Motion Planning and Control for Domestic Service Robots

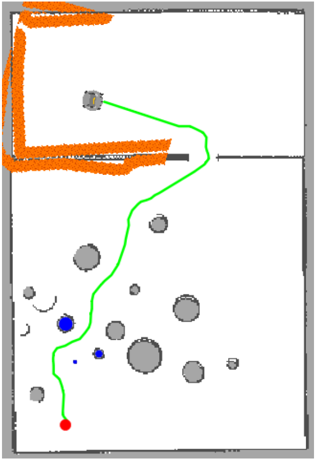
J.J.M. Lunenburg

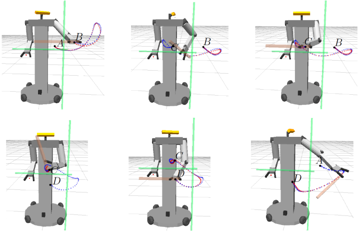
Technische Universiteit
Eindhoven
University of Technology

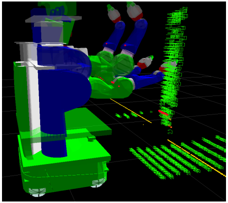
Where innovation starts

May 8, 2014

Motion Planning for Domestic Service Robots

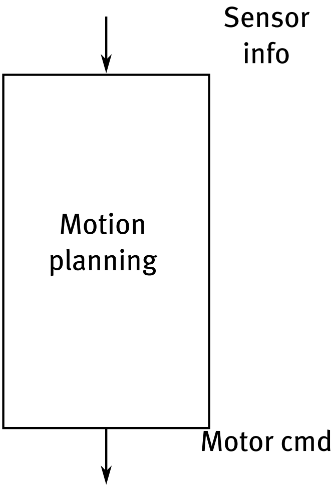






Moving PICO

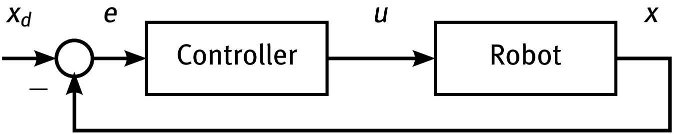
- Sensors:
  - Laser scan
  - Odometry
- Motor cmd
  - Velocity in x, y and ϕ direction
- Move towards a desired pose
- Don't crash into the wall!



Technische Universiteit
Eindhoven
University of Technology

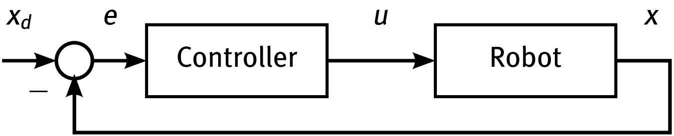
Moving to a desired pose

- Feedback control!
- Carrot planner
  - P(D)-controller
- Dynamic Window Approach:
  - Search for a translational and rotational velocity
  - Optimization over a finite horizon
  - MPC-controller



Moving to a desired pose

- Feedback control!
- Carrot planner
  - P(D)-controller
- Dynamic Window Approach:
  - Search for a translational and rotational velocity
  - Optimization over a finite horizon
  - MPC-controller
- What about obstacles?

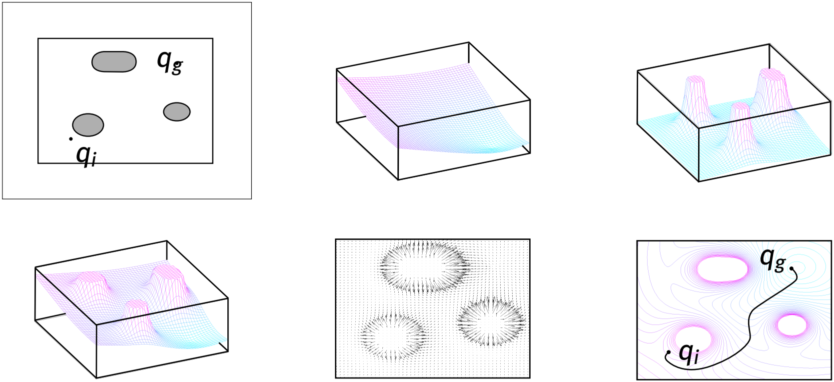


Don’t crash into obstacles

- Line collision checks
- Forward simulation: rejects inputs

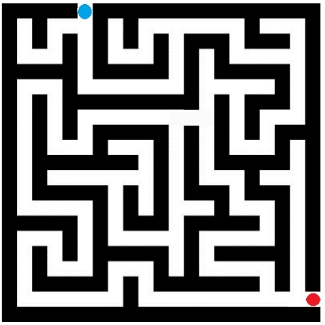
Don’t crash into obstacles

- Line collision checks
- Forward simulation: rejects inputs
- Alternative: potential fields
    - Goal and obstacles form attractive and repulsive forces



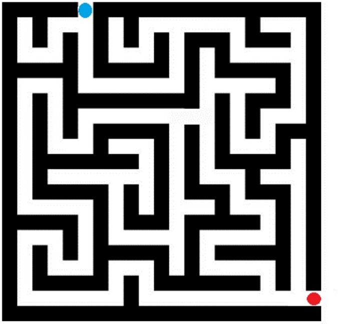
Limitations

- Local methods
  - Vicinity of the robot
- Completeness: ‘getting stuck’
- Optimality: ‘the shortest path’



Limitations

- Local methods
  - Vicinity of the robot
- Completeness: ‘getting stuck’
- Optimality: ‘the shortest path’
- Global connectivity information required

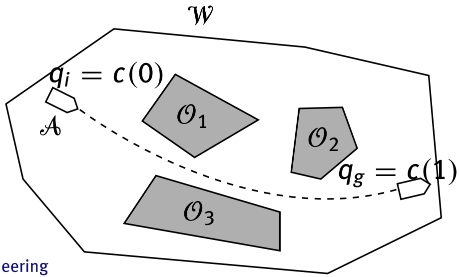


The Basic Motion Planning Problem

With:

- Pose (position and orientation)
- Single rigid body $\mathcal{A}$
- n-dimensional Euclidean space $\mathcal{W} = \mathbb{R}^n$
- Static, rigid obstacles $\mathcal{O}_i$ in $\mathcal{W}$

Given an initial pose and a goal pose of $\mathcal{A}$ in $\mathcal{W}$, find a path c in the form of a continuous sequence of poses of $\mathcal{A}$ that do not collide or contact with $\mathcal{O}_i$, that will allow $\mathcal{A}$ to move from its starting pose to its goal pose and report failure if such a path does not exist.



Specifications and properties

Six specifications and properties

- Completeness: finding a path if one exists
- Optimality: finding the optimal path
- Computational complexity
- Robustness against a dynamic environment
- Robustness against uncertainty
- Kinematic and dynamic constraints

Specifications and properties

Six specifications and properties

- Completeness: finding a path if one exists
- Optimality: finding the optimal path
- Computational complexity
- Robustness against a dynamic environment
- Robustness against uncertainty
- Kinematic and dynamic constraints

So how do we approach this problem?

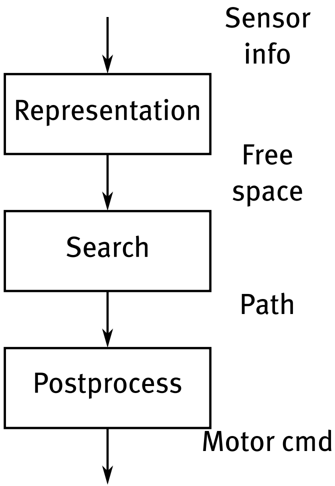
Specifications and properties

Six specifications and properties

- Completeness: finding a path if one exists
- Optimality: finding the optimal path
- Computational complexity
- Robustness against a dynamic environment
- Robustness against uncertainty
- Kinematic and dynamic constraints

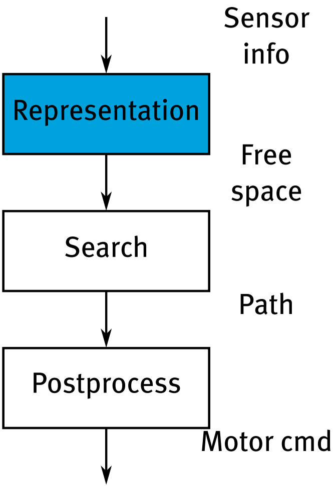
So how do we approach this problem?

Representation and searching!



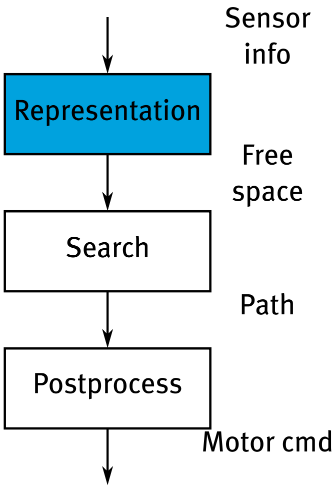
Environment representations

- The configuration space
  - Simplifies the problem: search for a solution for a single point
  - Generic
  - Computationally efficient



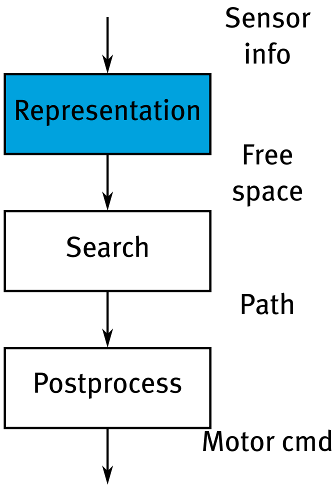
Environment representations

- The configuration space
  - Simplifies the problem: search for a solution for a single point
  - Generic
  - Computationally efficient
- Representation methods:
  - Exact
    - Roadmaps
    - Exact cell decomposition
  - Approximate
    - Approximate cell decompositions
    - Sampling-based methods
    - Potential fields

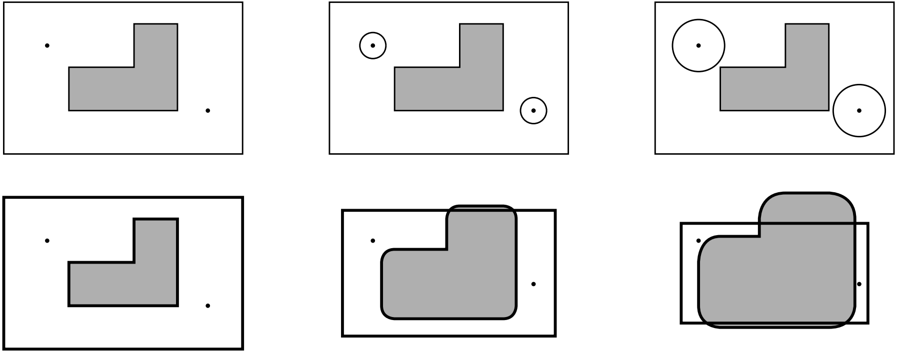


Environment representations

- The configuration space
  - Simplifies the problem: search for a solution for a single point
  - Generic
  - Computationally efficient
- Representation methods:
  - Exact
    - Roadmaps
    - Exact cell decomposition
  - Approximate
    - Approximate cell decompositions
    - Sampling-based methods
    - Potential fields
- Common assumption: localization

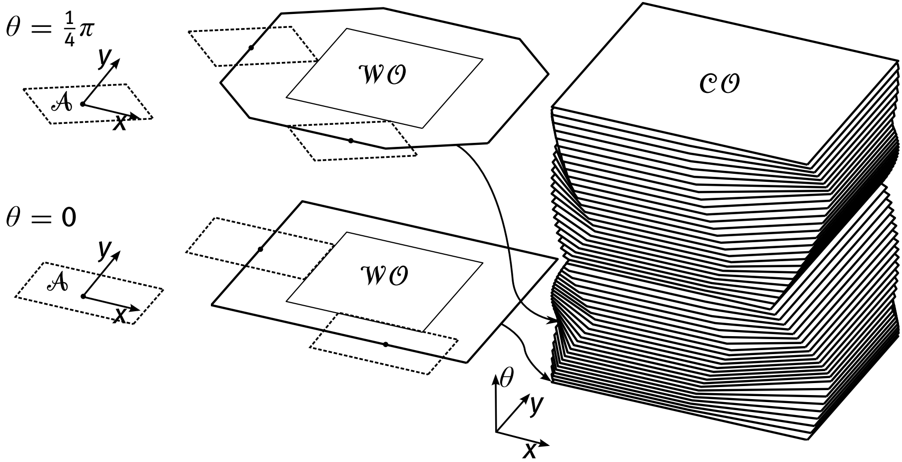


Constructing the configuration space



(a) (b) (c)

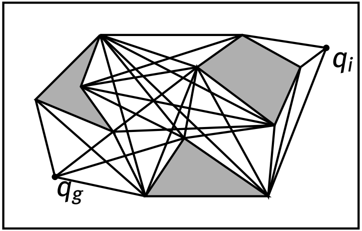
Non-circular robot footprints



Exact: Roadmaps

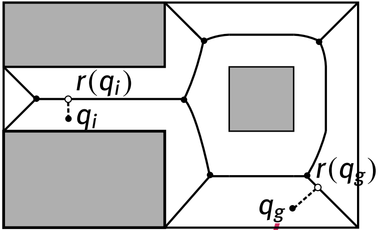
Visibility graph

- Two nodes are connected if the straight line between them is collision-free
- $\dim(\mathcal{C}) \leq 2$
- Optimal w.r.t. distance traveled



Deformation retracts

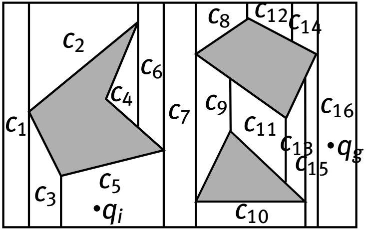
- 'Shrink' a space into a subspace
- (Generalized) Voronoi diagram
- Optimal w.r.t. distance to obstacles



Exact and approximate: Cell decompositions

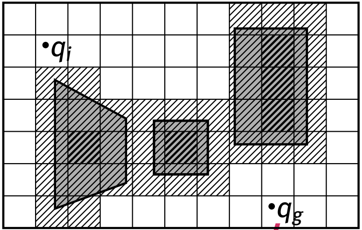
Exact decomposition

- Trapezoidal decomposition
- Sweep line algorithm
- Non-optimal



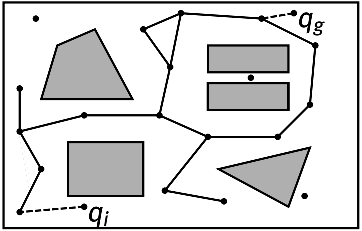
Approximate decomposition

- Obstacle boundaries do not coincide with cell boundaries
- Free cells, mixed cells and occupied cells
- Resolution complete



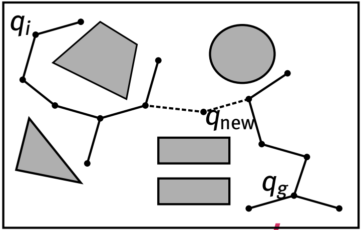
Probabilistic roadmap

- Learning phase: sample configuration q_{rand} and check for collisions
- Query phase: connect q_i and q_g to roadmap $\mathcal{R}$
- Probabilistically complete



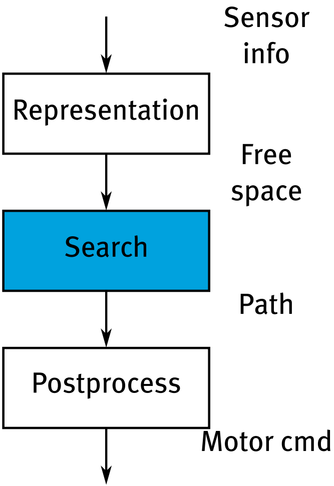
Single-query planner

- Explore relevant subset of $\mathcal{C}_{\text{free}}$
- (Bidirectional) Rapidly-exploring Random Tree
- No search algorithm required
- Probabilistically complete, non-optimal

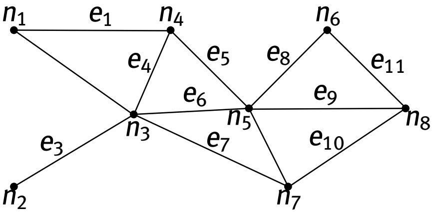


Search algorithms

- Graphs and costmaps
- Graph search algorithms:
  - Uninformed
  - Informed
  - Local

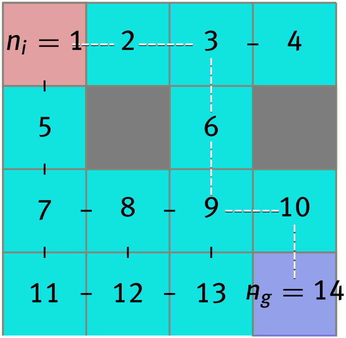


Graphs and costmaps



- Nodes (vertices) and edges
- Including weights: costmap
- Parent: node with subsequent nodes (children)
- Branch: series of nodes connecting the root to a leaf
- Frontier: set of all leaf nodes available for expansion
- Closed list (explored set): nodes that have been visited
- Expansion is determined by function $f(n)$

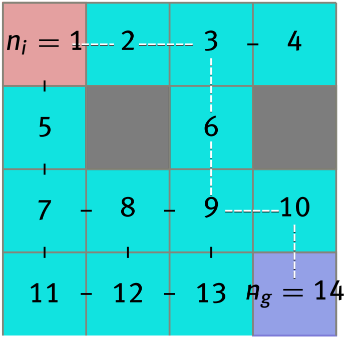
Uninformed search: Breadth-first

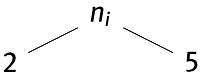


n_i

- $f(n) = g(n)$, with $g(n)$ a FIFO queue
- All nodes at a certain depth are expanded before going to the next level
- Complete (if 'branching' factor is finite)
- Optimal: only if all edges have equal costs

Uninformed search: Breadth-first

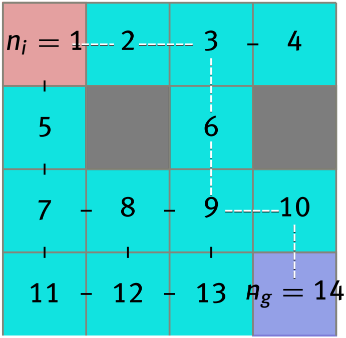


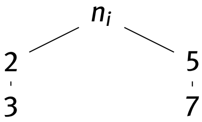


- $f(n) = g(n)$, with $g(n)$ a FIFO queue
- All nodes at a certain depth are expanded before going to the next level
- Complete (if 'branching' factor is finite)
- Optimal: only if all edges have equal costs

Technische Universiteit Eindhoven University of Technology

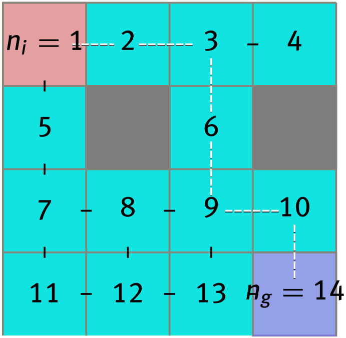
Uninformed search: Breadth-first

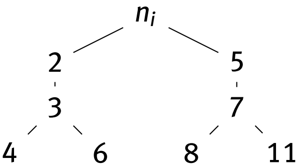




- $f(n) = g(n)$, with $g(n)$ a FIFO queue
- All nodes at a certain depth are expanded before going to the next level
- Complete (if ‘branching’ factor is finite)
- Optimal: only if all edges have equal costs

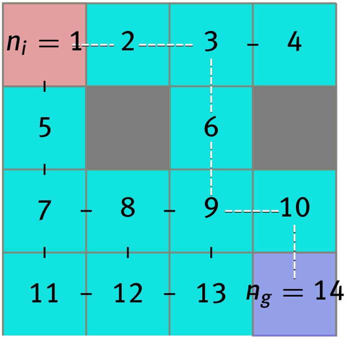
Uninformed search: Breadth-first

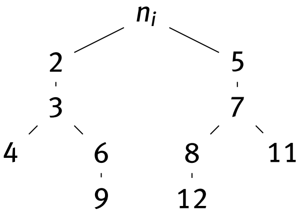




- $f(n) = g(n)$, with $g(n)$ a FIFO queue
- All nodes at a certain depth are expanded before going to the next level
- Complete (if ‘branching’ factor is finite)
- Optimal: only if all edges have equal costs

Uninformed search: Breadth-first

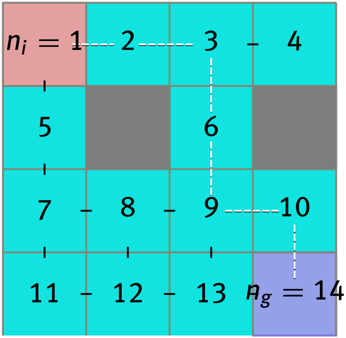


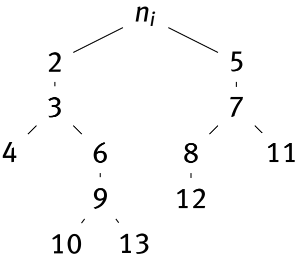


- $f(n) = g(n)$, with $g(n)$ a FIFO queue
- All nodes at a certain depth are expanded before going to the next level
- Complete (if ‘branching’ factor is finite)
- Optimal: only if all edges have equal costs

Technische Universiteit Eindhoven University of Technology

Uninformed search: Breadth-first

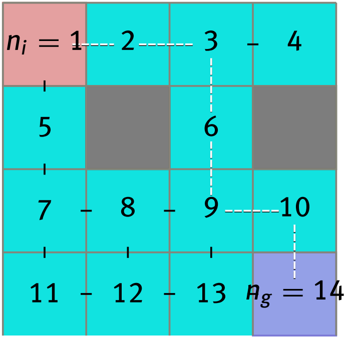


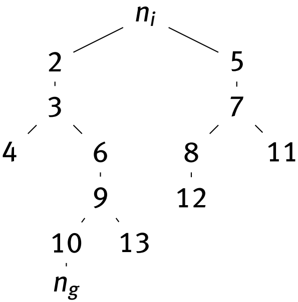


- $f(n) = g(n)$, with $g(n)$ a FIFO queue
- All nodes at a certain depth are expanded before going to the next level
- Complete (if 'branching' factor is finite)
- Optimal: only if all edges have equal costs

TU/e Technische Universiteit Eindhoven University of Technology

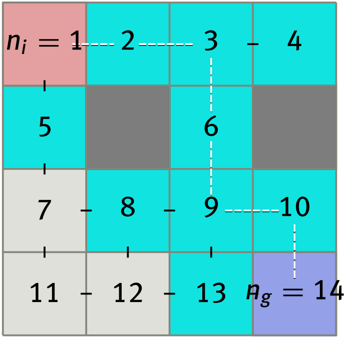
Uninformed search: Breadth-first





- $f(n) = g(n)$, with $g(n)$ a FIFO queue
- All nodes at a certain depth are expanded before going to the next level
- Complete (if ‘branching’ factor is finite)
- Optimal: only if all edges have equal costs

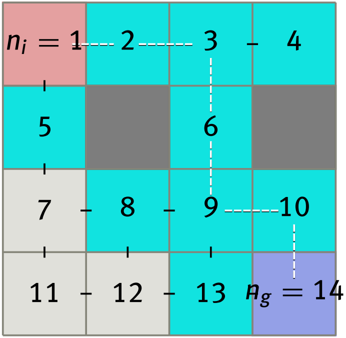
Uninformed search: Depth-first

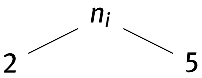


n_i

- $f(n) = g(n)$, with $g(n)$ a LIFO queue
- The most recent expanded node is put the beginning of the stack
- Completeness: if search space is finite
- Not optimal
  - Example: put goal at node 5

Uninformed search: Depth-first

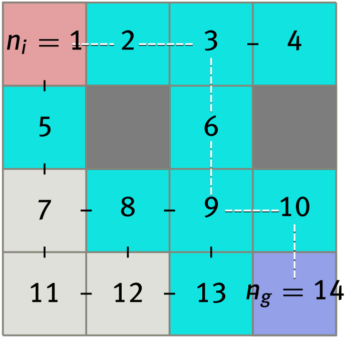


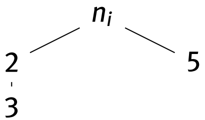


- $f(n) = g(n)$, with $g(n)$ a LIFO queue
- The most recent expanded node is put the beginning of the stack
- Completeness: if search space is finite
- Not optimal
  - Example: put goal at node 5

Technische Universiteit Eindhoven University of Technology

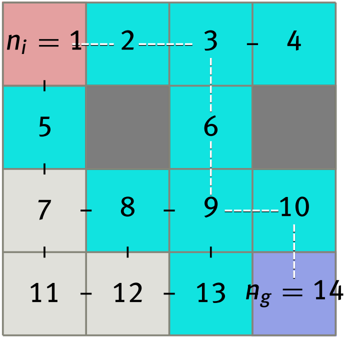
Uninformed search: Depth-first

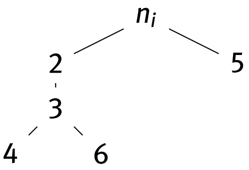




- $f(n) = g(n)$, with $g(n)$ a LIFO queue
- The most recent expanded node is put the beginning of the stack
- Completeness: if search space is finite
- Not optimal
  - Example: put goal at node 5

Uninformed search: Depth-first

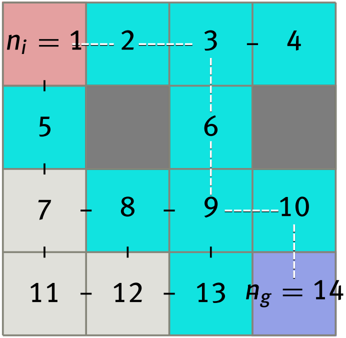


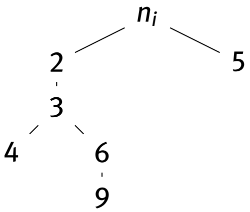


- $f(n) = g(n)$, with $g(n)$ a LIFO queue
- The most recent expanded node is put the beginning of the stack
- Completeness: if search space is finite
- Not optimal
  - Example: put goal at node 5

Technische Universiteit Eindhoven University of Technology

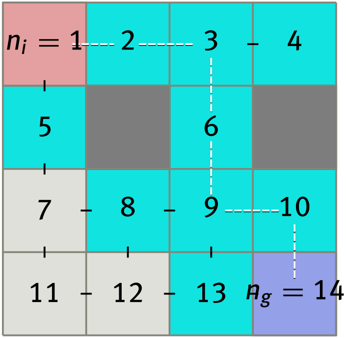
Uninformed search: Depth-first

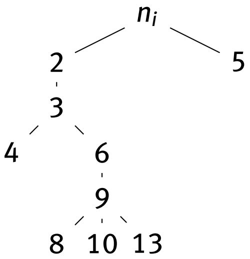




- $f(n) = g(n)$, with $g(n)$ a LIFO queue
- The most recent expanded node is put the beginning of the stack
- Completeness: if search space is finite
- Not optimal
  - Example: put goal at node 5

Uninformed search: Depth-first

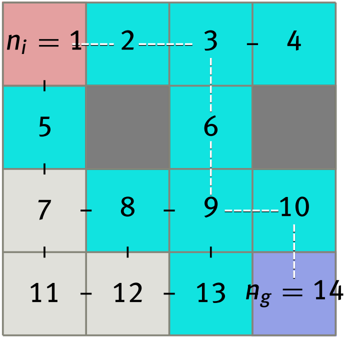


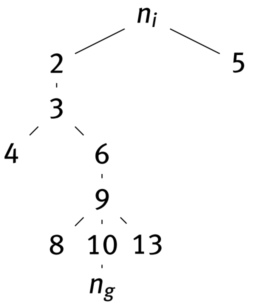


- $f(n) = g(n)$, with $g(n)$ a LIFO queue
- The most recent expanded node is put the beginning of the stack
- Completeness: if search space is finite
- Not optimal
  - Example: put goal at node 5

Technische Universiteit Eindhoven University of Technology

Uninformed search: Depth-first

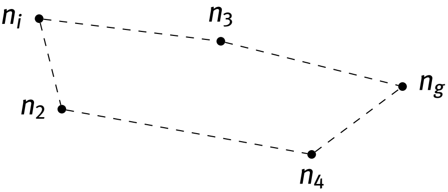




- $f(n) = g(n)$, with $g(n)$ a LIFO queue
- The most recent expanded node is put the beginning of the stack
- Completeness: if search space is finite
- Not optimal
  - Example: put goal at node 5

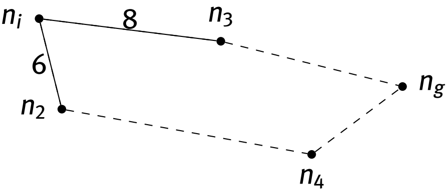
Uninformed search: Dijkstra's Algorithm

- $f(n) = g(n)$, with $g(n)$ a priority queue
- The node with the lowest cost is expanded
- Completeness: if search space is finite
- Optimal



Uninformed search: Dijkstra's Algorithm

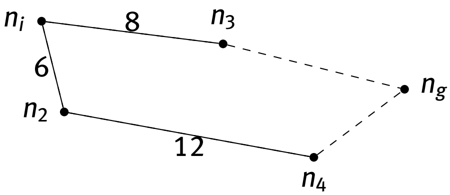
- $f(n) = g(n)$, with $g(n)$ a priority queue
- The node with the lowest cost is expanded
- Completeness: if search space is finite
- Optimal



- $6 < 8 \rightarrow$ expand n_2

Uninformed search: Dijkstra's Algorithm

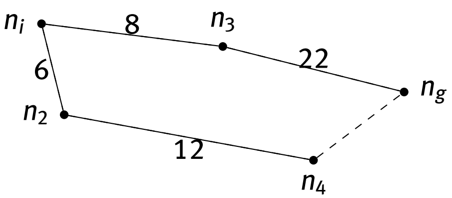
- $f(n) = g(n)$, with $g(n)$ a priority queue
- The node with the lowest cost is expanded
- Completeness: if search space is finite
- Optimal



- $6 < 8 \rightarrow$ expand n_2
- $8 < 6 + 12 \rightarrow$ expand n_3

Uninformed search: Dijkstra's Algorithm

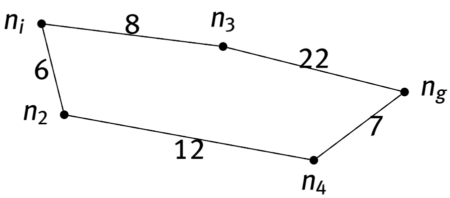
- $f(n) = g(n)$, with $g(n)$ a priority queue
- The node with the lowest cost is expanded
- Completeness: if search space is finite
- Optimal



- $6 < 8 \rightarrow$ expand n_2
- $8 < 6 + 12 \rightarrow$ expand n_3
- n_g reached, but $8 + 22 > 6 + 12$

Uninformed search: Dijkstra's Algorithm

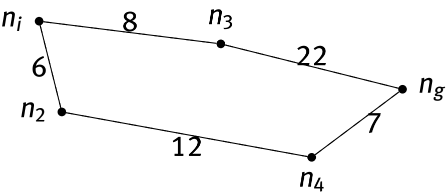
- $f(n) = g(n)$, with $g(n)$ a priority queue
- The node with the lowest cost is expanded
- Completeness: if search space is finite
- Optimal



- $6 < 8 \rightarrow$ expand n_2
- $8 < 6 + 12 \rightarrow$ expand n_3
- n_g reached, but $8 + 22 > 6 + 12$

Uninformed search: Dijkstra's Algorithm

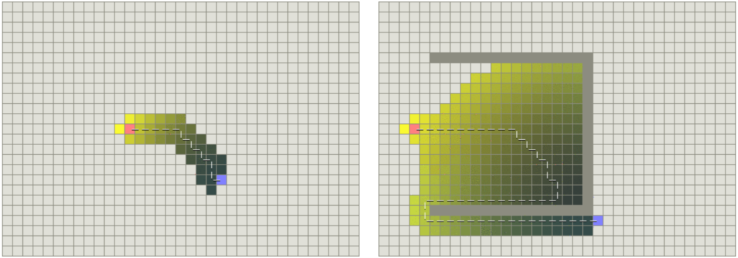
- $f(n) = g(n)$, with $g(n)$ a priority queue
- The node with the lowest cost is expanded
- Completeness: if search space is finite
- Optimal



- $6 < 8 \rightarrow$ expand n_2
- $8 < 6 + 12 \rightarrow$ expand n_3
- n_g reached, but $8 + 22 > 6 + 12$

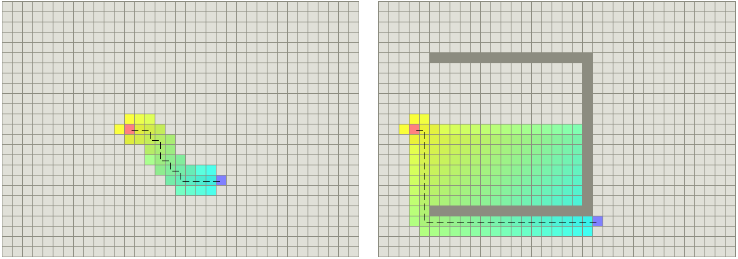
Why not use knowledge of the goal location?

Informed search: Greedy best-first



- $f(n) = h(n)$, with $h(n)$ a heuristic (distance) function
- Expands the node closest to the goal
- Complete
- Non-optimal (see figure)

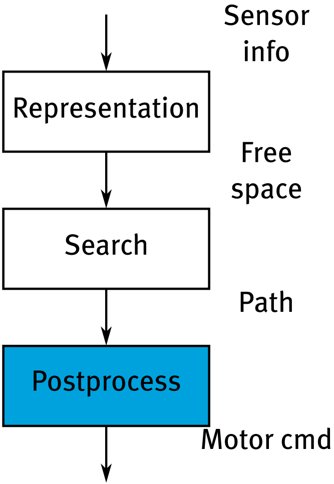
Informed search: A*



- $f(n) = g(n) + h(n)$, with $g(n)$ costs to reach a node and $h(n)$ heuristic to reach the goal
- Takes both costs into account
- Complete
- Optimal if the heuristic function is consistent:
  - $h(n) \leq c(n \rightarrow n') + h(n')$

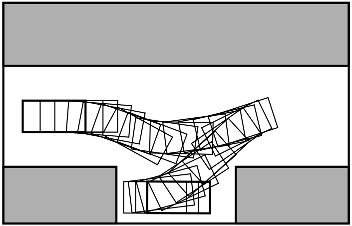
Extending the basic motion planning problem

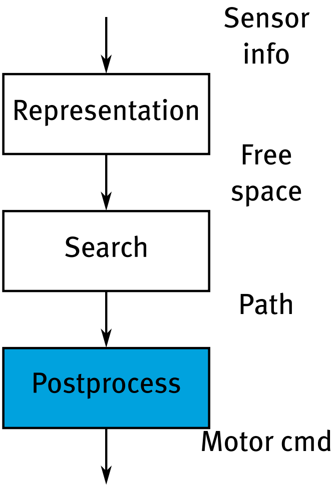
- The path resulting from searching the representation is not yet suitable for execution



Extending the basic motion planning problem

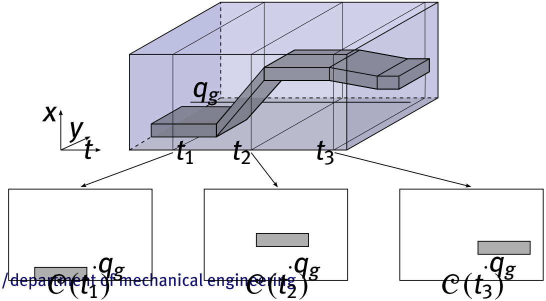
- The path resulting from searching the representation is not yet suitable for execution
- Kinodynamic constraints

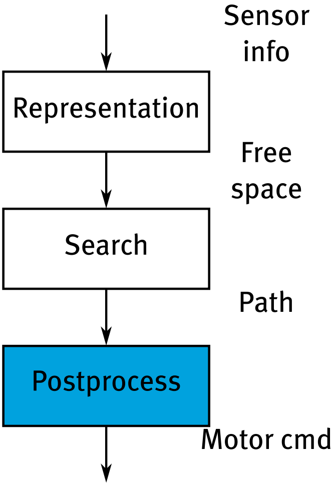




Extending the basic motion planning problem

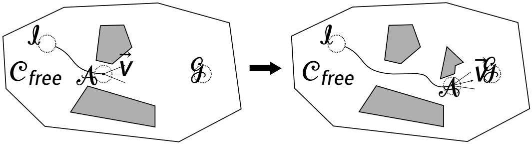
- The path resulting from searching the representation is not yet suitable for execution
- Kinodynamic constraints
- Dynamic environments

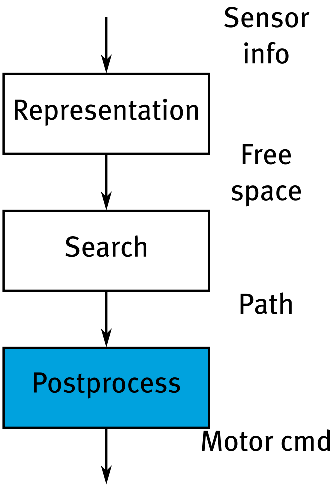




Extending the basic motion planning problem

- The path resulting from searching the representation is not yet suitable for execution
- Kinodynamic constraints
- Dynamic environments
- Uncertainty





Technische Universiteit Eindhoven University of Technology

Kinematic and dynamic constraints

Decoupled trajectory planning

- Path planning: collision free path c in $\mathcal{C}_{\text{free}}$
- Transform c into c', satisfying non-holonomic constraints
- Compute timing function such that $c'(t)$ satisfies kinodynamic constraints

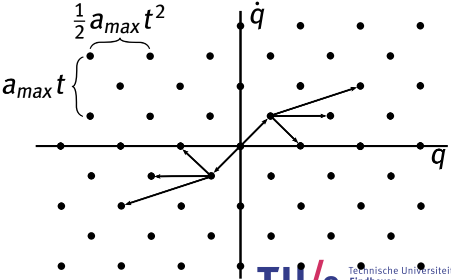
Technische Universiteit Eindhoven University of Technology

Decoupled trajectory planning

- Path planning: collision free path c in $\mathcal{C}_{\text{free}}$
- Transform c into c', satisfying non-holonomic constraints
- Compute timing function such that $c'(t)$ satisfies kinodynamic constraints

Direct trajectory planning

- Searching on a lattice
- Sampling based methods: select input at random from set of admissible controls



Kinematic and dynamic constraints

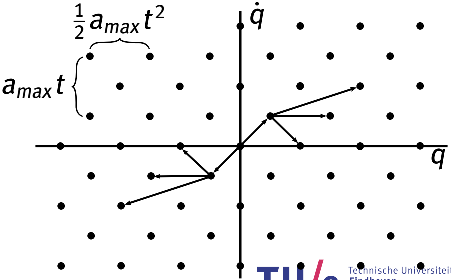
Decoupled trajectory planning

- Path planning: collision free path c in $\mathcal{C}_{\text{free}}$
- Transform c into c', satisfying non-holonomic constraints
- Compute timing function such that $c'(t)$ satisfies kinodynamic constraints

Direct trajectory planning

- Searching on a lattice
- Sampling based methods: select input at random from set of admissible controls

Motion primitives



Robustness against uncertainty

Re-planning (of an entire path)

- Re-planning from the current situation
- Reuse information of previous searches (incremental search)
- The planner can return an (approximate and suboptimal) plan at any time (anytime planning)

Hierarchical planning

- Reduction of complexity: divide the planning problem into global and local planner
  - Global planner: computes a path from start to goal
  - Local planner: satisfy kinodynamic constraints

Hierarchical planning

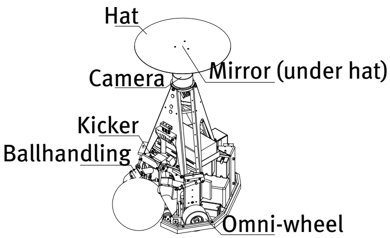
- Reduction of complexity: divide the planning problem into global and local planner
  - Global planner: computes a path from start to goal
  - Local planner: satisfy kinodynamic constraints
- Topological maps
  - Abstract representation that describes relationships between features of the environment
  - Compact and stable w.r.t. sensor noise and uncertainty

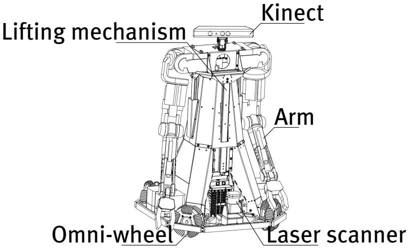
Hierarchical planning

- Reduction of complexity: divide the planning problem into global and local planner
  - Global planner: computes a path from start to goal
  - Local planner: satisfy kinodynamic constraints
- Topological maps
  - Abstract representation that describes relationships between features of the environment
  - Compact and stable w.r.t. sensor noise and uncertainty

How is motion planning applied in TU/e?

Examples: Motion Planning for RoboCup

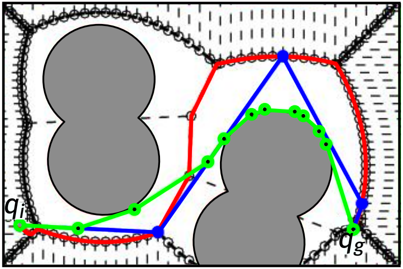


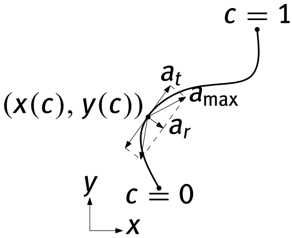


- Soccer pitch
- 12 m $\times$ 18 m
- Known environment
- Dynamic obstacles (hostile)
- 3 m/s

- House/care environment
- Arbitrary size
- Partially unknown
- Static and dynamic obstacles
- 1 m/s

Motion planning for MSL soccer robots

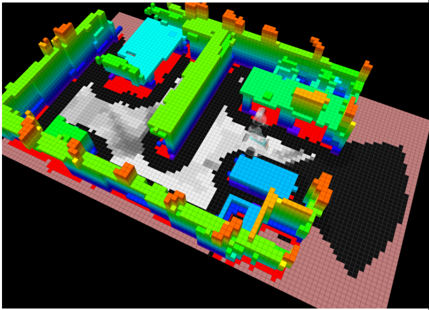




- Voronoi diagram representation, searched with Dijkstra's algorithm
- Shortcut algorithm to cut-off sharp turns
- Time-optimal trajectory through waypoints using Bézier curves

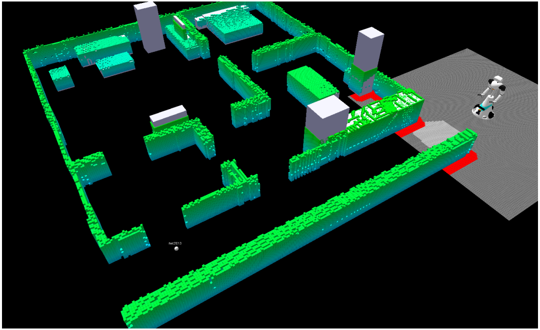
AMIGO: Environment representation

- Use Octomap for 3D navigation
- Project columns to 2D costmap and inflate costs and uncertainty for navigation
- Certainty decays over time instead of known/unknown
  - A wall never moves
  - People are likely to move



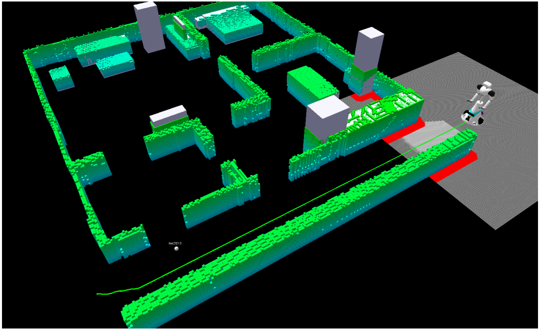
AMIGO: Global and local planner

- Global planner
  - A* Planner
- Local planner
  - Line collision check
  - Velocities based on safety
  - Assumptions on moving obstacles
  - Desired: DWA/MPC



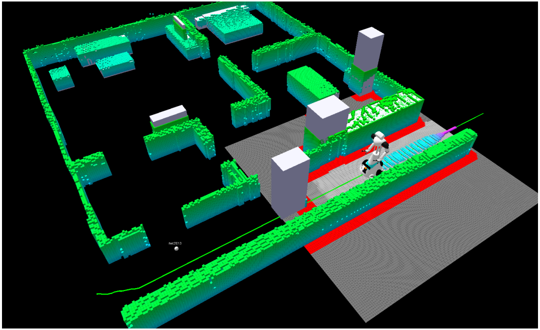
AMIGO: Global and local planner

- Global planner
  - A* Planner
- Local planner
  - Line collision check
  - Velocities based on safety
  - Assumptions on moving obstacles
  - Desired: DWA/MPC



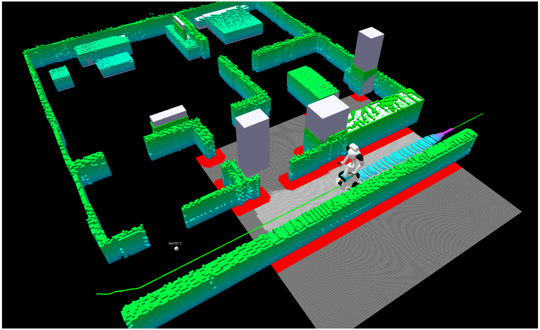
AMIGO: Global and local planner

- Global planner
  - A* Planner
- Local planner
  - Line collision check
  - Velocities based on safety
  - Assumptions on moving obstacles
  - Desired: DWA/MPC



AMIGO: Global and local planner

- Global planner
  - A* Planner
- Local planner
  - Line collision check
  - Velocities based on safety
  - Assumptions on moving obstacles
  - Desired: DWA/MPC



Technische Universiteit Eindhoven University of Technology

Finally

- Local planning methods
- Global planning methods
  - Representations
  - Searching
- Implementations
- Further reading: “Motion Planning for Mobile Robots - A Guide”

Finally

Questions?