Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

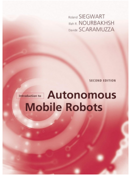
Institute for AI and Autonomous Systems

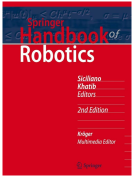
Path Planning

How a Robot Finds Its Way Around

Dr. Alex Mitrevski
Master of Autonomous Systems

Structure

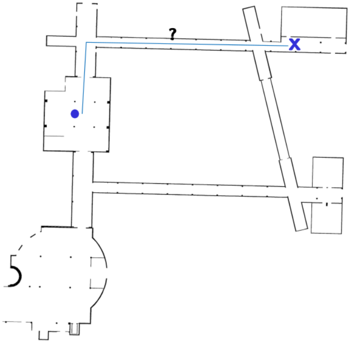




- Path planning preliminaries
- Path planning algorithms
- Local obstacle avoidance

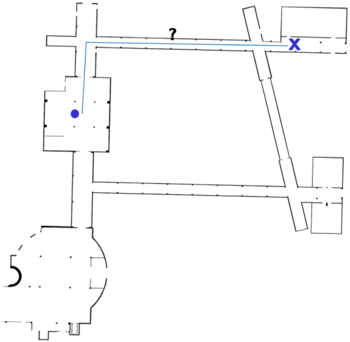
Path Planning Preliminaries

What is Path Planning?



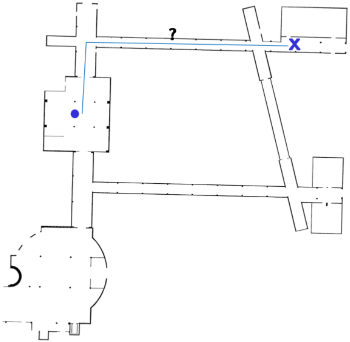
- Path planning is concerned with the problem of finding **a collision-free path $\mathcal{P}$ that brings a robot from a starting pose P_s to a goal pose P_g**

What is Path Planning?



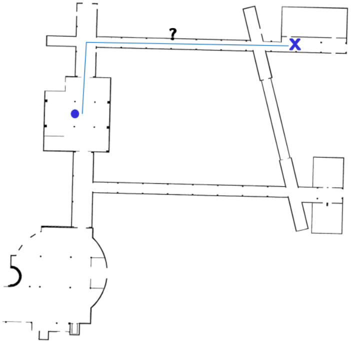
- Path planning is concerned with the problem of finding **a collision-free path $\mathcal{P}$ that brings a robot from a starting pose P_s to a goal pose P_g**
- Typically, $\mathcal{P}$ is defined as **a sequence of poses** $\mathcal{P} = (P_s, P_1, ..., P_n, P_g)$ through which the robot should pass

What is Path Planning?



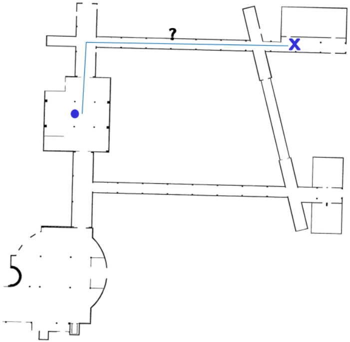
- Path planning is concerned with the problem of finding **a collision-free path $\mathcal{P}$ that brings a robot from a starting pose P_s to a goal pose P_g**
- Typically, $\mathcal{P}$ is defined as **a sequence of poses** $\mathcal{P} = (P_s, P_1, ..., P_n, P_g)$ through which the robot should pass
- There are two main questions that are of relevance for any path planning algorithm:

What is Path Planning?



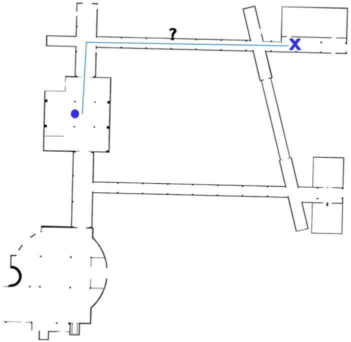
- Path planning is concerned with the problem of finding **a collision-free path $\mathcal{P}$ that brings a robot from a starting pose P_s to a goal pose P_g**
- Typically, $\mathcal{P}$ is defined as **a sequence of poses** $\mathcal{P} = (P_s, P_1, ..., P_n, P_g)$ through which the robot should pass
- There are two main questions that are of relevance for any path planning algorithm:
  1. **How to generate $\mathcal{P}$?**

What is Path Planning?



- Path planning is concerned with the problem of finding **a collision-free path $\mathcal{P}$ that brings a robot from a starting pose P_s to a goal pose P_g**
- Typically, $\mathcal{P}$ is defined as **a sequence of poses** $\mathcal{P} = (P_s, P_1, ..., P_n, P_g)$ through which the robot should pass
- There are two main questions that are of relevance for any path planning algorithm:
  1. **How to generate $\mathcal{P}$?**
  2. **How to ensure that $\mathcal{P}$ does not bring a robot into collisions?**

What is Path Planning?



- Path planning is concerned with the problem of finding **a collision-free path $\mathcal{P}$ that brings a robot from a starting pose P_s to a goal pose P_g**
- Typically, $\mathcal{P}$ is defined as **a sequence of poses** $\mathcal{P} = (P_s, P_1, ..., P_n, P_g)$ through which the robot should pass
- There are two main questions that are of relevance for any path planning algorithm:
  1. **How to generate $\mathcal{P}$?**
  2. **How to ensure that $\mathcal{P}$ does not bring a robot into collisions?**
- Note that path planning **requires an environment map to be performed** — obstacles need to be known so that collisions with them can be avoided

Soundness and Completeness

Soundness

The property of a path planning algorithm to produce valid paths that start at P_s and end at P_g

Soundness and Completeness

Soundness

The property of a path planning algorithm to produce valid paths that start at P_s and end at P_g

- An essential property of path planning algorithms — **an algorithm that is not sound cannot be relied on by a robot**

Soundness and Completeness

Soundness

The property of a path planning algorithm to produce valid paths that start at P_s and end at P_g

- An essential property of path planning algorithms — **an algorithm that is not sound cannot be relied on by a robot**

Completeness

A path planning algorithm is complete if, whenever a path from P_s to P_g exists, the algorithm will find it

Soundness and Completeness

Soundness

The property of a path planning algorithm to produce valid paths that start at P_s and end at P_g

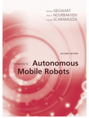
- An essential property of path planning algorithms — **an algorithm that is not sound cannot be relied on by a robot**

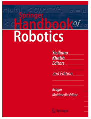
Completeness

A path planning algorithm is complete if, whenever a path from P_s to P_g exists, the algorithm will find it

- A desirable property of path planning algorithms that **cannot always be guaranteed** — due to the complexity of a planning configuration, **it may be impossible to find a path within a given time or memory budget**

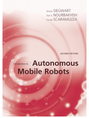
Configuration Space

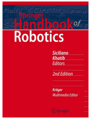




- Path planning needs to take into account the fact that a robot is not a point in space, but a full body
  - Achieved by planning not in the robot's physical space, but **in configuration space**, where each configuration is a point

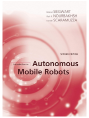
Configuration Space

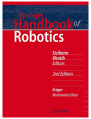




- Path planning needs to take into account the fact that a robot is not a point in space, but a full body
  - Achieved by planning not in the robot's physical space, but **in configuration space**, where each configuration is a point
- The configuration space $\mathcal{C}$ (aka C-space) is **a space of all configurations q that a robot can occupy**
  - For a planar navigating robot, the configuration space can be defined by planar poses $\boldsymbol{q} = (x, y, \theta)$

Configuration Space





- Path planning needs to take into account the fact that a robot is not a point in space, but a full body
  - Achieved by planning not in the robot's physical space, but **in configuration space**, where each configuration is a point
- The configuration space $\mathcal{C}$ (aka C-space) is **a space of all configurations q that a robot can occupy**
  - For a planar navigating robot, the configuration space can be defined by planar poses $\boldsymbol{q} = (x, y, \theta)$
- If $\mathcal{O} \subset \mathcal{W}$ is a workspace region occupied by an obstacle and $\mathcal{A}(\boldsymbol{q}) \subset \mathcal{W}$ is the set of workspace points occupied by a robot in $\boldsymbol{q}$, the **occupied region** in C-space is

$$C_{obs} = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset\}$$

Configuration Space

- Path planning needs to take into account the fact that a robot is not a point in space, but a full body
  - Achieved by planning not in the robot's physical space, but **in configuration space**, where each configuration is a point
- The configuration space $\mathcal{C}$ (aka C-space) is **a space of all configurations q that a robot can occupy**
  - For a planar navigating robot, the configuration space can be defined by planar poses $\boldsymbol{q} = (x, y, \theta)$
- If $\mathcal{O} \subset \mathcal{W}$ is a workspace region occupied by an obstacle and $\mathcal{A}(\boldsymbol{q}) \subset \mathcal{W}$ is the set of workspace points occupied by a robot in $\boldsymbol{q}$, the **occupied region** in C-space is

$$C_{obs} = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset\}$$

- **Obstacles are typically enlarged in the C-space**, and a valid path is one that **passes only through the free space** $C_{free} = C \setminus C_{obs}$

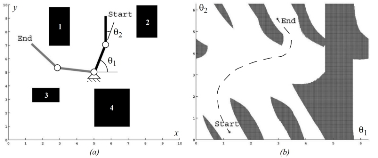
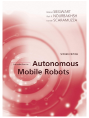
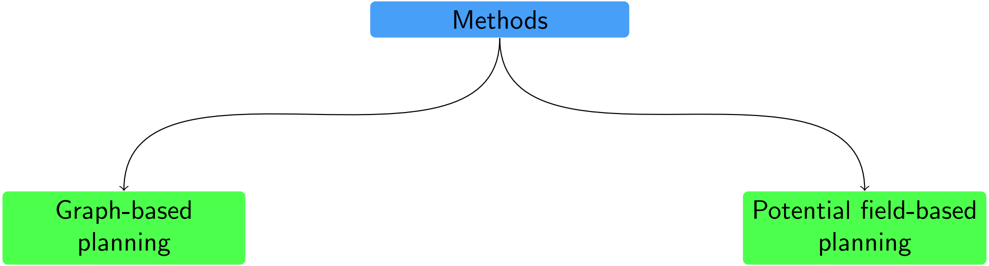


Figure 6.1
Physical space (a) and configuration space (b): (a) A two-link planar robot arm has to move from the configuration *start* to *end*. The motion is thereby constraint by the obstacles 1 to 4. (b) The corresponding configuration space shows the free space in joint coordinates (angle θ_1 and θ_2) and a path that achieves the goal.

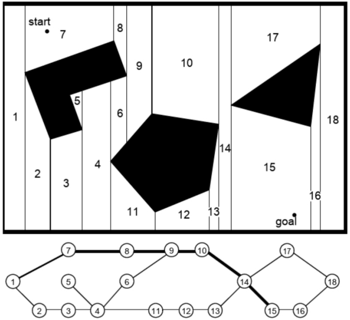
Path Planning Algorithms

Path Planning Methods



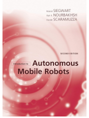


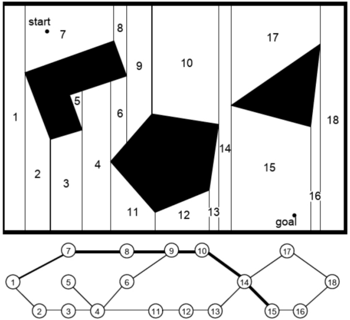
Graph Search



- The most common strategy for path planning is to **perform a path search from P_s to P_g on a graph $\mathcal{G}$**

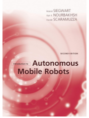
Graph Search

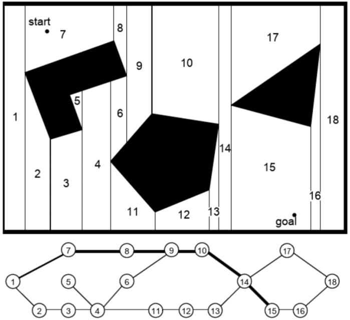




- The most common strategy for path planning is to **perform a path search from P_s to P_g on a graph $\mathcal{G}$**
- To use classical graph search for path planning, **space has to be decomposed into a set of connected regions**
  - The regions are the nodes in $\mathcal{G}$ and the connections between them are the edges

Graph Search





- The most common strategy for path planning is to **perform a path search from P_s to P_g on a graph $\mathcal{G}$**
- To use classical graph search for path planning, **space has to be decomposed into a set of connected regions**
  - The regions are the nodes in $\mathcal{G}$ and the connections between them are the edges
- The decompositions that we looked at in the last lecture (e.g. the exact cell decomposition) can be used as precursors to path planning using graph search

Graph Search Algorithms

- A variety of search algorithms can be used given a graph

Graph Search Algorithms

- A variety of search algorithms can be used given a graph

Depth-first search

Children nodes are expanded until a leaf node is reached; the search then backtracks one level and continues on

Graph Search Algorithms

- A variety of search algorithms can be used given a graph

Depth-first search

Children nodes are expanded until a leaf node is reached; the search then backtracks one level and continues on

Breadth-first search

All nodes at a given level of the search graph are expanded before expanding the nodes at the next level

Graph Search Algorithms

- A variety of search algorithms can be used given a graph

Depth-first search

Children nodes are expanded until a leaf node is reached; the search then backtracks one level and continues on

Breadth-first search

All nodes at a given level of the search graph are expanded before expanding the nodes at the next level

Djikstra's algorithm

An optimal search algorithm that selects nodes to expand based on the cost $g(n)$ of reaching n from the start node of the search

Graph Search Algorithms

- A variety of search algorithms can be used given a graph

Depth-first search

Children nodes are expanded until a leaf node is reached; the search then backtracks one level and continues on

Breadth-first search

All nodes at a given level of the search graph are expanded before expanding the nodes at the next level

Djikstra's algorithm

An optimal search algorithm that selects nodes to expand based on the cost $g(n)$ of reaching n from the start node of the search

A* algorithm

Selects nodes to expand based on a cost $f(n) = g(n) + h(n)$, where $h(n)$ is a heuristic estimate of the cost to reach the goal; optimal if $h(n)$ is admissible and consistent

Graph Search Algorithms

- A variety of search algorithms can be used given a graph

Depth-first search

Children nodes are expanded until a leaf node is reached; the search then backtracks one level and continues on

Breadth-first search

All nodes at a given level of the search graph are expanded before expanding the nodes at the next level

Djikstra's algorithm

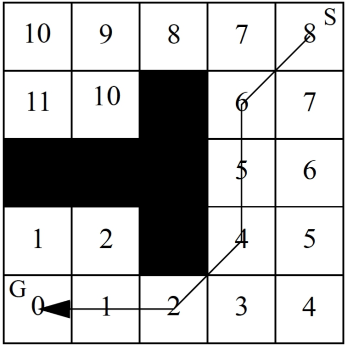
An optimal search algorithm that selects nodes to expand based on the cost $g(n)$ of reaching n from the start node of the search

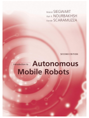
A* algorithm

Selects nodes to expand based on a cost $f(n) = g(n) + h(n)$, where $h(n)$ is a heuristic estimate of the cost to reach the goal; optimal if $h(n)$ is admissible and consistent

- These algorithms are called **deterministic search algorithm**
- More details about them are discussed in the AI course

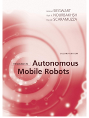
Wavefront Algorithm

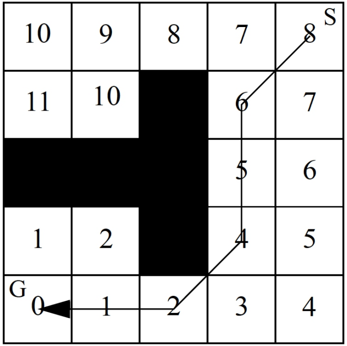




- The wavefront algorithm is a breadth-first search method that works in occupancy grids

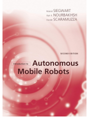
Wavefront Algorithm

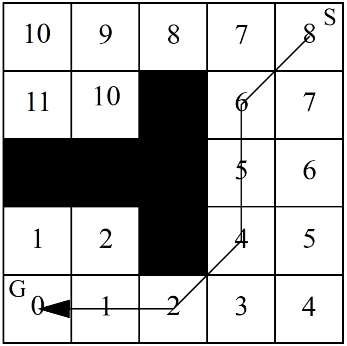




- The wavefront algorithm is a breadth-first search method that works in occupancy grids
- The algorithm **starts the search process from the goal** and **stops when a robot's initial position is reached**

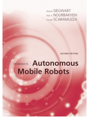
Wavefront Algorithm

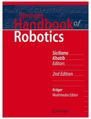




- The wavefront algorithm is a breadth-first search method that works in occupancy grids
- The algorithm **starts the search process from the goal** and **stops when a robot's initial position is reached**
- An important outcome of the wavefront algorithm is an **estimate of the distance from any expanded node to the goal** (represented as a Manhattan distance)

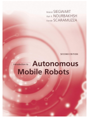
Rapidly Exploring Random Trees (RRTs)

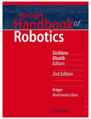




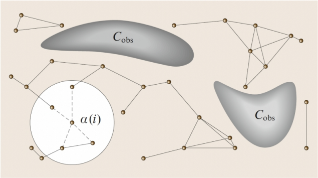
- Many robot planning tasks, particularly in high dimensions, are done using **randomised search**
  - Deterministic search tends to be inefficient — particularly under real-time constraints — and defining useful heuristics is often difficult

Rapidly Exploring Random Trees (RRTs)



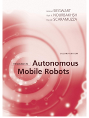


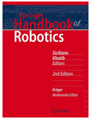
- Many robot planning tasks, particularly in high dimensions, are done using **randomised search**
  - Deterministic search tends to be inefficient — particularly under real-time constraints — and defining useful heuristics is often difficult



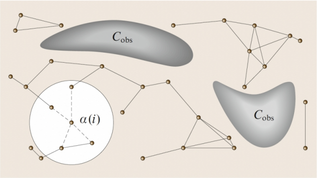
- RRT is one such algorithm that, at each step, **randomly select a free space node q'** and connects that to already an existing graph segment **if the connection leads to a collision-free path**
  - If there is a path from P_s to P_g, graph segments are likely to be connected eventually

Rapidly Exploring Random Trees (RRTs)





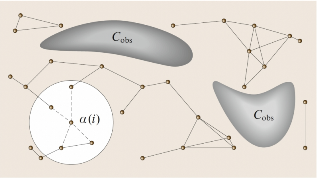
- Many robot planning tasks, particularly in high dimensions, are done using **randomised search**
  - Deterministic search tends to be inefficient — particularly under real-time constraints — and defining useful heuristics is often difficult



- RRT is one such algorithm that, at each step, **randomly select a free space node q'** and connects that to already an existing graph segment **if the connection leads to a collision-free path**
  - If there is a path from P_s to P_g, graph segments are likely to be connected eventually
- **RRT is a probabilistically complete algorithm** and **is not optimal**, but is fast and thus usually useful for practical purposes
  - The search typically needs to be repeated multiple times for a solution to be found

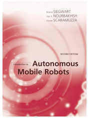
Rapidly Exploring Random Trees (RRTs)

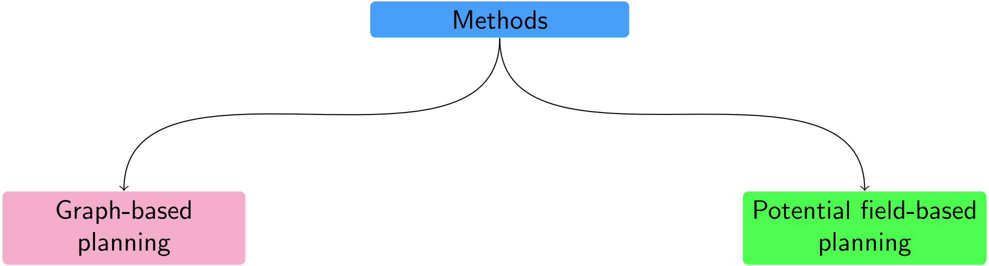
- Many robot planning tasks, particularly in high dimensions, are done using **randomised search**
  - Deterministic search tends to be inefficient — particularly under real-time constraints — and defining useful heuristics is often difficult



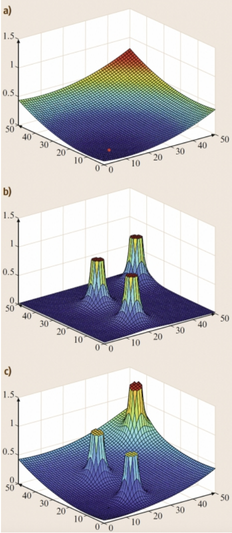
- RRT is one such algorithm that, at each step, **randomly select a free space node q'** and connects that to already an existing graph segment **if the connection leads to a collision-free path**
  - If there is a path from P_s to P_g, graph segments are likely to be connected eventually
- **RRT is a probabilistically complete algorithm** and **is not optimal**, but is fast and thus usually useful for practical purposes
  - The search typically needs to be repeated multiple times for a solution to be found
- As in some deterministic search algorithms, **the search process can be performed bidirectionally** (starting from both P_s and from P_g) to increase the likelihood of finding a path

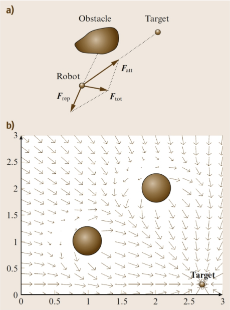
Path Planning Methods





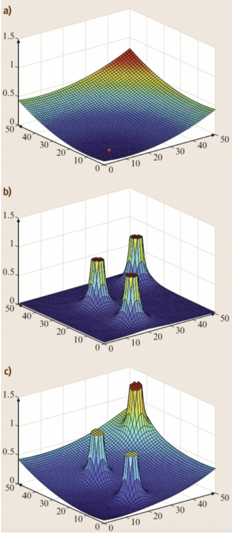
Potential Fields

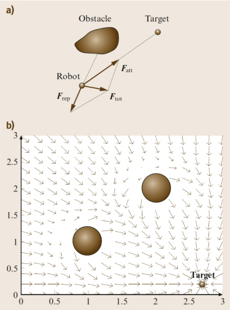




- Potential field planning is an alternative planning strategy based on which **the robot is treated as being under the influence of a potential field $U(q)$**

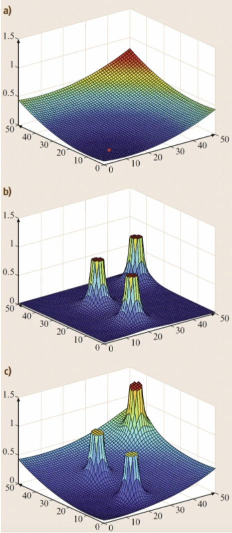
Potential Fields

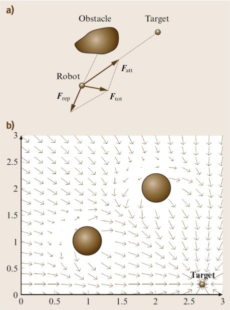




- Potential field planning is an alternative planning strategy based on which **the robot is treated as being under the influence of a potential field $U(\boldsymbol{q})$**
- $U(\boldsymbol{q})$ is created as a combination of attractive and repulsive potentials: $U(\boldsymbol{q}) = U_{attr}(\boldsymbol{q}) + U_{rep}(\boldsymbol{q})$
  - A goal configuration has an attractive potential
  - Obstacles have repulsive potentials

Potential Fields





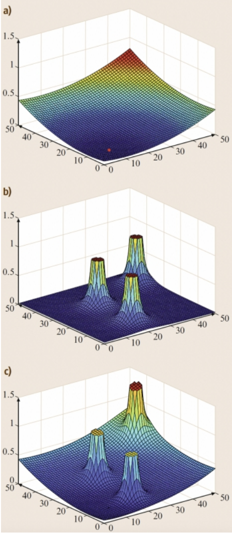
- Potential field planning is an alternative planning strategy based on which **the robot is treated as being under the influence of a potential field $U(\boldsymbol{q})$**

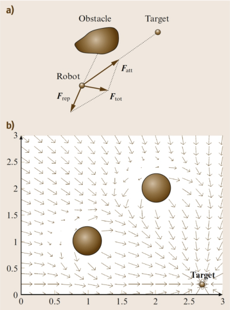
- $U(\boldsymbol{q})$ is created as a combination of attractive and repulsive potentials: $U(\boldsymbol{q}) = U_{attr}(\boldsymbol{q}) + U_{rep}(\boldsymbol{q})$
  - A goal configuration has an attractive potential
  - Obstacles have repulsive potentials

- Recall that a potential is associated with a **conservative force**, which is expressed as

$$\boldsymbol{F}(\boldsymbol{q}) = -\nabla U(\boldsymbol{q})$$

Potential Fields





- Potential field planning is an alternative planning strategy based on which **the robot is treated as being under the influence of a potential field $U(\boldsymbol{q})$**

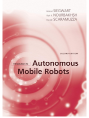
- $U(\boldsymbol{q})$ is created as a combination of attractive and repulsive potentials: $U(\boldsymbol{q}) = U_{attr}(\boldsymbol{q}) + U_{rep}(\boldsymbol{q})$
  - A goal configuration has an attractive potential
  - Obstacles have repulsive potentials

- Recall that a potential is associated with a **conservative force**, which is expressed as

$$\boldsymbol{F}(\boldsymbol{q}) = -\nabla U(\boldsymbol{q})$$

- This means that, at every point $\boldsymbol{q}$, a robot is subject to $F(\boldsymbol{q})$, which dictates the direction in which the robot should move

Attractive Potential



- An attractive potential should **guide a robot towards a given configuration**

- Attractive potentials are typically used only for goal configurations; such a potential can be expressed **as a function of the distance to the goal**

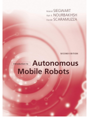
- Let $\|\boldsymbol{q} - \boldsymbol{q}_{goal}\|$ be the Euclidean distance between the current configuration and the goal configuration, and k_a be a positive constant; an example of an attractive potential would then be

$$U_{attr}(\boldsymbol{q}) = \frac{1}{2} k_a \|\boldsymbol{q} - \boldsymbol{q}_{goal}\|^2$$

- The associated force field is then

$$F_{attr}(\boldsymbol{q}) = -\nabla U_{attr}(\boldsymbol{q}) = -k_a(\boldsymbol{q} - \boldsymbol{q}_{goal})$$

Repulsive Potential



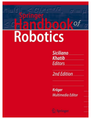
- A repulsive potential should **repel a robot from a given configuration**
- Repulsive potentials are typically used for avoiding obstacles, such that they can be expressed **as a function of the distance to obstacles** — each obstacle would have its own repulsive potential
- **Repulsive fields are typically active only within a given region** — faraway obstacles should not affect the motion of a robot
- Let $\|\boldsymbol{q} - \boldsymbol{q}_o\|$ be the minimum distance between $\boldsymbol{q}$ and any point of an obstacle, ρ_0 be a distance threshold, and k_r a positive constant; an example of a repulsive field is then

$$U_{rep}(\boldsymbol{q}) = \begin{cases} \frac{1}{2} k_r \left(\frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_o\|} - \frac{1}{\rho_0} \right)^2 & \|\boldsymbol{q} - \boldsymbol{q}_o\| \leq \rho_0 \\ 0 & \|\boldsymbol{q} - \boldsymbol{q}_o\| > \rho_0 \end{cases}$$

- The associated force field is given as

$$F_{rep}(\boldsymbol{q}) = \begin{cases} k_r \left(\frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_o\|} - \frac{1}{\rho_0} \right) \frac{1}{\|\boldsymbol{q}-\boldsymbol{q}_o\|^2} \frac{\boldsymbol{q}-\boldsymbol{q}_o}{\|\boldsymbol{q}-\boldsymbol{q}_o\|} & \|\boldsymbol{q} - \boldsymbol{q}_o\| \leq \rho_0 \\ 0 & \|\boldsymbol{q} - \boldsymbol{q}_o\| > \rho_0 \end{cases}$$

Potential Fields and Local Minima



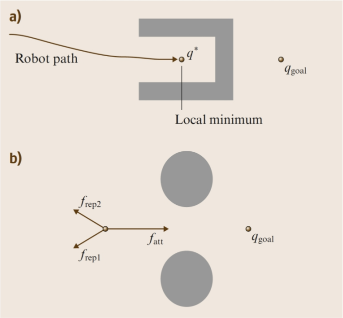
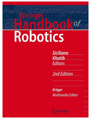


Fig.7.9a,b Two examples of the local minimum problem with potential functions

- Given the interplay between attractive and repulsive potentials, it can happen that the resulting force at a given point adds to $\mathbf{0}$ — **a robot gets stuck at a local minimum** in such a case

Potential Fields and Local Minima



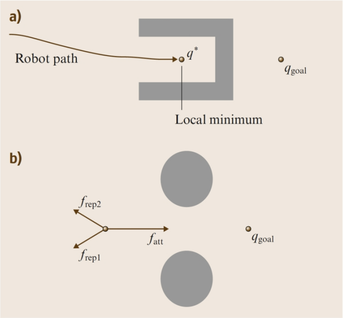
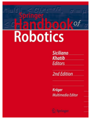


Fig.7.9a,b Two examples of the local minimum problem with potential functions

- Given the interplay between attractive and repulsive potentials, it can happen that the resulting force at a given point adds to $\mathbf{0}$ — **a robot gets stuck at a local minimum** in such a case
- Thus, on their own, **a potential field is not a complete path planner**

Potential Fields and Local Minima



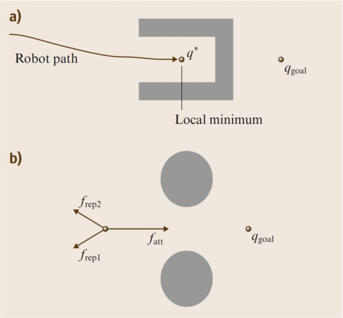
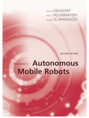
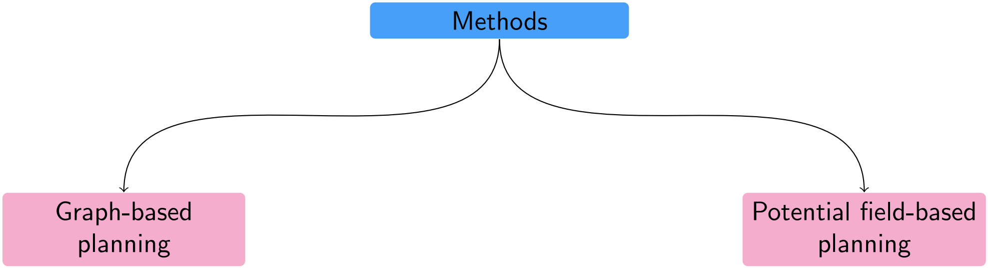


Fig.7.9a,b Two examples of the local minimum problem with potential functions

- Given the interplay between attractive and repulsive potentials, it can happen that the resulting force at a given point adds to $\mathbf{0}$ — **a robot gets stuck at a local minimum** in such a case
- Thus, on their own, **a potential field is not a complete path planner**
- One strategy to escape local minima is to employ **random walks** — this turns a potential field into a randomised planner

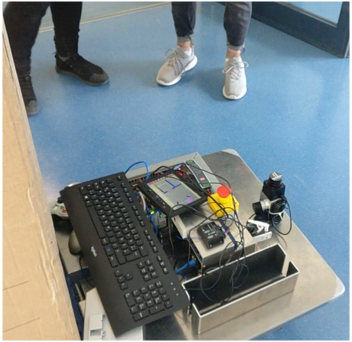
Path Planning Methods





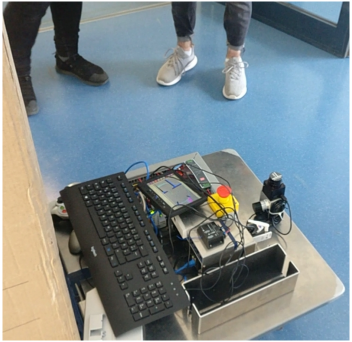
Local Obstacle Avoidance

Local Obstacle Avoidance for Unknown Obstacles



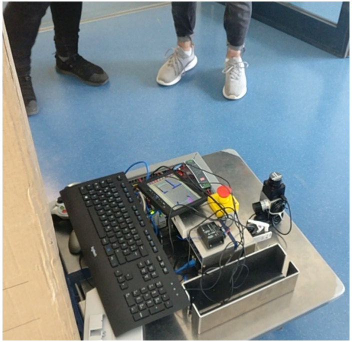
- Path planning can generate collision-free paths for known obstacles in the map, but **a robot should also have an ability to handle unknown and dynamic obstacles**
  - Very few environments are completely static — most are dynamic at least to some extent

Local Obstacle Avoidance for Unknown Obstacles



- Path planning can generate collision-free paths for known obstacles in the map, but **a robot should also have an ability to handle unknown and dynamic obstacles**
  - Very few environments are completely static — most are dynamic at least to some extent
- Local obstacle avoidance needs to **take the current sensor measurements into account** so that appropriate avoidance maneuvers can be performed

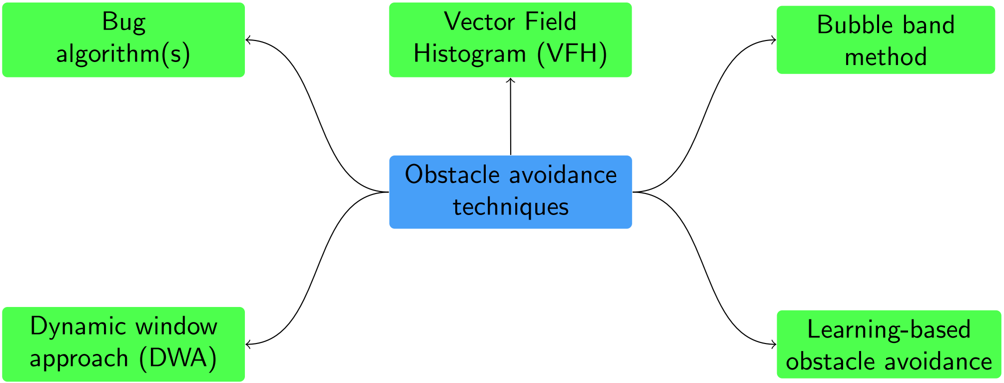
Local Obstacle Avoidance for Unknown Obstacles



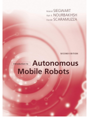
- Path planning can generate collision-free paths for known obstacles in the map, but **a robot should also have an ability to handle unknown and dynamic obstacles**
  - Very few environments are completely static — most are dynamic at least to some extent
- Local obstacle avoidance needs to **take the current sensor measurements into account** so that appropriate avoidance maneuvers can be performed
- Traditional obstacle avoidance strategies are defined for **static obstacles** — dynamic obstacles (such as people) pose a different level of challenge and **are most effective in conjunction with an obstacle motion model**

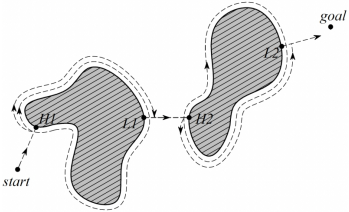
Obstacle Avoidance Techniques

There is a large variety of obstacle avoidance techniques in the literature; we will take a closer look at some of them on the following slides



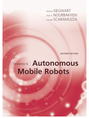
Bug1 Algorithm

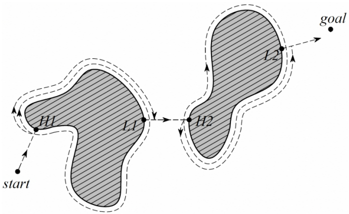




- The Bug1 algorithm is perhaps the simplest obstacle avoidance strategy

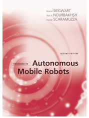
Bug1 Algorithm

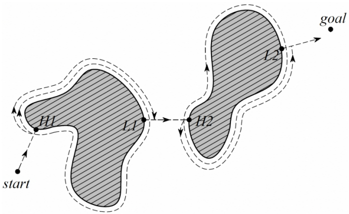




- The Bug1 algorithm is perhaps the simplest obstacle avoidance strategy
- The algorithm performs two steps:

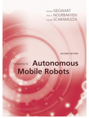
Bug1 Algorithm

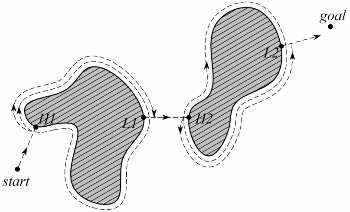




- The Bug1 algorithm is perhaps the simplest obstacle avoidance strategy
- The algorithm performs two steps:
  1. Circle around the obstacle (e.g. using wall following) to find the point that is closest to the goal and come back to the original obstacle approach point

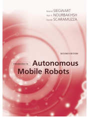
Bug1 Algorithm

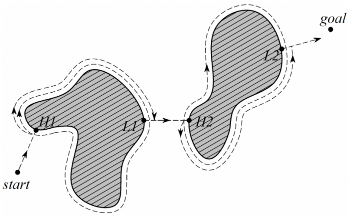




- The Bug1 algorithm is perhaps the simplest obstacle avoidance strategy
- The algorithm performs two steps:
  1. Circle around the obstacle (e.g. using wall following) to find the point that is closest to the goal and come back to the original obstacle approach point
  2. Follow the obstacle contour to the point closest to the goal and then leave the obstacle to move towards the goal

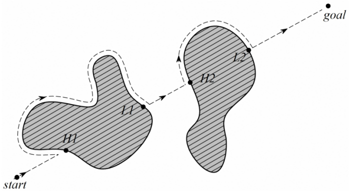
Bug1 Algorithm



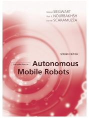


- The Bug1 algorithm is perhaps the simplest obstacle avoidance strategy
- The algorithm performs two steps:
  1. Circle around the obstacle (e.g. using wall following) to find the point that is closest to the goal and come back to the original obstacle approach point
  2. Follow the obstacle contour to the point closest to the goal and then leave the obstacle to move towards the goal
- Bug1 is a **naive and inefficient obstacle avoidance strategy**, as the full obstacle contour needs to be traversed so that a departure point is identified

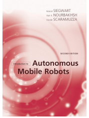
Bug2

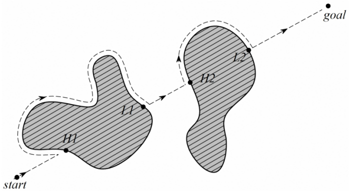


- Bug2 constitutes a more efficient version of Bug1



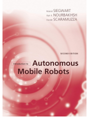
Bug2

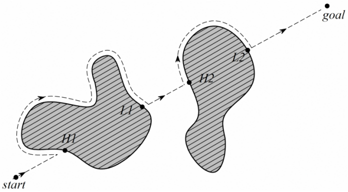




- Bug2 constitutes a more efficient version of Bug1
- The idea behind Bug2 is to **follow the obstacle's contour until reaching a point from which there is a direct path to the goal**; at this point, the robot leaves the obstacle and starts moving towards the goal

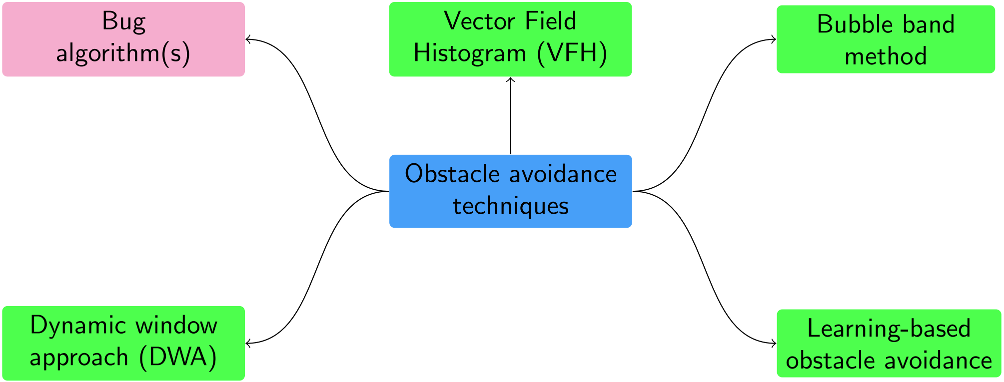
Bug2



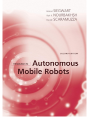


- Bug2 constitutes a more efficient version of Bug1
- The idea behind Bug2 is to **follow the obstacle's contour until reaching a point from which there is a direct path to the goal**; at this point, the robot leaves the obstacle and starts moving towards the goal
- Some non-convex obstacle shapes may lead to a suboptimal or oscillatory behaviour of the bug algorithms

Obstacle Avoidance Techniques

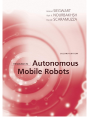


Vector Field Histogram (VFH)



- A vector field histogram is an obstacle avoidance method that uses a local map based on recent sensor measurements

Vector Field Histogram (VFH)



- A vector field histogram is an obstacle avoidance method that uses a local map based on recent sensor measurements

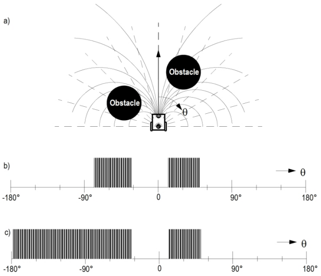
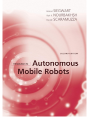


Figure 6.17
Example of blocked directions and resulting polar histograms [54]. (a) Robot and blocking obstacles. (b) Polar histogram. (b) Masked polar histogram.

- The method **creates a discrete histogram that encodes the probability that there is an obstacle at a given direction from the robot**

Vector Field Histogram (VFH)



- A vector field histogram is an obstacle avoidance method that uses a local map based on recent sensor measurements

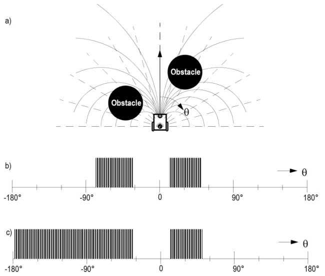


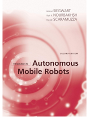
Figure 6.17
Example of blocked directions and resulting polar histograms [54]. (a) Robot and blocking obstacles. (b) Polar histogram. (b) Masked polar histogram.

- The method **creates a discrete histogram that encodes the probability that there is an obstacle at a given direction from the robot**
- Given the histogram, **candidate passages that would fit the robot are found**, and a direction of motion is **identified based on a cost function** of the form:

$$J = w_1 h + w_2 \gamma + w_3 \Delta h$$

- Here $w_{1,2,3}$ are positive constants, h is the orientation towards the goal, γ is the change in wheel orientation that would be necessary to move in the candidate orientation, and Δh is the necessary orientation change to achieve h

Vector Field Histogram (VFH)



- A vector field histogram is an obstacle avoidance method that uses a local map based on recent sensor measurements

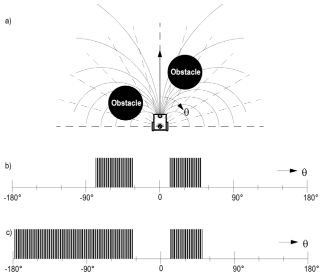


Figure 6.17
Example of blocked directions and resulting polar histograms [54]. (a) Robot and blocking obstacles. (b) Polar histogram. (b) Masked polar histogram.

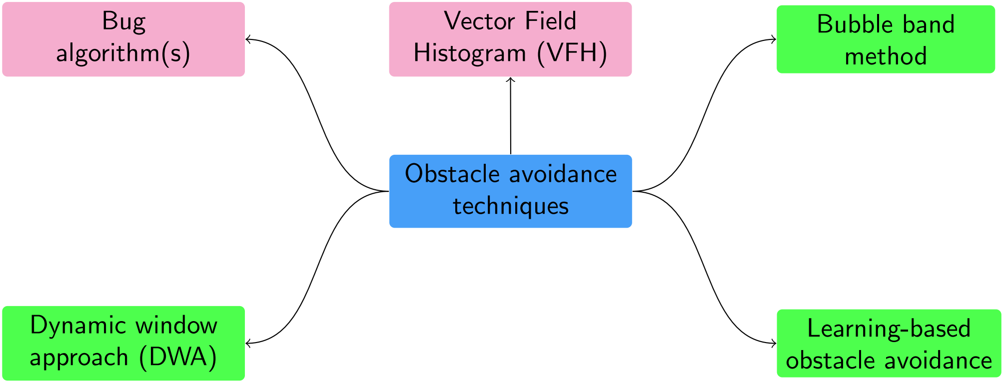
- The method **creates a discrete histogram that encodes the probability that there is an obstacle at a given direction from the robot**
- Given the histogram, **candidate passages that would fit the robot are found**, and a direction of motion is **identified based on a cost function** of the form:

$$J = w_1 h + w_2 \gamma + w_3 \Delta h$$

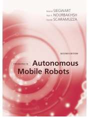
- Here $w_{1,2,3}$ are positive constants, h is the orientation towards the goal, γ is the change in wheel orientation that would be necessary to move in the candidate orientation, and Δh is the necessary orientation change to achieve h

- An extended VFH method assumes **motion along straight lines and arcs**, and creates a **masked histogram** that prevents motion directions that would pass through the obstacles

Obstacle Avoidance Techniques



Bubble Band Method



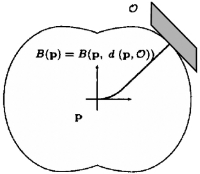


Figure 6.18
Shape of the bubbles around the vehicle. Courtesy of Raja Chatila [165].

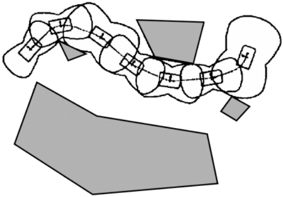
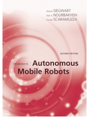


Figure 6.19
A typical bubble band. Courtesy of Raja Chatila [165].

- The bubble band method **models the robot as a bubble**, where a bubble is **the maximum reachable space without collisions around a configuration q**

Bubble Band Method



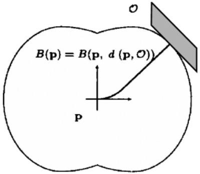


Figure 6.18
Shape of the bubbles around the vehicle. Courtesy of Raja Chatila [165].

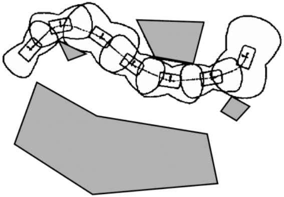
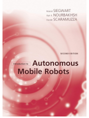


Figure 6.19
A typical bubble band. Courtesy of Raja Chatila [165].

- The bubble band method **models the robot as a bubble**, where a bubble is **the maximum reachable space without collisions around a configuration q**
- A bubble band can be used to pre-plan a full trajectory, which consists of **a sequence of overlapping bubbles**

Bubble Band Method



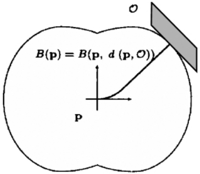


Figure 6.18
Shape of the bubbles around the vehicle. Courtesy of Raja Chatila [165].

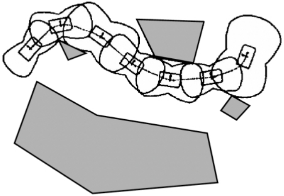
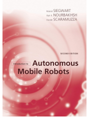


Figure 6.19
A typical bubble band. Courtesy of Raja Chatila [165].

- The bubble band method **models the robot as a bubble**, where a bubble is **the maximum reachable space without collisions around a configuration q**
- A bubble band can be used to pre-plan a full trajectory, which consists of **a sequence of overlapping bubbles**
- During online execution:
  - **internal forces** are used for online energy minimisation (so that a smooth trajectory is achieved)
  - **obstacles apply external repulsive forces** to the bubbles

Bubble Band Method



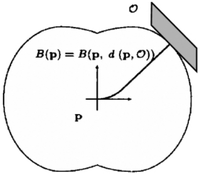


Figure 6.18
Shape of the bubbles around the vehicle. Courtesy of Raja Chatila [165].

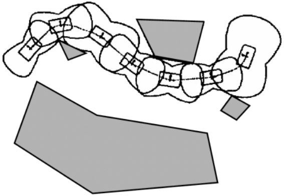
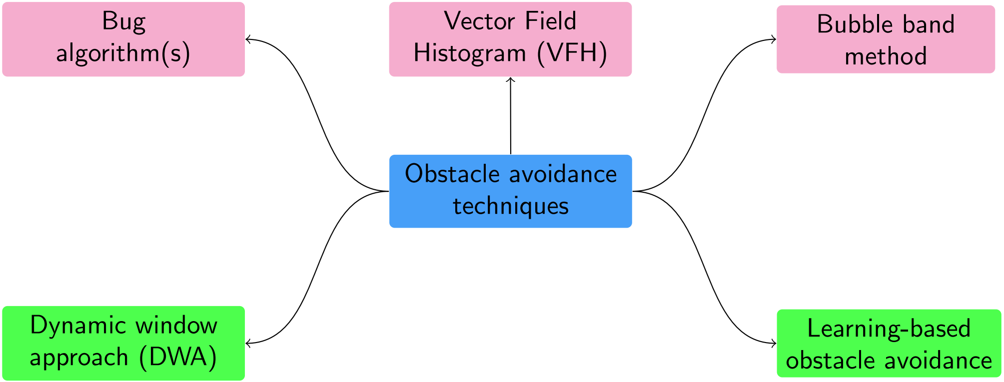


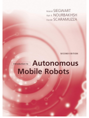
Figure 6.19
A typical bubble band. Courtesy of Raja Chatila [165].

- The bubble band method **models the robot as a bubble**, where a bubble is **the maximum reachable space without collisions around a configuration q**
- A bubble band can be used to pre-plan a full trajectory, which consists of **a sequence of overlapping bubbles**
- During online execution:
  - **internal forces** are used for online energy minimisation (so that a smooth trajectory is achieved)
  - **obstacles apply external repulsive forces** to the bubbles
- The bubble band method is thus **a path and motion planning method**

Obstacle Avoidance Techniques



Dynamic Window Approach (DWA)



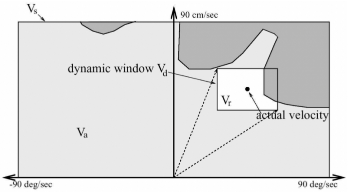
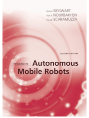


Figure 6.21
The dynamic window approach (courtesy of Dieter Fox [130]).
The rectangular window shows the possible speeds (v, ω) and the overlap with obstacles in configuration space.

- The dynamic window approach enables obstacle avoidance **by considering kinematic constraints**

Dynamic Window Approach (DWA)



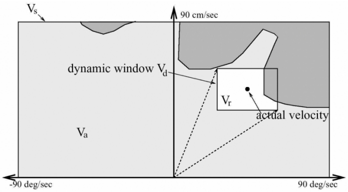
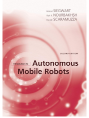


Figure 6.21
The dynamic window approach (courtesy of Dieter Fox [130]).
The rectangular window shows the possible speeds (v, ω) and the overlap with obstacles in configuration space.

- The dynamic window approach enables obstacle avoidance **by considering kinematic constraints**
- There are multiple variations of the technique, but they can roughly be divided into:
  - **Local DWA**, which only considers local obstacle information
  - **Global DWA**, which also includes global environment information in its planning process

Dynamic Window Approach (DWA)



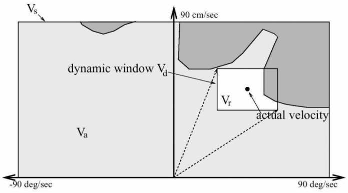
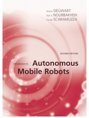


Figure 6.21
The dynamic window approach (courtesy of Dieter Fox [130]). The rectangular window shows the possible speeds (v, ω) and the overlap with obstacles in configuration space.

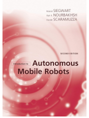
- The dynamic window approach enables obstacle avoidance **by considering kinematic constraints**
- There are multiple variations of the technique, but they can roughly be divided into:
  - **Local DWA**, which only considers local obstacle information
  - **Global DWA**, which also includes global environment information in its planning process
- DWA in not just a method for path planning, but also for **motion planning**
  - Prediction of the effects of the robot's motion — based on a motion model — are thus done by the algorithm

Local Dynamic Window Approach



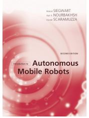
- The local DWA assumes circular motion with linear velocity v and angular velocity ω, such that it tries to **find instantaneous velocities that would bring the robot closer to the goal without causing an obstacle collision**

Local Dynamic Window Approach



- The local DWA assumes circular motion with linear velocity v and angular velocity ω, such that it tries to **find instantaneous velocities that would bring the robot closer to the goal without causing an obstacle collision**
- The approach performs two steps at every iteration (i.e. at every step of the control algorithm):
  1. Finding a dynamic window of **feasible velocities** that a robot can reach within the next control step
  2. Reducing the dynamic window by only considering **admissible velocities**, namely those that guarantee that no obstacle collision will occur

Local Dynamic Window Approach

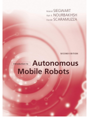


- The local DWA assumes circular motion with linear velocity v and angular velocity ω, such that it tries to **find instantaneous velocities that would bring the robot closer to the goal without causing an obstacle collision**
- The approach performs two steps at every iteration (i.e. at every step of the control algorithm):
  1. Finding a dynamic window of **feasible velocities** that a robot can reach within the next control step
  2. Reducing the dynamic window by only considering **admissible velocities**, namely those that guarantee that no obstacle collision will occur
- From the admissible set, v and ω are chosen so that they **keep the robot as away from obstacles**, **are as aligned with the goal**, and **are as fast** as possible
  - This is achieved using an objective function of the form

$$J(v, \omega) = w_1 h(v, \omega) + w_2 s(v, \omega) + w_3 d(v, \omega)$$

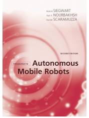
where $w_{1,2,3}$ are positive constants, h is the heading, s the speed, and d the closest distance to an obstacle

Global Dynamic Window Approach



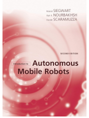
- The global DWA is an extension of the local DWA approach that additionally considers global environment information

Global Dynamic Window Approach



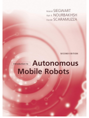
- The global DWA is an extension of the local DWA approach that additionally considers global environment information
- This is particularly done by **maintaining a local occupancy grid** and **recalculating the distance to the goal using the wavefront algorithm**

Global Dynamic Window Approach



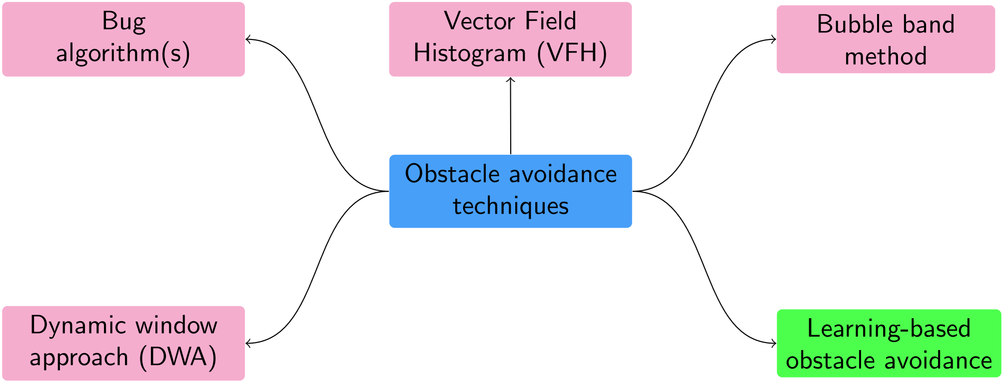
- The global DWA is an extension of the local DWA approach that additionally considers global environment information
- This is particularly done by **maintaining a local occupancy grid** and **recalculating the distance to the goal using the wavefront algorithm**
- **The size of the region covered by the local occupancy grid is dynamically changed** so that the goal can always be found from the robot's current position

Global Dynamic Window Approach

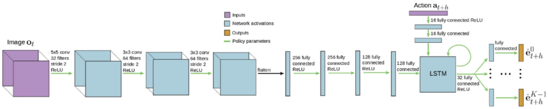


- The global DWA is an extension of the local DWA approach that additionally considers global environment information
- This is particularly done by **maintaining a local occupancy grid** and **recalculating the distance to the goal using the wavefront algorithm**
- **The size of the region covered by the local occupancy grid is dynamically changed** so that the goal can always be found from the robot's current position
- The global DWA **reverts to the local DWA** when the robot is surrounded by obstacles and a path to the goal cannot be found using the wavefront algorithm

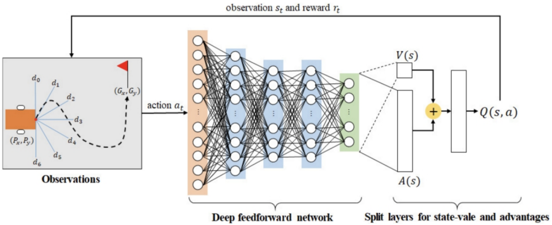
Obstacle Avoidance Techniques



Learning-Based Obstacle Avoidance



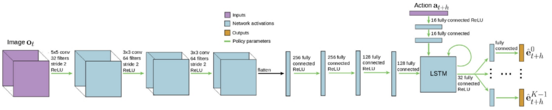
G. Kahn, P. Abbeel and S. Levine, "BADGR: An Autonomous Self-Supervised Learning-Based Navigation System," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, Apr. 2021.



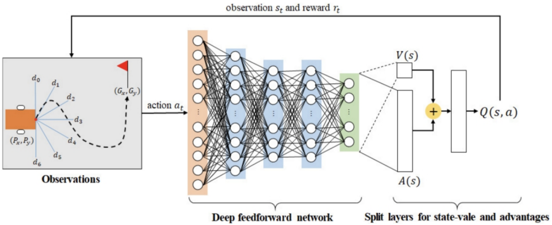
S. -H. Han et al., "Sensor-Based Mobile Robot Navigation via Deep Reinforcement Learning," in *Proc. IEEE Int. Conf. Big Data and Smart Computing (BigComp)*, 2018, pp. 147–154.

- The previously discussed obstacle avoidance strategies are **model-based** — a model of the robot (and sometimes of obstacles) is used for path and motion planning

Learning-Based Obstacle Avoidance



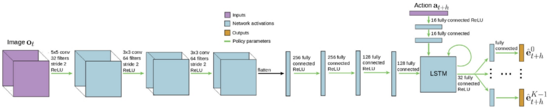
G. Kahn, P. Abbeel and S. Levine, "BADGR: An Autonomous Self-Supervised Learning-Based Navigation System," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, Apr. 2021.



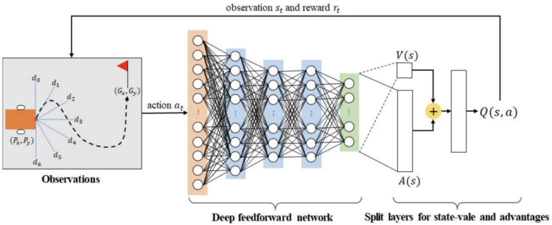
S. -H. Han et al., "Sensor-Based Mobile Robot Navigation via Deep Reinforcement Learning," in *Proc. IEEE Int. Conf. Big Data and Smart Computing (BigComp)*, 2018, pp. 147–154.

- The previously discussed obstacle avoidance strategies are **model-based** — a model of the robot (and sometimes of obstacles) is used for path and motion planning
- In recent years, there have been attempts to use learning algorithm that **acquire local navigation behaviours that map sensor measurements to motions** — often using **learned neural network-based policies**

Learning-Based Obstacle Avoidance



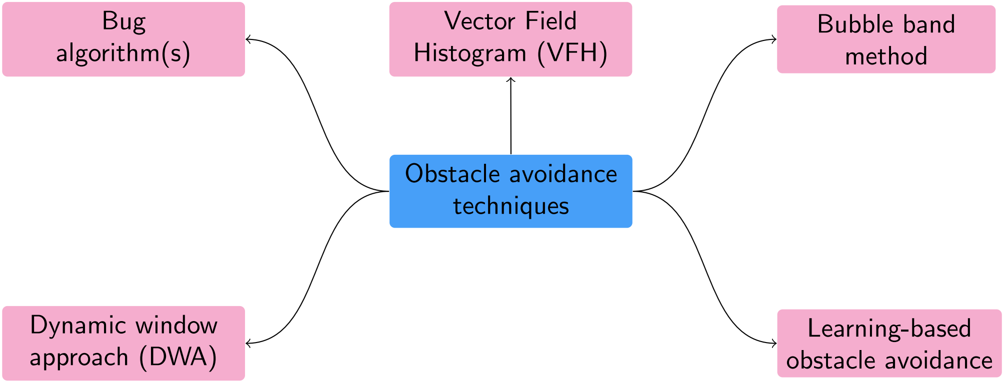
G. Kahn, P. Abbeel and S. Levine, "BADGR: An Autonomous Self-Supervised Learning-Based Navigation System," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, Apr. 2021.



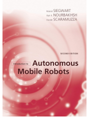
S. -H. Han et al., "Sensor-Based Mobile Robot Navigation via Deep Reinforcement Learning," in *Proc. IEEE Int. Conf. Big Data and Smart Computing (BigComp)*, 2018, pp. 147–154.

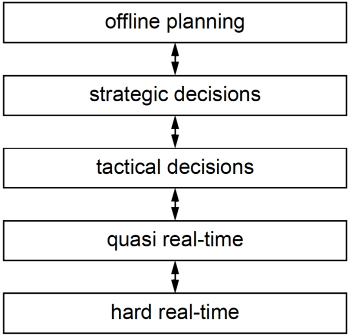
- The previously discussed obstacle avoidance strategies are **model-based** — a model of the robot (and sometimes of obstacles) is used for path and motion planning
- In recent years, there have been attempts to use learning algorithm that **acquire local navigation behaviours that map sensor measurements to motions** — often using **learned neural network-based policies**
- The development and exploration of such methods is, however, still an ongoing process — **model-based techniques still dominate navigation applications**

Obstacle Avoidance Techniques



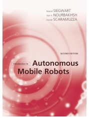
Temporal Considerations

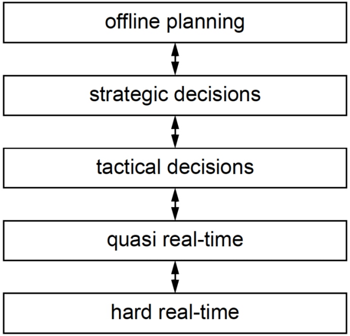




- When integrating path and motion planning algorithms on robot platforms, it is important to **consider any timing constraints** that need to be fulfilled for **successful and safe operation**

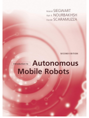
Temporal Considerations

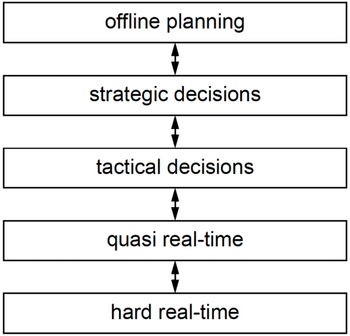




- When integrating path and motion planning algorithms on robot platforms, it is important to **consider any timing constraints** that need to be fulfilled for **successful and safe operation**
- In this respect, it is important to distinguish between:
  - operations with **real-time demands**, which need to have a very low latency (e.g. real-time obstacle avoidance), and
  - operations where some latency (due to sensor processing or planning) can be tolerated (e.g. planning a path to a goal)

Temporal Considerations





- When integrating path and motion planning algorithms on robot platforms, it is important to **consider any timing constraints** that need to be fulfilled for **successful and safe operation**
- In this respect, it is important to distinguish between:
  - operations with **real-time demands**, which need to have a very low latency (e.g. real-time obstacle avoidance), and
  - operations where some latency (due to sensor processing or planning) can be tolerated (e.g. planning a path to a goal)
- Temporal constraints are taken into account within a **navigation architecture**

Summary

- Path planning is the problem of finding a collision-free path that brings a robot from its initial location to a goal
- There are various (offline) path planning algorithms, which can be observed as belonging to two major categories: graph-based search and potential field planning
- Path planning algorithms find a path in a known map, but online obstacle avoidance is also required for dealing with environmental changes; there are many obstacle avoidance methods in the literature, most of which perform both path and motion planning (e.g. DWA)
- Machine learning-based approaches aim to replace the dependency on (simple) robot models by acquiring navigation behaviours from data
- Navigation architectures need to take into account timing constraints on the operation of a robot, particularly for functionalities that have hard real-time constraints