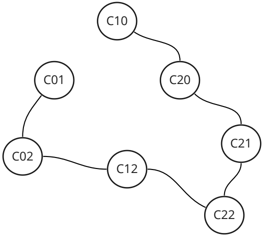
German Research Center for Artificial Intelligence



A grid map with obstacles

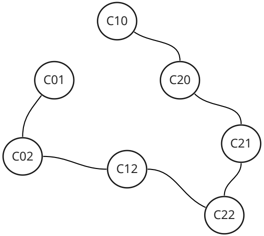
- **Path**: A sequence of free cells in the grid map.
  - First cell on path → *start* cell
  - Last cell on path → *end* cell
- **Path Length**: No. of cells to be traversed in order to reach the *end* cell from the *start* cell. In the example on the left, the length of the path marked in blue is 10.
- **Objective:** To find the **shortest path** from *start* to *goal* in a given grid map.

- Path planning algorithms need **adjacency** and **connectivity** information.
  - Which cells are located next to each other and are reachable from one another?
- This information is best represented in a **graph**.



German Research Center for Artificial Intelligence

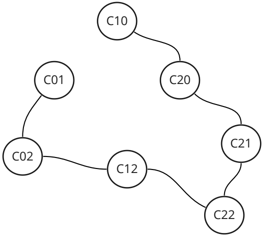
- Path planning algorithms need **adjacency** and **connectivity** information.
  - Which cells are located next to each other and are reachable from one another?
- This information is best represented in a **graph**.



Graph

- A graph consists of **nodes** and **edges** between nodes.

- Path planning algorithms need **adjacency** and **connectivity** information.
  - Which cells are located next to each other and are reachable from one another?
- This information is best represented in a **graph**.

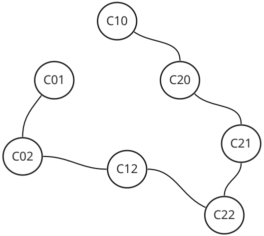


Graph

- A graph consists of **nodes** and **edges** between nodes.
- The edges represent relationships between nodes.

German Research Center for Artificial Intelligence

- Path planning algorithms need **adjacency** and **connectivity** information.
  - Which cells are located next to each other and are reachable from one another?
- This information is best represented in a **graph**.

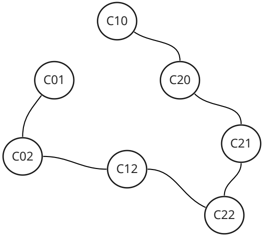


Graph

- A graph consists of **nodes** and **edges** between nodes.
- The edges represent relationships between nodes.
- E.g., if nodes are locations, then edges could represent the connectivity between two adjacent locations.

Universität Bremen

- Path planning algorithms need **adjacency** and **connectivity** information.
  - Which cells are located next to each other and are reachable from one another?
- This information is best represented in a **graph**.

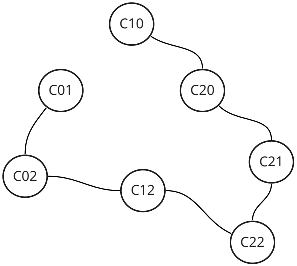


Graph

- A graph consists of **nodes** and **edges** between nodes.
- The edges represent relationships between nodes.
- E.g., if nodes are locations, then edges could represent the connectivity between two adjacent locations.
- Edges can be assigned **weights**. E.g., distance between two adjacent locations.

Path Planning
From Grid Maps to Graphs

	0	1	2
0		C10	C20
1	C01		C21
2	C02	C12	C22



Create a graph from grid map

- Create a node for each **free cell**.
- If two free cells are adjacent to each other, then add an edge between the corresponding nodes in the graph.

German Research Center for Artificial Intelligence

So far, we learned to:

- Define a path in an occupancy grid map.
- Define the length of a path in an occupancy grid map.
- Create a graph connecting the adjacent free cells of an occupancy grid map.

Universität Bremen

So far, we learned to:

- Define a path in an occupancy grid map.
- Define the length of a path in an occupancy grid map.
- Create a graph connecting the adjacent free cells of an occupancy grid map.

Now, let us look at the first path planning algorithm – Breadth-first search.

One possible search strategy is the **Breadth-First Search**:

Given: Graph, start node, goal node.

German Research Center for Artificial Intelligence

One possible search strategy is the Breadth-First Search:

Given: Graph, start node, goal node.

1. Begin search at the *start* node and do a test for *goal*.
   1.1 If *start* is same as *goal*, then stop search.
   1.2 Otherwise, continue search.

One possible search strategy is the **Breadth-First Search**:

Given: Graph, start node, goal node.

1. Begin search at the *start* node and do a test for *goal*.
   1.1 If *start* is same as *goal*, then stop search.
   1.2 Otherwise, continue search.
2. Visit the adjacent nodes of *start* and test whether any of them is *goal*.

German Research Center for Artificial Intelligence

One possible search strategy is the **Breadth-First Search**:

Given: Graph, start node, goal node.

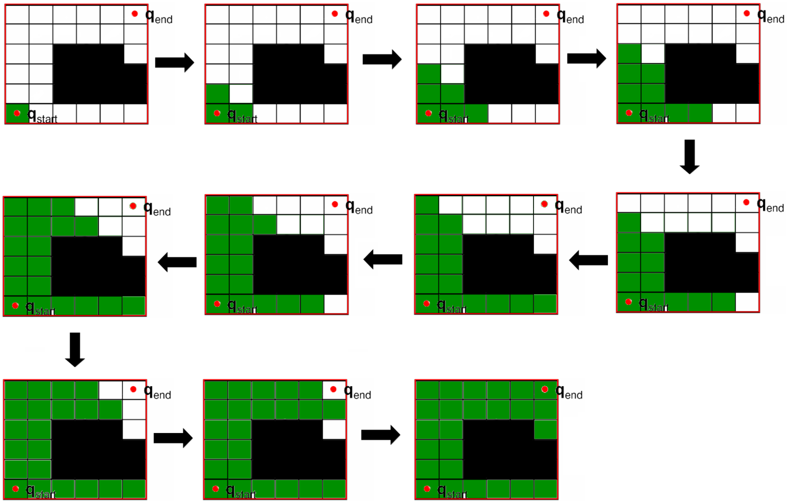
1. Begin search at the *start* node and do a test for *goal*.
   1.1 If *start* is same as *goal*, then stop search.
   1.2 Otherwise, continue search.
2. Visit the adjacent nodes of *start* and test whether any of them is *goal*.
3. If *goal* not found, then continue search at the adjacent nodes of the nodes visited in previous step. Skip any node that has already been visited.

One possible search strategy is the **Breadth-First Search**:

Given: Graph, start node, goal node.

1. Begin search at the *start* node and do a test for *goal*.
   1.1 If *start* is same as *goal*, then stop search.
   1.2 Otherwise, continue search.
2. Visit the adjacent nodes of *start* and test whether any of them is *goal*.
3. If *goal* not found, then continue search at the adjacent nodes of the nodes visited in previous step. Skip any node that has already been visited.
4. Continue Step 3, until either the *goal* is found or until all nodes that can be reached from *start* node have been visited.

Search Strategy: Breadth-First Search – An Illustration



Universität Bremen

German Research Center for Artificial Intelligence

German Research Center for Artificial Intelligence

Exhaustive or brute-force search strategy

- Enumerate systematically all possible candidates and test if any of them is the goal.
- In breadth-first search, all paths of length 'n' from *start* are checked before checking the paths of length '$n+1$', for all $n >= 0$.

Universität Bremen

German Research Center for Artificial Intelligence

Creates and maintains three lists:

1. A list of **visited** nodes.
   - These nodes have failed the test for goal node.

Creates and maintains three lists:

1. A list of **visited** nodes.
   - These nodes have failed the test for goal node.
2. A list of **frontier** nodes.
   - Frontier nodes are the neighbours of already visited nodes that have not been visited (goal-tested) yet.

German Research Center for Artificial Intelligence

Creates and maintains three lists:

1. A list of **visited** nodes.
   - These nodes have failed the test for goal node.
2. A list of **frontier** nodes.
   - Frontier nodes are the neighbours of already visited nodes that have not been visited (goal-tested) yet.
3. A list of **predecessor** nodes.
   - Predecessor is recorded when a **node is expanded**, i.e. when its neighbours are added to the frontier list.

Path Planning Algorithms
Breadth-First Search – Pseudocode


```
procedure breadth_first_search(Graph, start, goal):
  current_node ← none
  frontier ← a queue initialised with start
  visited ← empty set
  predecessor ← empty set

  do
    if frontier is empty then
      return failure
    current_node ← frontier.pop()
    if current_node is goal then
      path ← predecessor.backtrace(current_node)
      return path
    visited.add(current_node)
    for each of current_node's neighbors q do
      if q is not in visited then
        frontier.add(q)
        predecessor.add(q, current_node)
```

- Lines 2–5 ------> Initialisation
- `if frontier is empty then` ------> No more nodes to explore
- `current_node ← frontier.pop()` ------> Remove node at the front of queue
- `path ← predecessor.backtrace(current_node)` ------> Trace path back to start
- `for each of current_node's neighbors q do` ------> Expand current_node
- `predecessor.add(q, current_node)` ------> Record the predecessor of q

German Research Center for Artificial Intelligence

Pros

- Simple and easy to implement.
- If a path exists from start to goal, it will always find the shortest path (complete and optimal).
- Suitable for smaller grids (fewer cells).

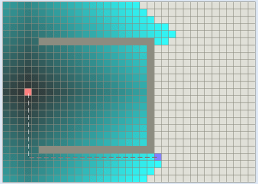
Universität Bremen

Pros

- Simple and easy to implement.
- If a path exists from start to goal, it will always find the shortest path (complete and optimal).
- Suitable for smaller grids (fewer cells).

Cons

- Searches ‘blindly’. Only adjacency or neighbourhood information is used. (Uninformed search)



© 2021 Amit Patel (Source: `http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html`)

- Inefficient, especially for large grids populated sparsely with obstacles.

- Informed search strategies use **knowledge about the problem domain** to guide the search more efficiently.
- They evaluate each node using a **cost function**.
  - In path planning problems, the most commonly used cost functions include the **estimated distance to the goal node** and the **actual distance from the start node**.

- Informed search strategies use **knowledge about the problem domain** to guide the search more efficiently.
- They evaluate each node using a **cost function**.
  - In path planning problems, the most commonly used cost functions include the **estimated distance to the goal node** and the **actual distance from the start node**.
- Informed search strategies are **greedy best-first** approaches.
  - Instead of exploring all nodes with equal priority, informed search prioritises the exploration of those nodes that have a lower cost.

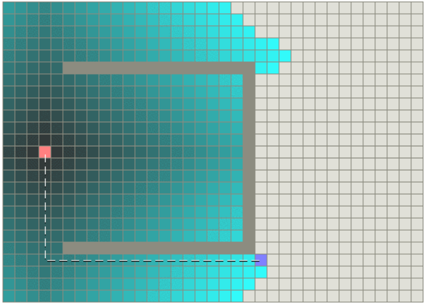
- **A*** (pronounced **“A-star”**) is the most well-known and widely used best-first search algorithm.

- **A*** (pronounced **"A-star"**) is the most well-known and widely used best-first search algorithm.

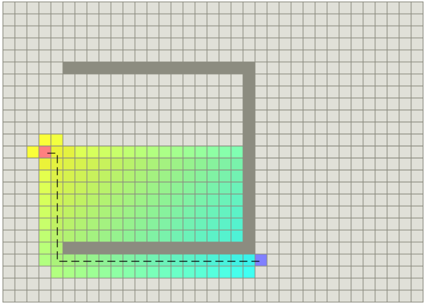
Cost function used in A* algorithm

- Heuristic $h(q)$: Estimated length of the path from the node q to the goal node.
- Exact cost $g(q)$: Length of the actual path from the start node to the node q.
- Total cost for node q: $f(q) = h(q) + g(q)$

Path Planning Algorithms
A* versus Breadth-First Search



Breadth-first search



A* (best-first search)

German Research Center for Artificial Intelligence

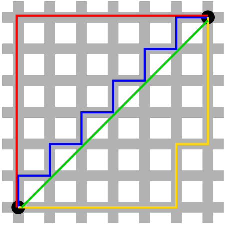
- To find an optimal (least-cost) path from start to goal, the heuristic used by A* should be ***admissible***, i.e. it should never overestimate the actual cost of getting to goal node from the current node.

Universität Bremen

- To find an optimal (least-cost) path from start to goal, the heuristic used by A* should be ***admissible***, i.e. it should never overestimate the actual cost of getting to goal node from the current node.

Examples of admissible heuristic:

- For a continuous environment (e.g., outdoor feature maps): Euclidean distance, spherical distance, etc.
- For a discrete environment (e.g., indoor occupancy grid maps): Manhattan distance.



2D space: Euclidean distance is show by the green-coloured straight line. Manhattan distance is shown by the blue, red and yellow lines.

- Euclidean distance between 2D points $(x1, y1)$ and $(x2, y2)$:
  $\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$
- Manhattan distance between 2D points $(x1, y1)$ and $(x2, y2)$:
  $|x2 - x1| + |y2 - y1|$

(Can be extended to higher dimensional spaces.)

Path Planning Algorithms
A* – Pseudocode

German Research Center for Artificial Intelligence

```
procedure a_star(Graph, start, goal):
    current_node ← none
    cost ← 0 + heuristic(start)
    frontier ← a priority queue initialised with (start, cost)          ------> Initialisation
    visited ← empty set
    predecessor ← empty set

    do
        if frontier is empty then         ------> No more nodes to explore
            return failure
        current_node ← frontier.pop()  ------> Remove node at the front of priority queue
        if current_node is goal then
            path ← predecessor.backtrace(current_node) ------> Trace path back to start
            return path
        visited.add(current_node)
        for each of current_node's neighbors q do  ------> Expand current_node
            if q is not in visited then
                cost ← exact-cost(q) + heuristic(q)  ------> Compute total cost
                frontier.add(q, cost)
                predecessor.add(q, current_node)   ------> Record the predecessor of q
```

Marked in blue are the changes with respect to breadth-first search.

- Dijkstra's algorithm is also a best-first search algorithm.

- Dijkstra's algorithm is also a best-first search algorithm.
- However, unlike A*, its cost function has only one term, namely the exact cost $g(q)$. This is the actual cost of moving from the start node to node q.

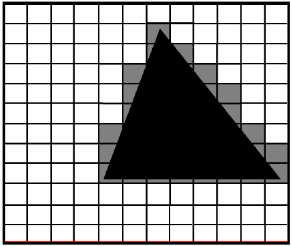
- Dijkstra's algorithm is also a best-first search algorithm.
- However, unlike A*, its cost function has only one term, namely the exact cost $g(q)$. This is the actual cost of moving from the start node to node q.
- If the actual cost of moving from a node to an adjacent node is the identical for all nodes in the graph, then Dijkstra's algorithm is identical to breadth-first search.

- Dijkstra's algorithm is also a best-first search algorithm.
- However, unlike A*, its cost function has only one term, namely the exact cost $g(q)$. This is the actual cost of moving from the start node to node q.
- If the actual cost of moving from a node to an adjacent node is the identical for all nodes in the graph, then Dijkstra's algorithm is identical to breadth-first search.

ROS 2 Navigation stack includes Dijkstra's and A* path planners.

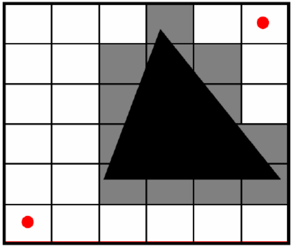
Influence of Grid Map Resolution

- More precise obstacle boundaries.
- High no. of free cells and nodes in graph.



High resolution

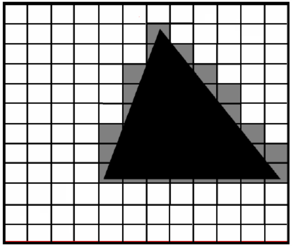
- Imprecise obstacle boundaries.
- Fewer no. of free cells and nodes in graph.



Low resolution

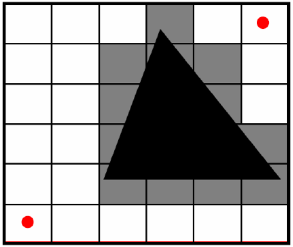
Influence of Grid Map Resolution

- More precise obstacle boundaries.
- High no. of free cells and nodes in graph.



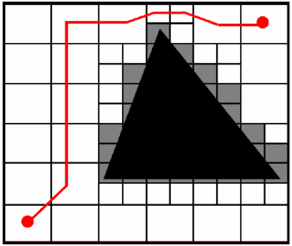
High resolution

- Imprecise obstacle boundaries.
- Fewer no. of free cells and nodes in graph.



Low resolution

- More precise obstacle boundaries.
- Fewer no. of free cells and nodes in graph



Variable resolution

Obstacle Avoidance and Velocity Control

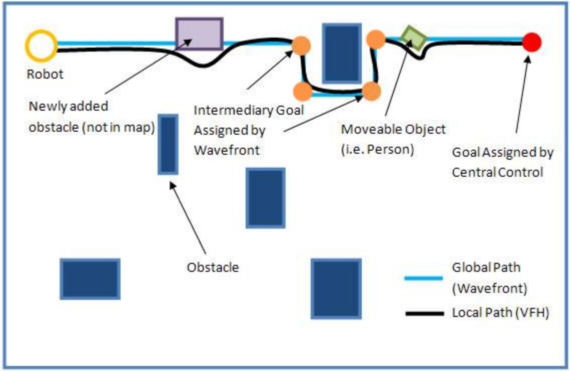
- Path planning finds a **globally optimal path** to reach the goal pose from the initial pose.
- However, it assumes a **static** world, i.e. the positions of obstacles are fixed and fully known.

German Research Center for Artificial Intelligence

- Path planning finds a **globally optimal path** to reach the goal pose from the initial pose.
- However, it assumes a **static** world, i.e. the positions of obstacles are fixed and fully known.
- But, **real-world environments are dynamic**.
  - Obstacles may **change position** dynamically (e.g. furniture may get displaced).
  - New obstacles **may appear on the scene** (e.g. something falls down from the overhead shelf).
  - Obstacles may be **constantly moving** (e.g. people or other robots moving through corridors).

- Path planning finds a **globally optimal path** to reach the goal pose from the initial pose.
- However, it assumes a **static** world, i.e. the positions of obstacles are fixed and fully known.
- But, **real-world environments are dynamic**.
  - Obstacles may **change position** dynamically (e.g. furniture may get displaced).
  - New obstacles **may appear on the scene** (e.g. something falls down from the overhead shelf).
  - Obstacles may be **constantly moving** (e.g. people or other robots moving through corridors).
- Global path planning is slow and hence not suitable for fast **obstacle avoidance**.

To avoid collisions in dynamic environments, we also need **fast, local, reactive approaches** that continuously generate appropriate **linear and angular velocities** to **safely steer** the robot around obstacles on its way to the goal.



(Source: http://www.eng.uwaterloo.ca/~smasiada/FirebotsReport_files/image040.jpg)

Obstacle Avoidance

Local Approaches: Basic Idea

1. **Sensor data** (e.g. lidar, sonar) is used to continuously **detect obstacles** as the robot moves.

German Research Center for Artificial Intelligence

1. **Sensor data** (e.g. lidar, sonar) is used to continuously **detect obstacles** as the robot moves.
2. **Odometry data** is used to obtain the **robot's current pose and velocity**.

1. **Sensor data** (e.g. lidar, sonar) is used to continuously **detect obstacles** as the robot moves.
2. **Odometry data** is used to obtain the **robot's current pose and velocity**.
3. A set of **possible solutions (candidates)** are generated based on sensor and odometry data to steer the robot.

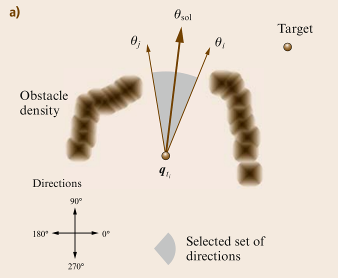
1. **Sensor data** (e.g. lidar, sonar) is used to continuously **detect obstacles** as the robot moves.
2. **Odometry data** is used to obtain the **robot's current pose and velocity**.
3. A set of **possible solutions (candidates)** are generated based on sensor and odometry data to steer the robot.
4. An **objective function** evaluates each possible solution.

1. **Sensor data** (e.g. lidar, sonar) is used to continuously **detect obstacles** as the robot moves.
2. **Odometry data** is used to obtain the **robot's current pose and velocity**.
3. A set of **possible solutions (candidates)** are generated based on sensor and odometry data to steer the robot.
4. An **objective function** evaluates each possible solution.
5. The **best (optimal) solution** is selected and used to control the robot's linear and angular velocities.

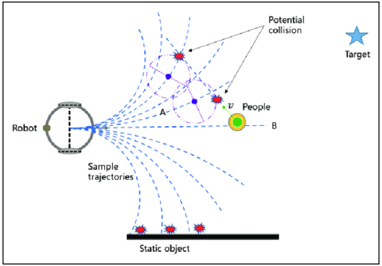
1. **Sensor data** (e.g. lidar, sonar) is used to continuously **detect obstacles** as the robot moves.
2. **Odometry data** is used to obtain the **robot's current pose and velocity**.
3. A set of **possible solutions (candidates)** are generated based on sensor and odometry data to steer the robot.
4. An **objective function** evaluates each possible solution.
5. The **best (optimal) solution** is selected and used to control the robot's linear and angular velocities.

Only a local region around the robot or a short time window is considered at a time for obstacle avoidance ==> **locally optimal motion control**.

- Depending on the **nature of the solutions**, obstacle avoidance approaches can be categorized into:
  1. **Motion direction** based methods: e.g Vector Field Histogram (VFH)
  2. **Velocity control** based methods: e.g. Dynamic Window Approach (DWA)



VFH Approach (Fig. 35.10a, Page 840, Springer Handbook of Robotics, 2008 Edition)



DWA (https://www.researchgate.net/publication/317584521/figure/fig10/AS:505271367344128@1497477489107/Illustration-of-the-improved-DWA-method-DWA-dynamic-window-approach.png)

German Research Center for Artificial Intelligence

Main steps in DWA:

1. Find the set of velocities that the robot can reach within a short time interval of T seconds. This set of velocities is called the **dynamic window**.

German Research Center for Artificial Intelligence

Main steps in DWA:

1. Find the set of velocities that the robot can reach within a short time interval of T seconds. This set of velocities is called the **dynamic window**.
2. Determine the velocities in the dynamic window that are safe or **admissible**.

German Research Center for Artificial Intelligence

Main steps in DWA:

1. Find the set of velocities that the robot can reach within a short time interval of T seconds. This set of velocities is called the **dynamic window**.
2. Determine the velocities in the dynamic window that are safe or **admissible**.
3. Compute a **score** for each admissible velocity in the dynamic window.

German Research Center for Artificial Intelligence

Main steps in DWA:

1. Find the set of velocities that the robot can reach within a short time interval of T seconds. This set of velocities is called the **dynamic window**.
2. Determine the velocities in the dynamic window that are safe or **admissible**.
3. Compute a **score** for each admissible velocity in the dynamic window.
4. Choose the velocity with the **highest score**.

German Research Center for Artificial Intelligence

Main steps in DWA:

1. Find the set of velocities that the robot can reach within a short time interval of T seconds. This set of velocities is called the **dynamic window**.
2. Determine the velocities in the dynamic window that are safe or **admissible**.
3. Compute a **score** for each admissible velocity in the dynamic window.
4. Choose the velocity with the **highest score**.
5. **Steer** the robot at the chosen velocity.

German Research Center for Artificial Intelligence

Main steps in DWA:

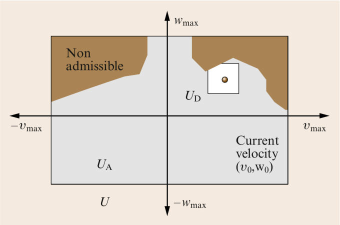
1. Find the set of velocities that the robot can reach within a short time interval of T seconds. This set of velocities is called the **dynamic window**.
2. Determine the velocities in the dynamic window that are safe or **admissible**.
3. Compute a **score** for each admissible velocity in the dynamic window.
4. Choose the velocity with the **highest score**.
5. **Steer** the robot at the chosen velocity.
6. **Repeat** the above procedure (steps 1 to 5) until the robot arrives at the goal.

- **Original paper:** D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in IEEE Robotics & Automation Magazine, 1997. Click here for full text.

- **Velocity**: An ordered pair of robot velocities (v, ω); v is the linear velocity, ω is the angular velocity.

German Research Center for Artificial Intelligence

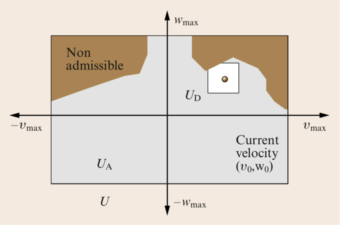
- **Velocity**: An ordered pair of robot velocities (v, ω); v is the linear velocity, ω is the angular velocity.
- **Velocity space** U: The 2D space containing all possible velocities permissible for the robot.
  - Defined by the min and max values for v and ω.
  - $v \in [-v_{max}, v_{max}]$
  - $\omega \in [-\omega_{max}, \omega_{max}]$



2D space of permissible velocities (Fig. 35.12, Page 842, Springer Handbook of Robotics, 2008 Edition)

German Research Center for Artificial Intelligence

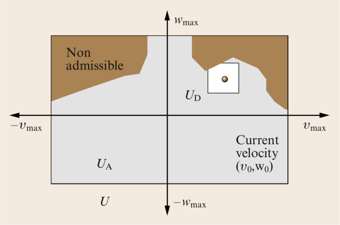
- **Admissible velocities** U_A: The set of all velocities in U that produce **safe trajectories**.
  - That is, the robot can be stopped before colliding with nearest obstacle by applying the maximum permissible deceleration $(a^v_{max}, a^\omega_{max})$.



Admissible velocities are marked in gray (Fig. 35.12, Page 842, Springer Handbook of Robotics, 2008 Edition)

German Research Center for Artificial Intelligence

- **Dynamic window** U_D: The set of all velocities (v, ω) that can be reached by the robot in a short time interval (e.g. 0.25 seconds), by applying the limited set of possible robot accelerations.
  - Dynamic window is computed around the current velocity.



Dynamic window is marked in white. Current velocity (v_0, ω_0) **is shown by the brown dot.** (Fig. 35.12, Page 842, Springer Handbook of Robotics, 2008 Edition)

Candidate velocities U_C:

The set of all **admissible** velocities that lie inside the **dynamic window**.

$$U_C = U \cap U_A \cap U_D$$

German Research Center for Artificial Intelligence

Candidate velocities U_C:

The set of all **admissible** velocities that lie inside the **dynamic window**.

$$U_C = U \cap U_A \cap U_D$$

Objective Function $F(v, \omega)$:

- Each candidate (v, ω) defines a trajectory.
- Objective function $F(v, \omega)$ evaluates each trajectory and assigns a score.

Universität Bremen

German Research Center for Artificial Intelligence

Objective function $F(v, \omega)$:

- The objective function $F(v, \omega)$ evaluates trajectories based on **three criteria**:

German Research Center for Artificial Intelligence

Objective function $F(v, \omega)$:

- The objective function $F(v, \omega)$ evaluates trajectories based on **three criteria**:
  1. **Closeness to goal pose**: How close to the goal pose would this trajectory bring the robot?

Universität Bremen

German Research Center for Artificial Intelligence

Objective function $F(v, \omega)$:

- The objective function $F(v, \omega)$ evaluates trajectories based on **three criteria**:
  1. **Closeness to goal pose**: How close to the goal pose would this trajectory bring the robot?
  2. **Clearance from nearest obstacles**: How far is the nearest obstacle on this trajectory?

German Research Center for Artificial Intelligence

Objective function $F(v, \omega)$:

- The objective function $F(v, \omega)$ evaluates trajectories based on **three criteria**:
  1. **Closeness to goal pose**: How close to the goal pose would this trajectory bring the robot?
  2. **Clearance from nearest obstacles**: How far is the nearest obstacle on this trajectory?
  3. **Speed of motion**: How fast does the robot move on this trajectory?

German Research Center for Artificial Intelligence

Objective function $F(v, \omega)$:

- The objective function $F(v, \omega)$ evaluates trajectories based on **three criteria**:
  1. **Closeness to goal pose**: How close to the goal pose would this trajectory bring the robot?
  2. **Clearance from nearest obstacles**: How far is the nearest obstacle on this trajectory?
  3. **Speed of motion**: How fast does the robot move on this trajectory?
- The objective is to drive in the **correct direction** as **fast** as possible while staying as **far away from obstacles** as possible.

http://ais.informatik.uni-freiburg.de/teaching/ss03/ams/colli02.pdf

http://ais.informatik.uni-freiburg.de/teaching/ss03/ams/colli02.pdf

- **Strength**: Fast approach for obstacle avoidance.

http://ais.informatik.uni-freiburg.de/teaching/ss03/ams/colli02.pdf

- **Strength**: Fast approach for obstacle avoidance.
- **Weakness**: Difficulty to enter narrow passages and doorways (robot does not stop on time).

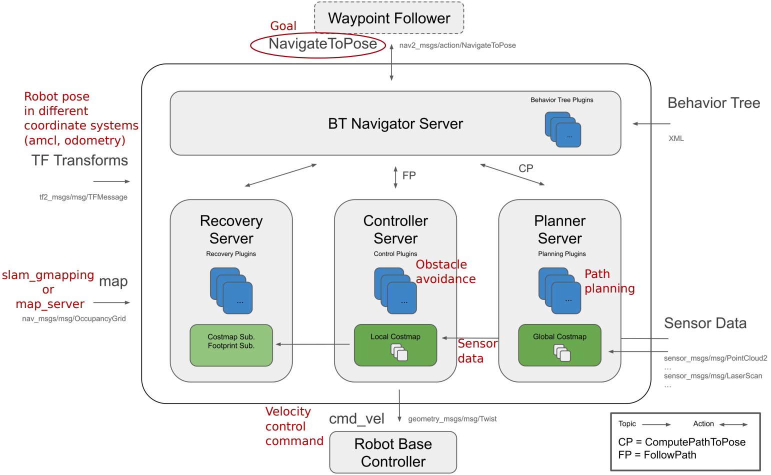
German Research Center for Artificial Intelligence

http://ais.informatik.uni-freiburg.de/teaching/ss03/ams/colli02.pdf

- **Strength**: Fast approach for obstacle avoidance.
- **Weakness**: Difficulty to enter narrow passages and doorways (robot does not stop on time).
- **Improved DWA**: Optimize position and velocity simultaneously.

Robot Navigation: The Complete Picture

ROS 2 Navigation Stack



https://navigation.ros.org/

Criteria	Path planning	Obstacle avoidance
Input	Initial pose, goal pose, global map	Waypoint, local map, current pose and current velocity
Output	Globally optimal path (sequence of robot poses)	Locally optimal velocity control command
Environment	Assumes static environment	Designed for dynamic environments
Dynamic constraints	Not taken into account	Motion constraints considered

Summary

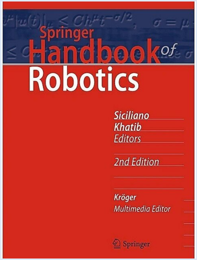
Key Topics Covered in This Lecture

- Definition of a path
- Definition of path planning
- Pre-requisites for path planning in mobile robots
- Path planning algorithms: breadth-first search, A*, Dijkstra's
- Obstacle avoidance using dynamic window approach
- ROS 2 navigation stack
- Comparison of path planning and obstacle avoidance

References

Springer Handbook of Robotics (English)

- Chapter 5.1: Motion Planning Concepts
- Chapter 5.3: Alternative Approaches
- Chapter 35.7: From Motion Planning to Obstacle Avoidance
- Chapter 35.8: Definition of Obstacle Avoidance
- Chapter 35.9: Obstacle Avoidance Techniques



Source: https://link.springer.com/referencework/10.1007%2F978-3-540-30301-5

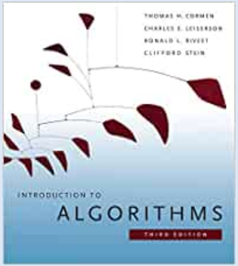
Mobile Roboter (German)

- Chapter 7.4: Pfadplanung
- Chapter 7.6: Planbasierte Robotersteuerung

Source: `https://link.springer.com/book/10.1007/978-3-642-01726-1`

Introduction to Algorithms (English)

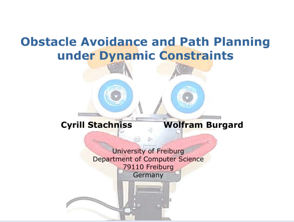
- Chapter 24: Single-Source Shortest Paths



Source: `https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf`

Additional Literature

Obstacle Avoidance and Path Planning under Dynamic Constraints, Uni Freiburg


Obstacle Avoidance and Path Planning
under Dynamic Constraints
Cyrill Stachniss
Wolfram Burgard
University of Freiburg
Department of Computer Science
79110 Freiburg
Germany

Source: `http://ais.informatik.uni-freiburg.de/teaching/ss03/ams/colli02.pdf`

Universität Bremen

Thank You for Your Attention.