TÉCNICO LISBOA

INTRODUCTION TO ROBOTICS

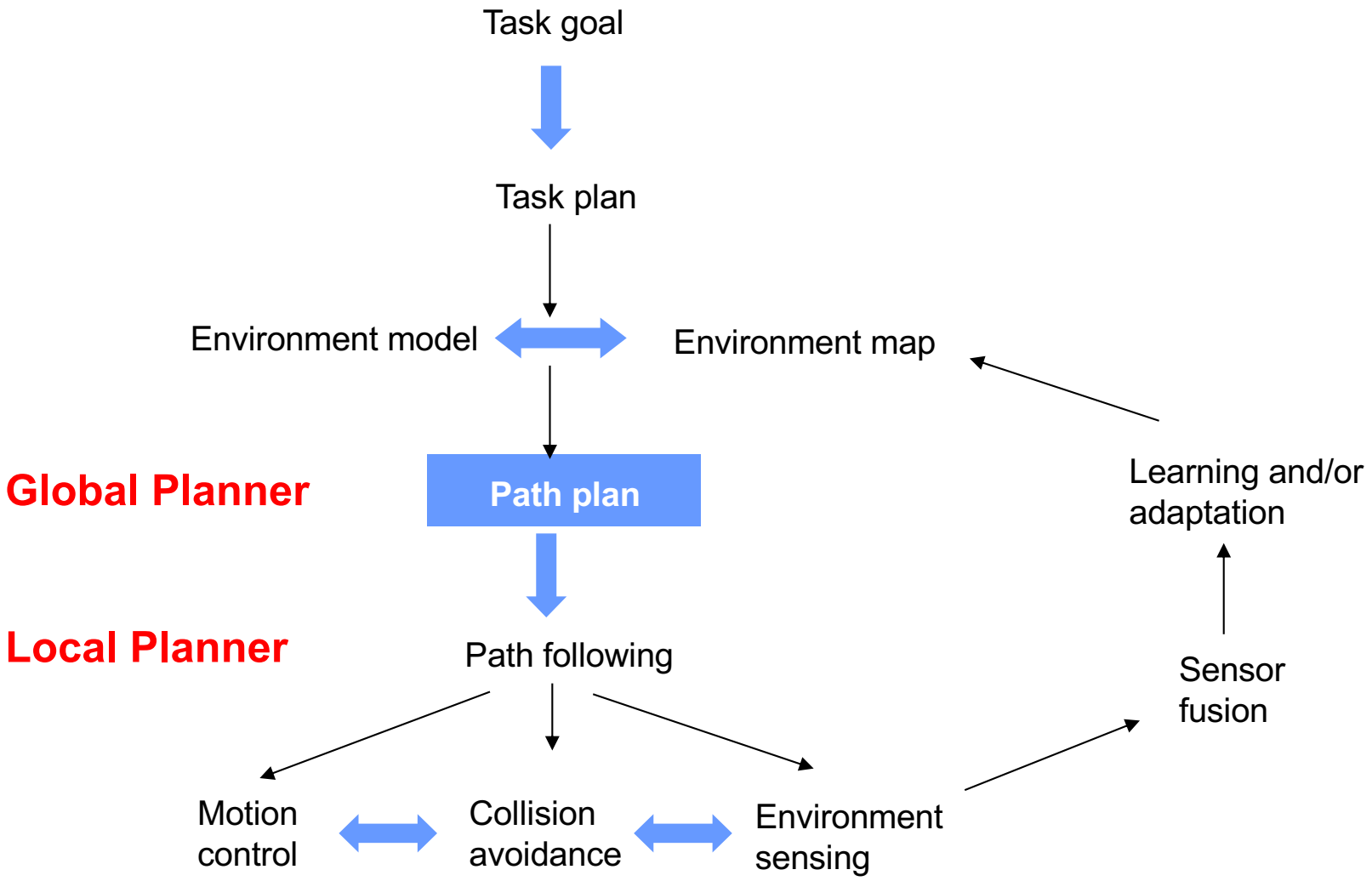
MOTION PLANNING AND GUIDANCE

Pedro U. Lima
Small revision to 2022 course handouts
Instituto Superior Técnico/Instituto de Sistemas e Robótica

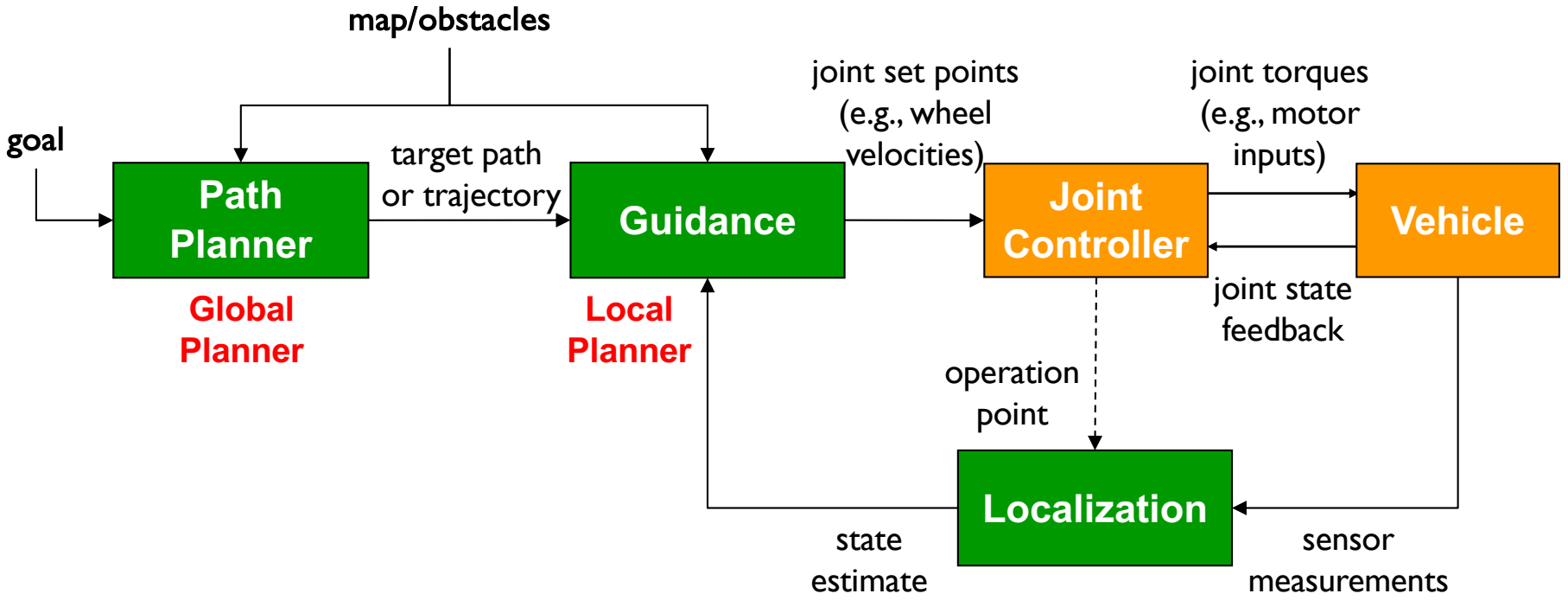
Course Handouts

The Navigation Problem

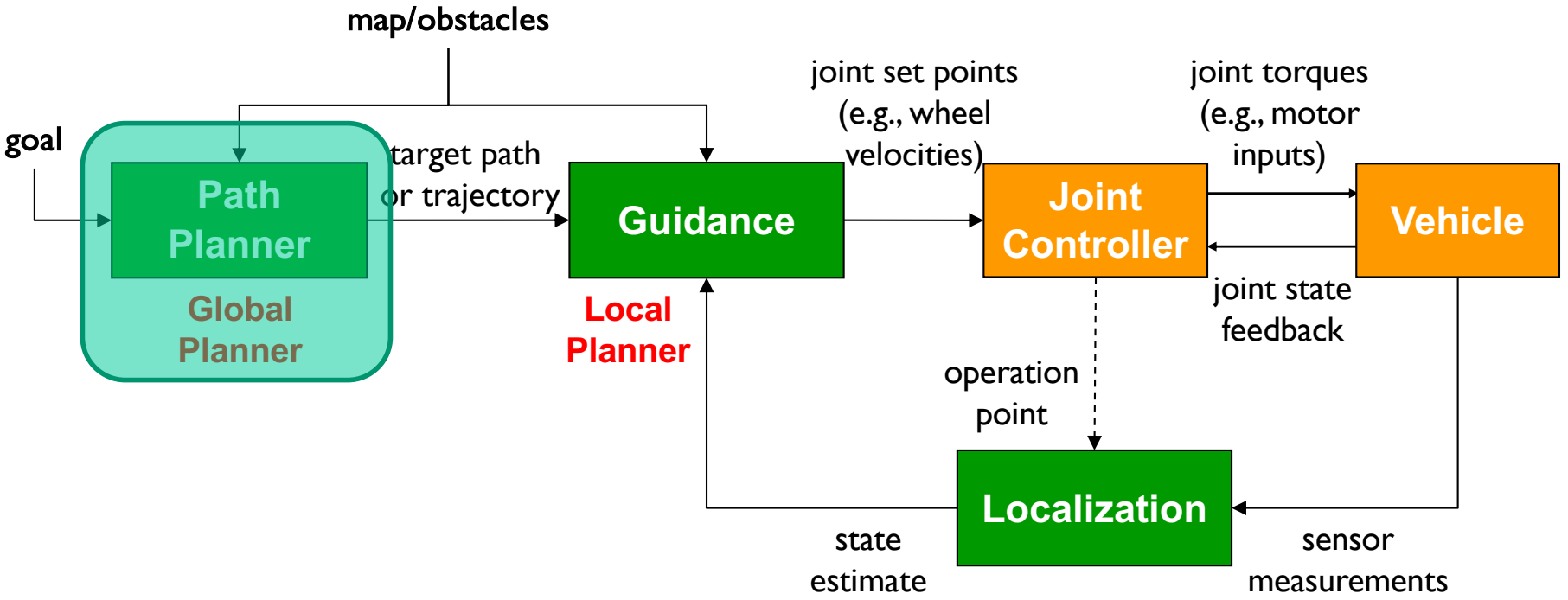
Find a path from a start (S) to a final goal (G) location and traverse it without collisions with initially known and unknown obstacles



The Navigation Loop



The Navigation Loop



Robot Motion Planning

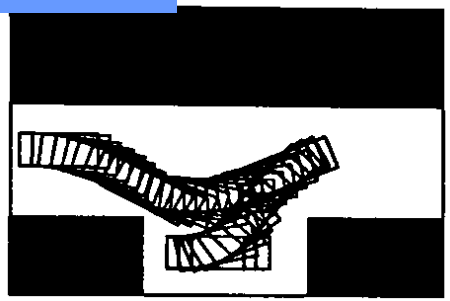
Problem Statement (Global Planner)

- Find a collision free path between an initial pose and the goal, taking into account the constraints (geometrical, physical, temporal)

Problem Classes

- Path Planning
- Trajectory Generation
- Maneuver Planning

MANEUVERS



PATH

A PATH is a geometric locus of way points – in a given space – where the robot must pass

TRAJECTORY

A TRAJECTORY is a path for which a temporal law is specified (e.g., acceleration and velocity at each point)

Robot Motion Planning

Notation

- **A**: single rigid object – (the robot)
- **W**: Euclidean space where **A** moves

(typically $W = \Re^2 \;\; or \;\; \Re^3$)

- $\mathbf{B_1}$, $\mathbf{B_2}$, ..., $\mathbf{B_m}$ fixed rigid objects distributed in **W**. These are the obstacles

Assumptions

- The geometry of **A** and $\mathbf{B_i}$ is known
- The localization of the $\mathbf{B_i}$ in **W** is accurately known
- There are no kinematic constraints on the motion of **A** (**A** is a free-flying object)

Robot Motion Planning

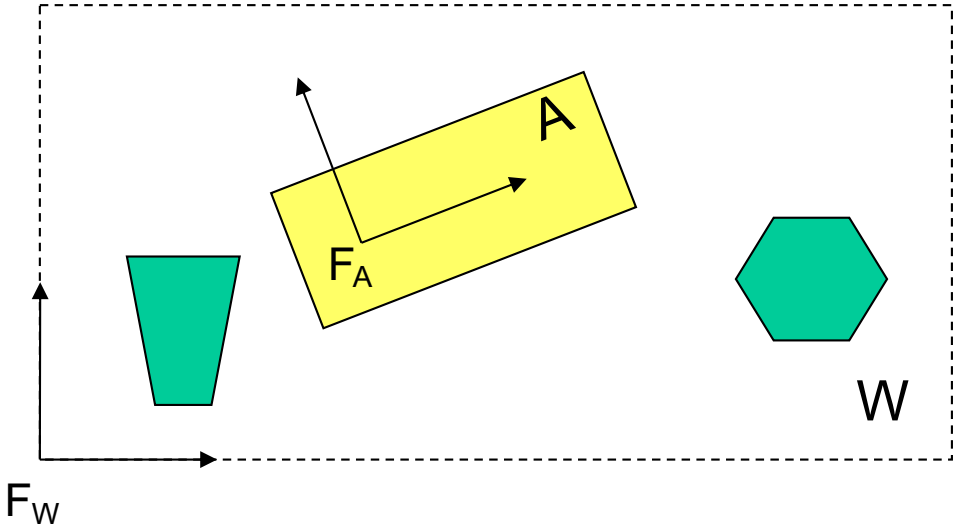
Problem

Given an initial pose and a goal pose of **A** in **W**, generate a path τ specifying a continuous sequence of poses of **A** avoiding contact with the $\mathbf{B}_i$, starting at the initial pose and terminating at the goal pose.

Report failure if no such path exists.

Robot Motion Planning

Configuration Space



F_W – world frame (static)

F_A – robot frame (moving)

The objects $\mathbf{B}_i$ are fixed = any point in $\mathbf{B}_i$ has a fixed position with respect to F_W.

A configuration q of A

pose (position and orientation) of F_A with respect to F_W.

Configuration space of A

space **C** of all configurations of **A**

A(**q**) = subspace of **W** occupied by **A** at configuration **q**

Robot Motion Planning
Configuration Space

path of A

Set of configurations from $\mathbf{q_{init}}$ to $\mathbf{q_{goal}}$, defined as a continuous map:

$$\tau : [0,1] \rightarrow C \qquad \tau(0) = q_{init} \quad \tau(1) = q_{goal}$$

Continuity is defined using a metric topology in **C**.
The topology is induced by an Euclidean distance

- This definition of path considers that **A** is a free-flying object for the path to be feasible
- No objects present

The only constraints on its motions are due to obstacles.
No kinematic or dynamic constraints.

Robot Motion Planning
Configuration Space

OBSTACLES IN THE CONFIGURATION SPACE

- No pose of the robot **A** along a **path** can intersect any object (i.e., a path should be collision free)
- The robot has a given shape
  - Along its path the robot with its shape spans a region of **W**
  - This spanned region is a function of the robot shape and consecutive poses along the path
  - No point of this spanned area can intersect any object

W → C

$\mathbf{B_i}$ → $CB_i = \{q \in C : A(q) \cap B_i \neq \phi\}$

object → **C-object**

Robot Motion Planning

Configuration Space

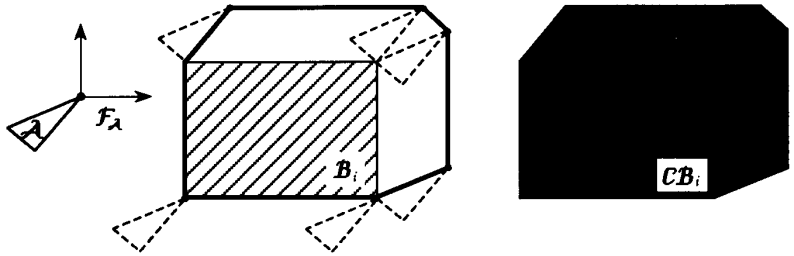
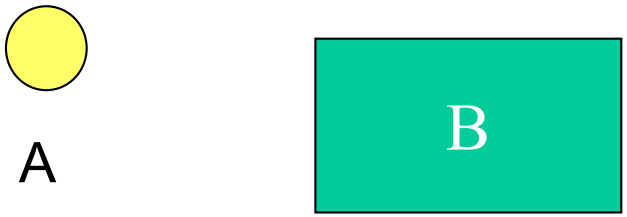


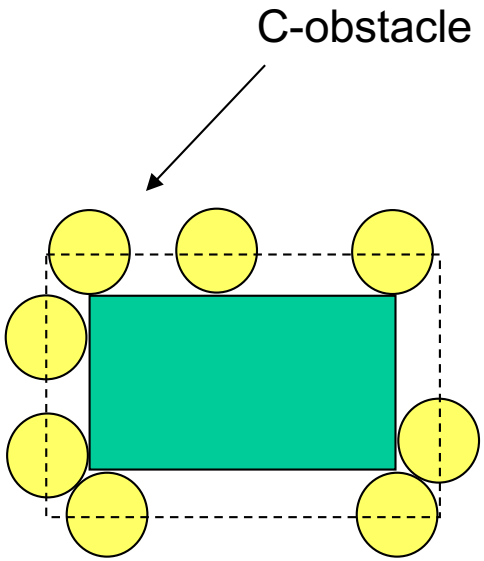
Figure 3. The robot $\mathcal{A}$ (a triangle) can translate freely in the plane at fixed orientation. Its configuration is represented as $\mathbf{q} = (x, y)$, the coordinates in $\mathcal{F}_{\mathcal{W}}$ of the vertex of $\mathcal{A}$ marked as a small circle (the origin of $\mathcal{F}_{\mathcal{A}}$). Hence, $\mathcal{A}$'s configuration space is $\mathcal{C} = \mathbf{R}^2$. The C-obstacle $\mathcal{CB}_i$ (shown dark) is obtained by "growing" the corresponding workspace obstacle $\mathcal{B}_i$ (a rectangle) by the shape of $\mathcal{A}$. Planning a motion of $\mathcal{A}$ relative to $\mathcal{B}_i$ is equivalent to planning a motion of the marked vertex of $\mathcal{A}$ relative to $\mathcal{CB}_i$.

From
Robot Motion Planning
J.C. Latombe

A only has translational motion

A is a disk-shaped robot





Robot Motion Planning

Configuration Space

C-OBSTACLE REGION

$B_1, B_2, \ldots, B_m$ ——— obstacles

CB_i ——— C-obstacle

$\bigcup_{i=1}^{m} CB_i$ ——— C-obstacle region

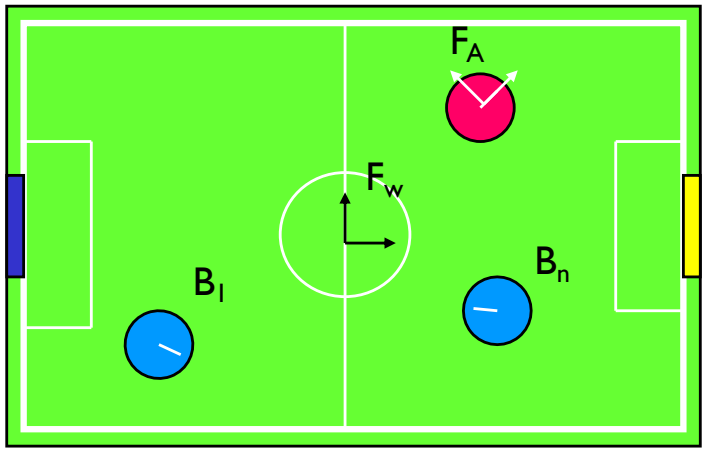
FREE SPACE

$$C_{free} = C \setminus \bigcup_{i=1}^{m} CB_i = \{q \in C : A(q) \cap \bigcup_{i=1}^{m} CB_i = \phi\}$$

Free configuration q iff $q \in C_{free}$

Robot Motion Planning

Basic Problem



- A – single rigid object (the robot)
- W – Euclidean space where A is moving (e.g., R^2)
- B_1,..., B_n – rigid non-mobile objects distributed in W (obstacles)
- Assumptions:
  - A, B_i's geometry known
  - Posture of B_i's in F_w known
  - A has no kinematic constraint (*free-flying object*)

Problem
Given an initial position and orientation and a goal position and orientation of A in W, generate a path specifying a continuous sequence of postures of A avoiding contact with B_i's, starting at the initial posture and ending at the goal posture. (Latombe,91)

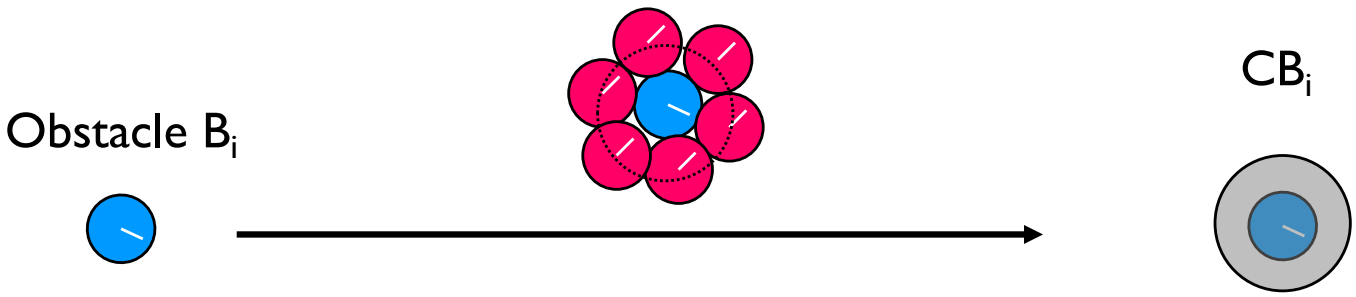
Robot Motion Planning

Basic Problem

- configuration q of A – posture of F_A with respect to F_w (e.g., (x_A, y_A, θ_A))
- configuration space C – space of all configurations
- A(q) – subspace of W occupied by A for configuration q
- C-obstacle – obstacles in configuration space

$$CB_i \equiv \{q \in C : A(q) \cap B_i \neq \Phi\}$$

$$C_{\text{free}} \equiv C \setminus \bigcup_{i=1}^{n} CB_i$$



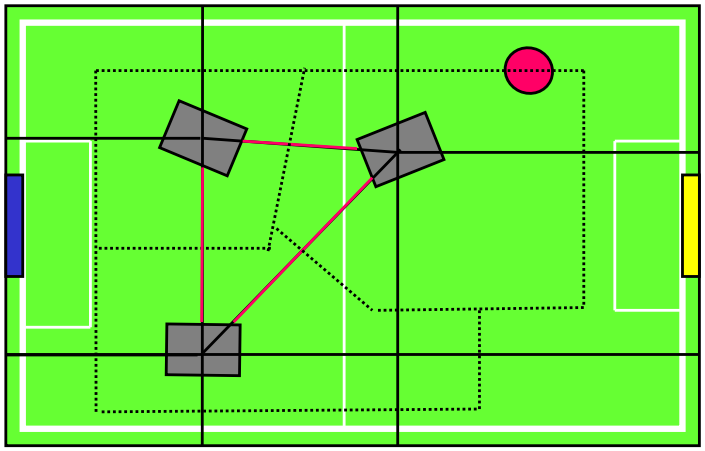
Path Planning Approaches

- Roadmap
  - A graph is defined from C_{free} (Roadmap)
  - Ways to obtain the Roadmap: Visibility graph, Voronoi diagram
- Cell Decomposition
  - The robot free space (C_{free}) is decomposed into simple regions (cells)
  - The path between two poses of a cell can be easily generated
- Sampling-Based Planning
  - Rapidly Exploring Dense Trees
- Potential Field
  - Path planning is based on a powerful analogy, where the robot is treated as a particle acting under the influence of a potential field
- Bug Algorithms

Voronoi Diagram

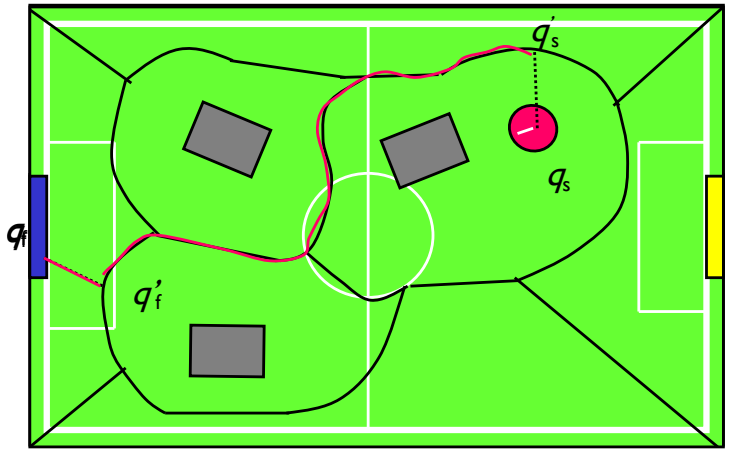
Originally, the Voronoi Diagram partitions C_{free} in sets of points closest to *seed* points (e.g., the center of mass of the obstacles).

A similar diagram can be obtained by Delaunay triangulation using the seed points, modified by taking the line segments between the obstacle boundaries only.



Generalized Voronoi Diagram

One-dimensional subset of C_{free} that maximizes the clearance between the robot and the obstacles



When C_{free} is the interior of a bounded polygonal region:

- A straight arc in $Vor(C_{free})$ is the set of configurations closest to the same pair of edges, or the same pair of vertices.
- A parabolic arc in $Vor(C_{free})$ is the set of configurations closest to the same (edge,vertex) pair
- Each point in the open regions of C_{free} (those delimited by the arcs of the Voronoi diagram and the edges of the boundary of C_{free}) has minimum distance to one single point of the obstacles boundary

Generalized Voronoi Diagram

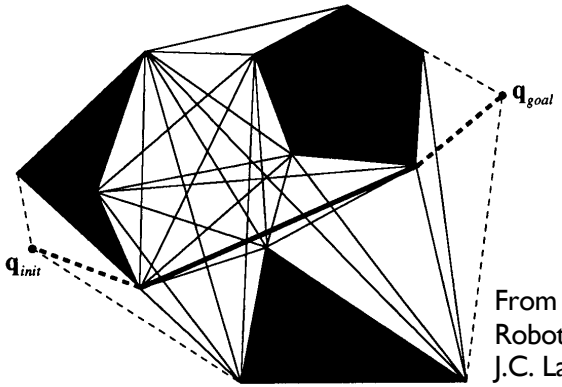
n is the number of vertices in the boundary of C_{free}

Complexity of $Vor(C_{free})$:

- naive algorithm: $O(n^4)$, by considering the $O(n^2)$ pairs (edge,edge), (edge,vertex), (vertex,vertex), and computing the intersection of the corresponding equidistant curves
- the number of arcs of $Vor(C_{free})$ is $O(n)$ – this leads to an algorithm in $O(n^2)$ time
- other: $O(n \log^2 n)$ time
- optimal: $O(n \log n)$ time

Roadmap: Visibility Graph

- Hypothesis: Polygonal C-obstacles
- VISIBILITY GRAPH = non directed graph whose nodes are:
  - The initial pose (q_{init})
  - The goal pose (q_{goal})
  - The vertices of the obstacles (in the C-obstacle space)



From
Robot Motion Planning
J.C. Latombe

Two nodes are linked by a line segment if the straight line segment joining them does not intersect any obstacle

The plain lines connect the obstacle vertices → Roadmap

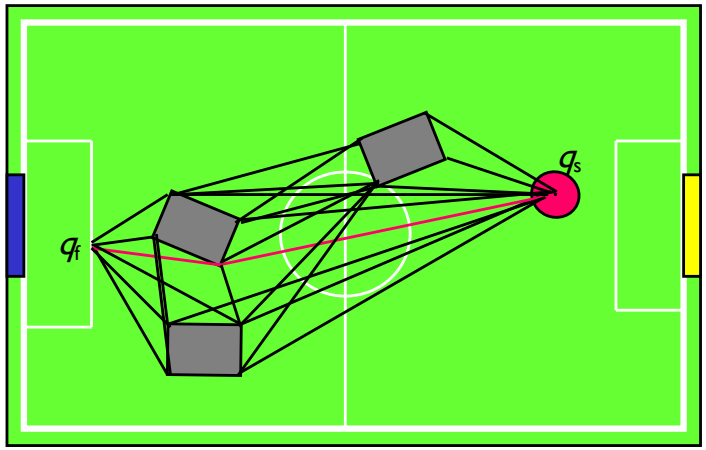
Independent of the initial and goal poses

The dashed lines connect q_{init} and q_{goal} to the roadmap

- HOW TO FIND THE SHORTEST PATH ?
- IS THIS PATH SMOOTH ?
- IS THIS PATH COLLISION FREE?

Roadmap: Visibility Graph

Non-directed graph whose nodes are: the initial posture, the goal posture, the vertices of the obstacles. Two nodes are linked by a line segment if the straight-line segment joining them does not intersect any obstacle



- Applies to 2D configuration spaces with polygonal C-obstacles only
- One of the earliest path planning methods (Nilsson, 1969)

Roadmap: Visibility Graph

n is the number of vertices of CB

Complexity of building the Visibility Graph algorithm:

- naive algorithm: $O(n^3)$
- other: $O(n^2 \log n)$, $O(n^2)$ time

Complexity of finding the shortest path (A* algorithm)

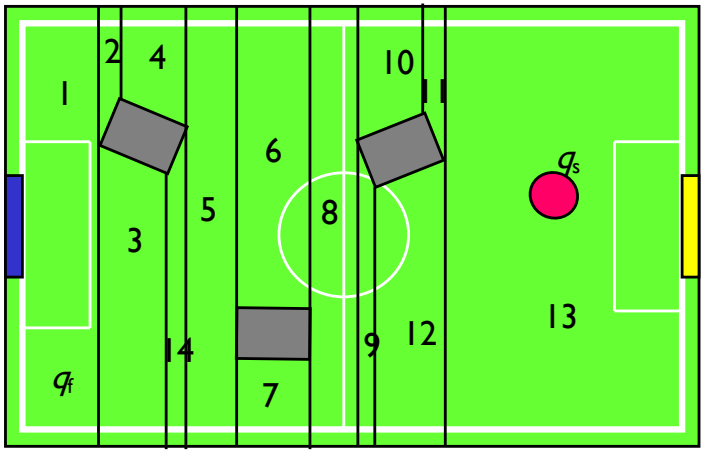
- $O(n^2)$ time (if the list OPEN is not sorted)
- $O(k \log n)$ time (if the list OPEN is sorted), where k is the number of edges of the visibility graph

Cell Decomposition Methods

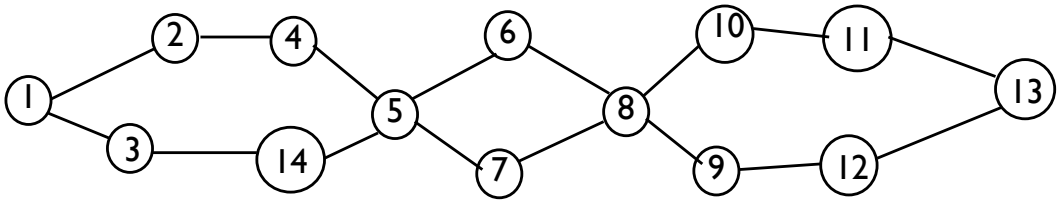
Exact Methods: the free space is decomposed into cells whose unions is exactly the free space (i.e., a partition of C_{free})

Example:

1. decomposing the free space into trapezoidal cells



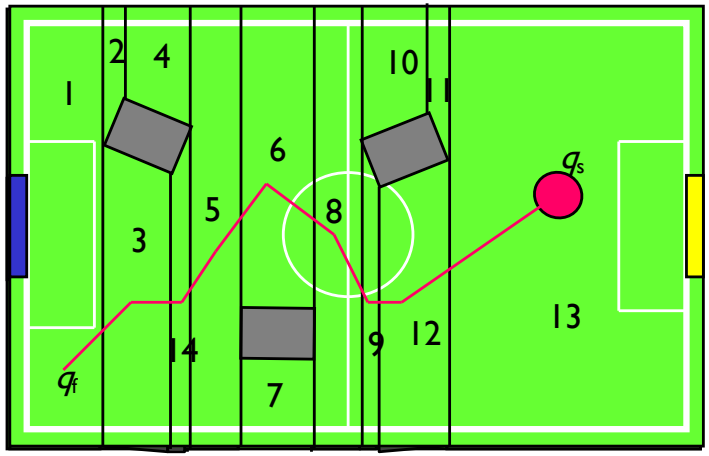
2. Building a connectivity graph representing the adjacency relation between the cells



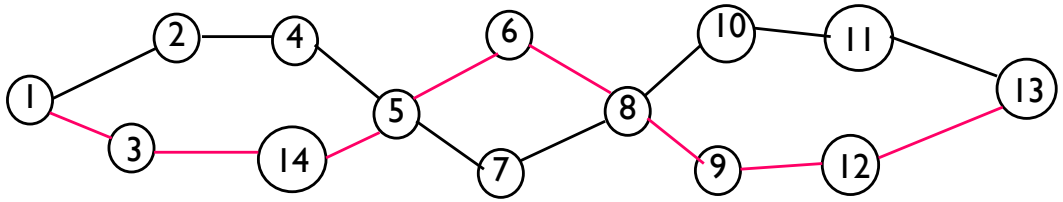
Cell Decomposition Methods

Exact Methods: the free space is decomposed into cells whose unions is exactly the free space (i.e., a partition of C_{free})

3. Search the graph for a path (sequence of adjacent cells)



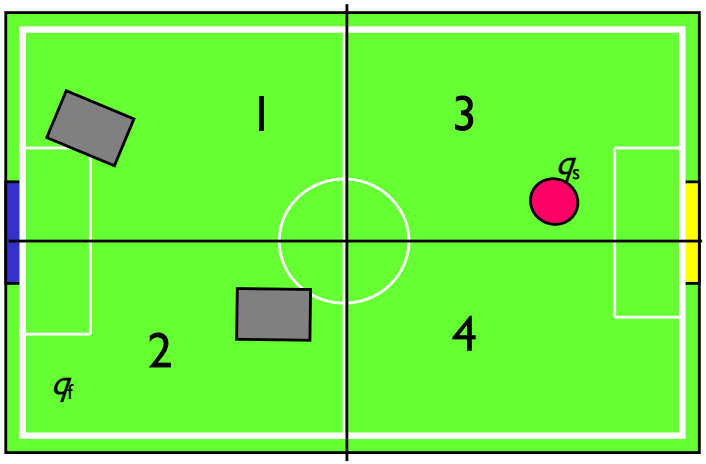
4. Transform the sequence of cells into a free path

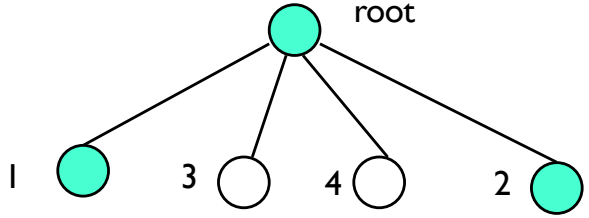


Cell Decomposition Methods

Approximate Methods: the free space is decomposed into cells of pre-defined shape (e.g., squares) whose union is strictly included in the free space

Most usual method: QUADTREE DECOMPOSITION

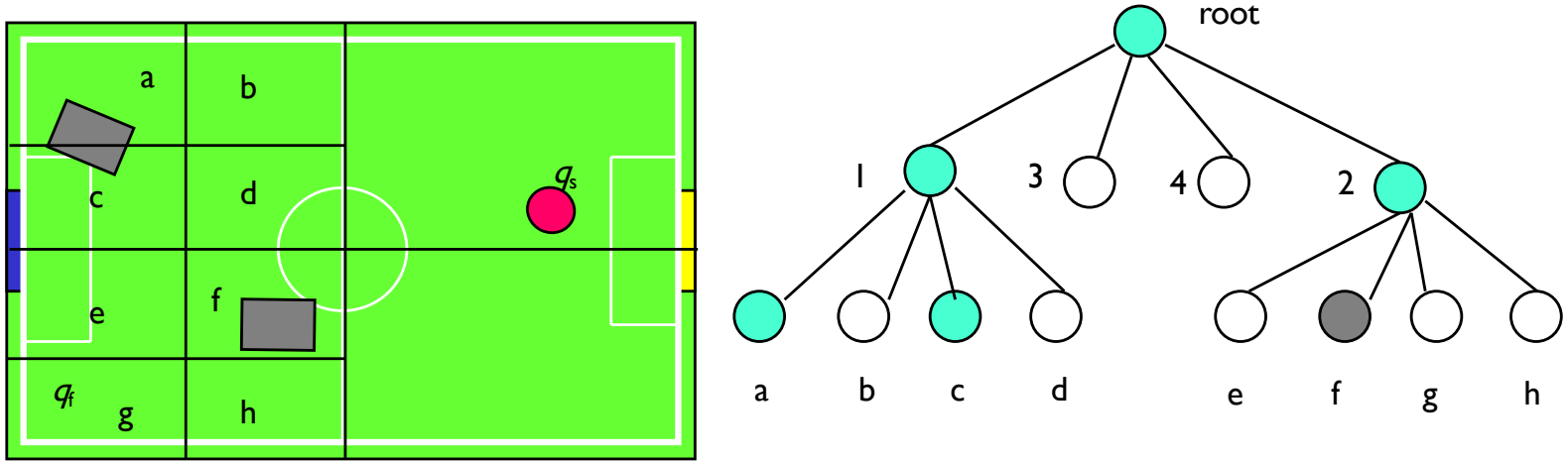




Cell Decomposition Methods

Approximate Methods: the free space is decomposed into cells of pre-defined shape (e.g., squares) whose union is strictly included in the free space

Most usual method: QUADTREE DECOMPOSITION



Tree leaves:

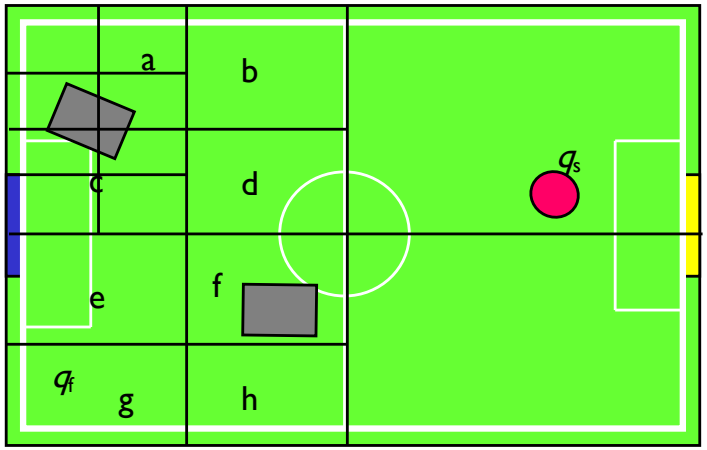
Occupied cells

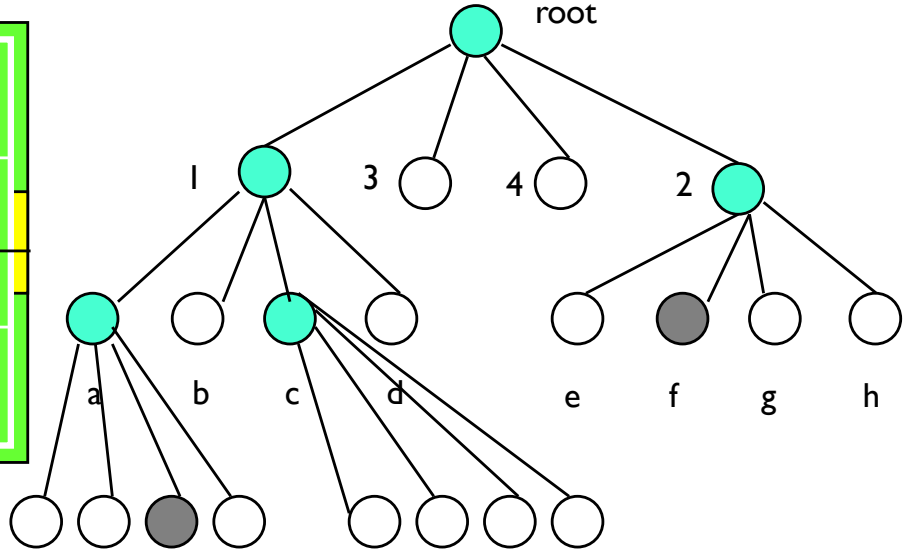
Free cells

Cell Decomposition Methods

Approximate Methods: the free space is decomposed into cells of pre-defined shape (e.g., squares) whose union is strictly included in the free space

Most usual method: QUADTREE DECOMPOSITION





Tree leaves:

Occupied cells

Free cells

Tradeoff between accuracy and complexity

Cell Decomposition Methods

n is the number of vertices of *CB*

Complexity of the trapezoidal decomposition algorithm:

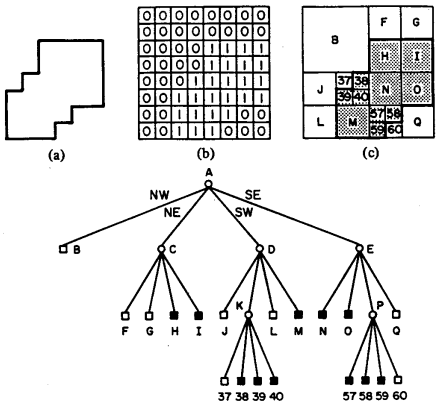
- cell decomposition + graph + start and goal cell id: $O(n \log n)$
- shortest path search (A^*): $O(n \log n)$

Number of nodes in a quadtree of height $h = 4^h$

The A* algorithm in Path Planning

Multiresolution Planners (an example)

S.K.Kambhampati, L.S.Davis, "Multiresolution Path Planning for Mobile Robots" IEEE Journal of Robotics and Automation, RA-2, 3, pp.135-145, 1986.



Quadtree representation of world model

ALGORITHM

1. Robot → point. The objects are extended to include the robot dimensions.
2. Given the Start and Goal points, determine the quadtree leaf nodes S and G, representing the regions containing these points.
3. Plan a minimum cost path between S and G in the graph formed by the non-obstacle leaf nodes, using the A* algorithm.

Cost function at an arbitary node c

$$f(c) = g(c) + h(c)$$

- $h(c)$: Heuristic estimate of the cost of the remaining path from c to G
- $g(c)$: Cost of the path from S to c

$$g(c) = g(p) + \tilde{g}(p,c)$$

- $\tilde{g}(p,c)$: Cost of the path segment between p and c
- $g(p)$: Cost of the path from S to the c's predecessor p on the path

The A* algorithm in Path Planning

Cost function at an arbitary node n

$$f(c) = g(c) + h(c)$$

- $h(c)$: Heuristic estimate of the cost of the remaining path from n to G
  - → Euclidean distance between the midpoints of the regions represented by c and G
- $g(c)$: Cost of the path from S to c

$$g(c) = g(p) + \tilde{g}(p,c)$$

- $\tilde{g}(p,c)$: Cost of the path segment between p and c
- $g(p)$: Cost of the path from S to the c's predecessor p on the path

$$\tilde{g}(p,c) = D(p,c) + \alpha\, d(c)$$

- $d(c)$: Cost incurred by including node c on the path. d(c) depends upon clearance of the node c from the nearby obstcales
- $D(p,c)$: Distance between nodes p and c (half the sum of the nodes size)

The A* algorithm is guaranteed to find the minimum cost path (as determined by g(c)) to the goal G if the estimate h(c) never overestimates the actual value of the cost of the remaining path to G.

Rapidly Exploring Dense Trees (RDTs)

- incremental sampling and searching approach without any parameter tuning
- dense sequence of samples is used as a guide in the incremental construction of the tree
- *rapidly exploring random tree (RRT)* – random sequence

Rapidly Exploring Dense Trees (RDTs)

```
SIMPLE_RDT(q_0)
1  G.init(q_0);
2  for i = 1 to k do
3      G.add_vertex(α(i));
4      q_n ← NEAREST(S(G), α(i));
5      G.add_edge(q_n, α(i));
```

Reprinted from (Lavalle,2006)

α Infinite, dense sequence of samples in *C*

$\alpha(i)$ i^{th} sample of the sequence of samples in *C*

$\mathcal{G}(V,E)$ Topological graph representing the RDT

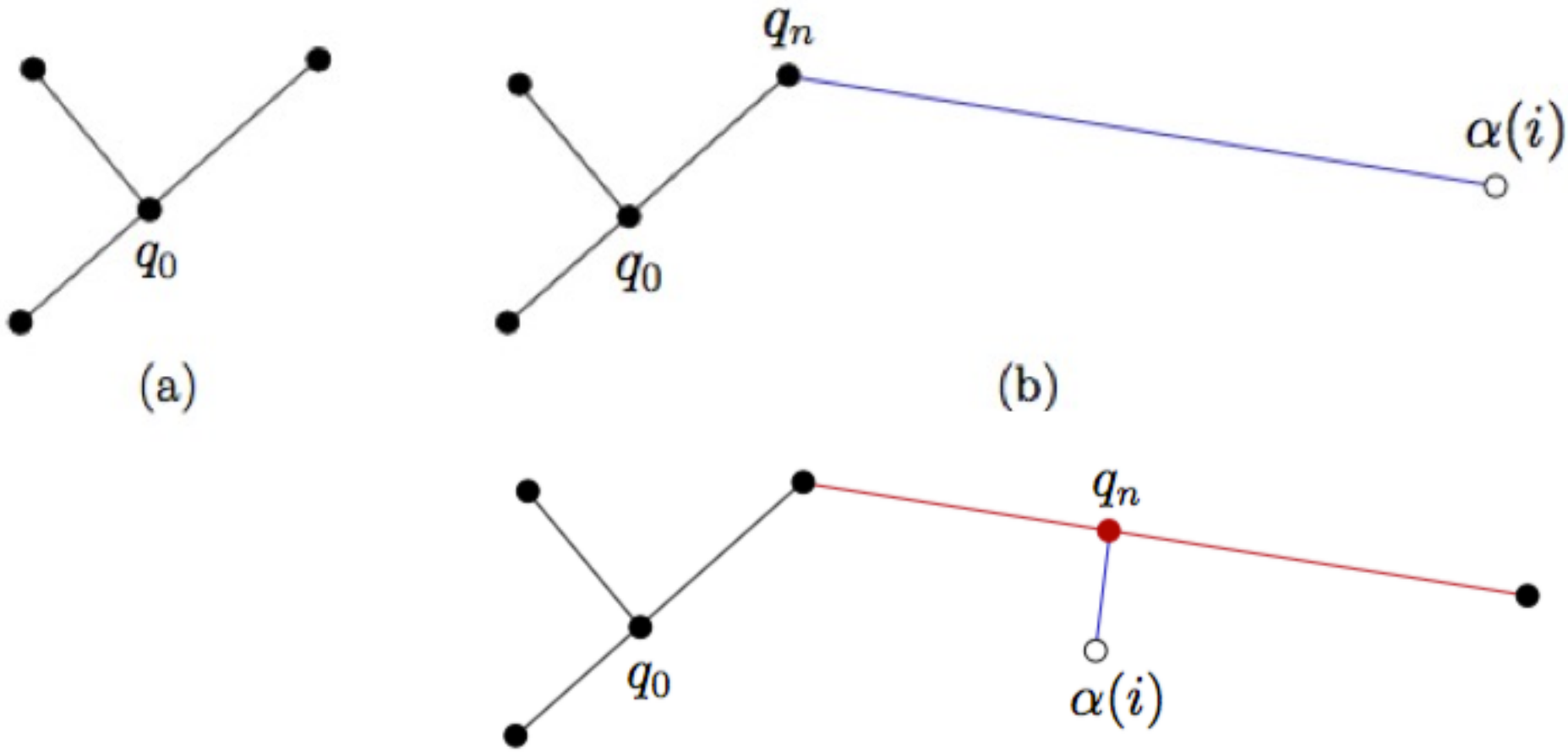
$S \subset C_{free}$ (Swath) Set of points reached by $\mathcal{G}$

$S = \bigcup_{e \in E} e([0,1])$ where $e([0,1]) \in C_{free}$ is the image of path *e*

Rapidly Exploring Dense Trees (RDTs)

C is a metric space
No obstacles

Reprinted from (Lavalle,2006)



Rapidly Exploring Random Trees (RRTs)

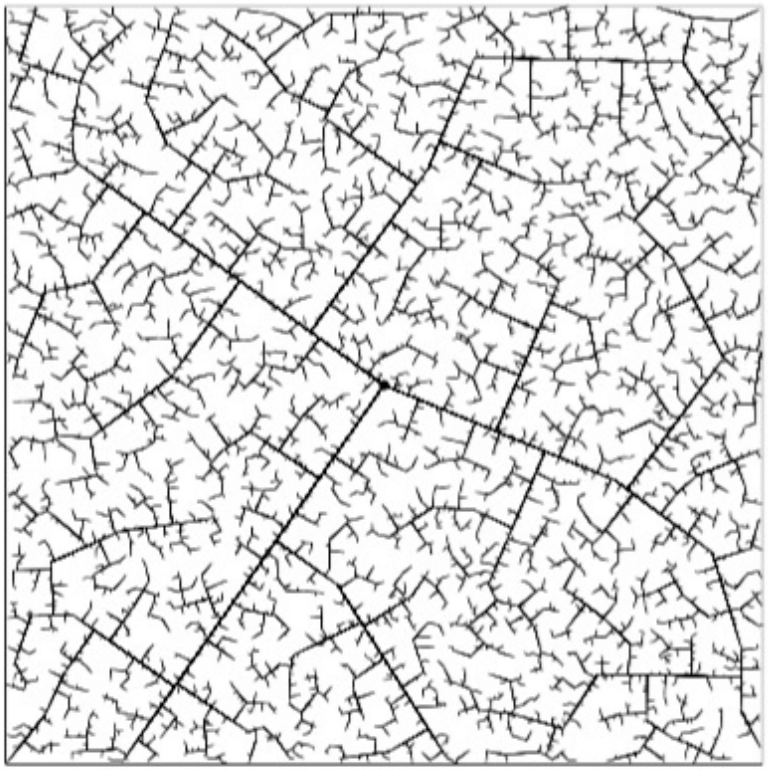
samples are drawn randomly from the state space

$C = [0,1]^2$

$q_0 = (0.5,0.5)$

Reprinted from (Lavalle,2006)

45 iterations



2345 iterations

Rapidly Exploring Random Trees (RRTs)

Reprinted from (Lavalle,2006)

RRT* - more recent variant of RRT that connects the point to the tree in a way that minimizes the overall path length.

costs associated to the edges and

e.g., considering all points in the tree within a *d*-ball

of fixed radius from the point to add and

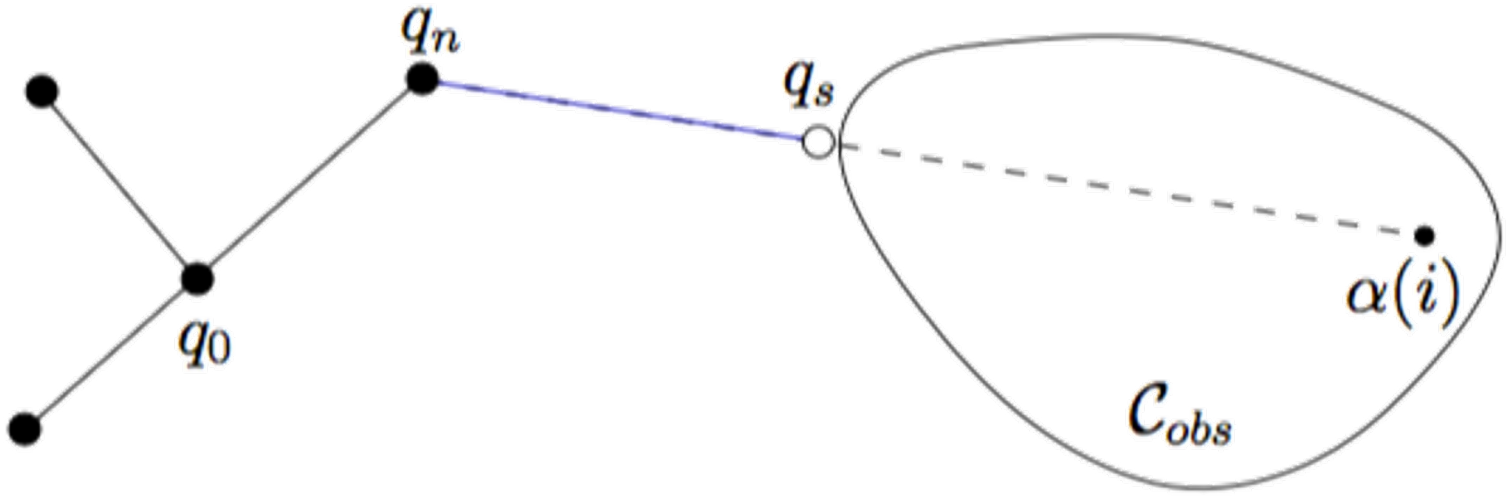
finding the point that minimizes the overall path length to the start.

Rapidly Exploring Dense Trees (RDTs)

With obstacles

```
RDT(q_0)
1  G.init(q_0);
2  for i = 1 to k do
3      q_n ← NEAREST(S, α(i));
4      q_s ← STOPPING-CONFIGURATION(q_n, α(i));
5      if q_s ≠ q_n then
6          G.add_vertex(q_s);
7          G.add_edge(q_n, q_s);
```

Reprinted from (Lavalle,2006)



Rapidly Exploring Dense Trees (RDTs)

Path Planning – Single Tree

Use the RDT_with_obstacle algorithm to grow a tree from q_{init} and periodically check whether it is possible to connect the RDT to q_{goal}

- e.g., start with a dense sequence α and periodically insert q_{goal} with probability p_g
- If p_g is too large → RRT becomes too greedy
- If p_g is too small→ no incentive to reach the goal

Rapidly Exploring Dense Trees (RDTs)

Path Planning – Double Tree

Reprinted from (Lavalle,2006)

```
RDT_BALANCED_BIDIRECTIONAL(q_I, q_G)
 1   T_a.init(q_I); T_b.init(q_G);
 2   for i = 1 to K do
 3       q_n ← NEAREST(S_a, α(i));
 4       q_s ← STOPPING-CONFIGURATION(q_n, α(i));
 5       if q_s ≠ q_n then
 6           T_a.add_vertex(q_s);
 7           T_a.add_edge(q_n, q_s);
 8           q'_n ← NEAREST(S_b, q_s);
 9           q'_s ← STOPPING-CONFIGURATION(q'_n, q_s);
10           if q'_s ≠ q'_n then
11               T_b.add_vertex(q'_s);
12               T_b.add_edge(q'_n, q'_s);
13           if q'_s = q_s then return SOLUTION;
14       if |T_b| > |T_a| then SWAP(T_a, T_b);
15   return FAILURE
```

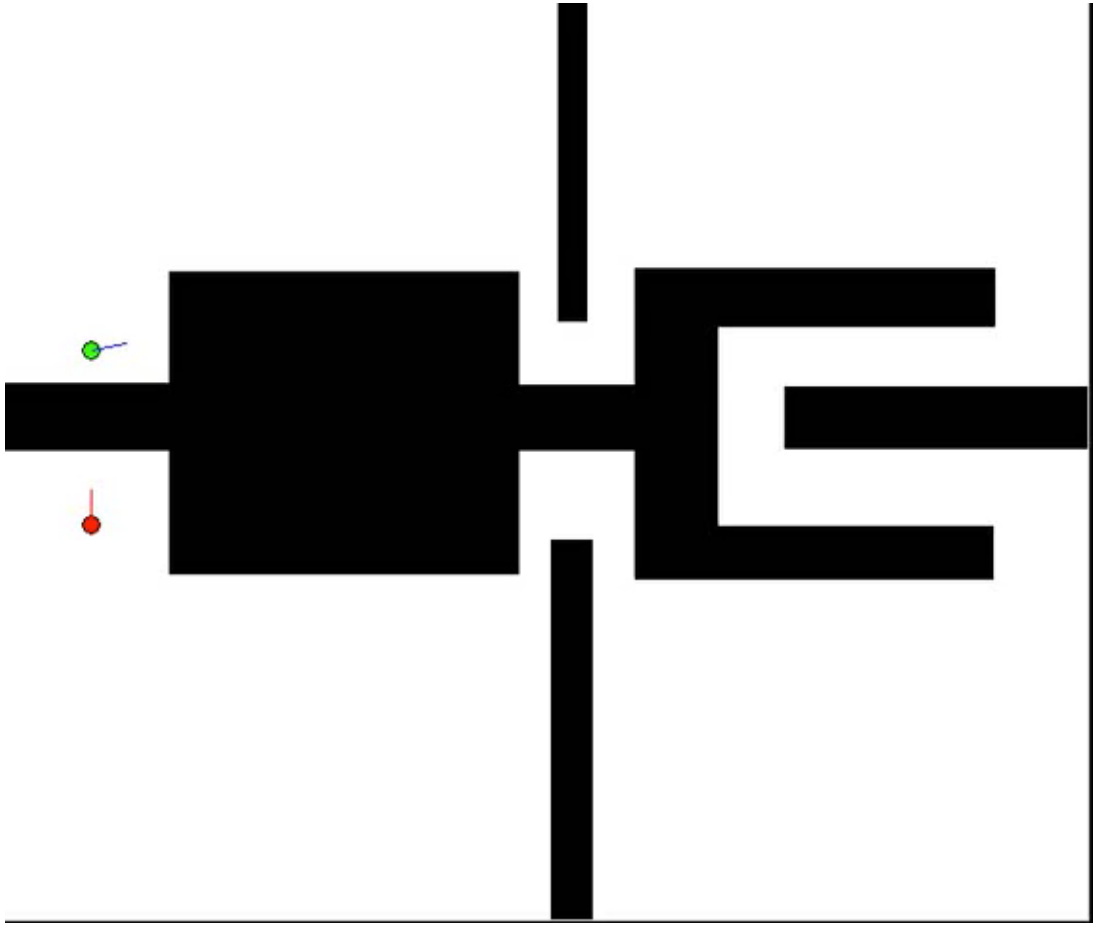
Instead of using a new sample of the dense sequence to grow T_a, the new vertex q'_s is used to grow T_b towards T_a.

Two trees are connected

Two trees are swapped when one is larger than the other (e.g., #vertices; #edges)

Rapidly Exploring Dense Trees (RDTs)

Path Planning – Double Tree



https://youtu.be/E_MC7vWb62A
(vídeo by Dhiraj Gandhi, 26/02/2015)

Planning in Unknown Continuous Environments
Bug Algorithm

S. Lavalle "Planning Algorithms", Cambridge University Press, 2006

Model

- point robot placed into an unknown 2D environment
- finite number of bounded obstacles
- boundary of each obstacle and the outer boundary (if it exists) are piecewise-analytic (each piece is smooth and switches its curvature sign only a finite number of times)
  - polygons,
  - smooth curves,
  - some combination of curved and linear parts.

Two sensors:

1. goal sensor measures
   current Euclidean distance to the goal and
   the direction to the goal

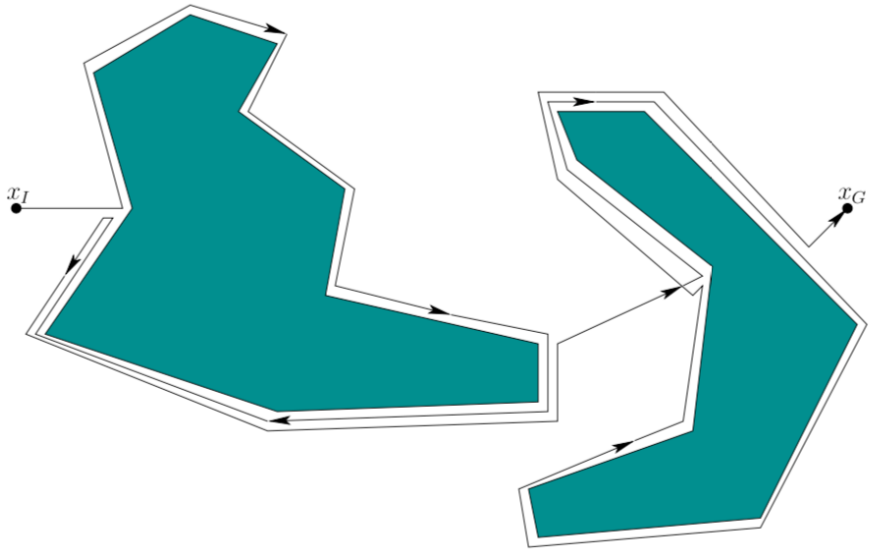
2. local visibility sensor provides
   the exact shape of the boundary **within a small distance** from the robot

Planning in Unknown Continuous Environments
Bug Algorithm

Bug1 Strategy

V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2:403–430, 1987.

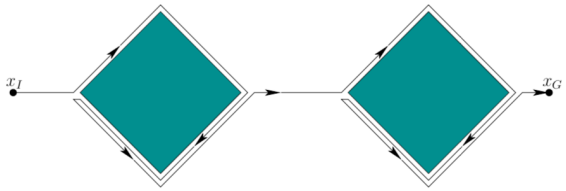
1. Move toward the goal until an obstacle or the goal is encountered. If the goal is reached, then stop.
2. Turn left and follow the entire perimeter of the contacted obstacle. Once the full perimeter has been visited, then return to the point at which the goal was closest, and *go to Step 1*.



worst case distance travelled

$$d + \frac{3}{2}\sum_{i=1}^{M} p_i$$

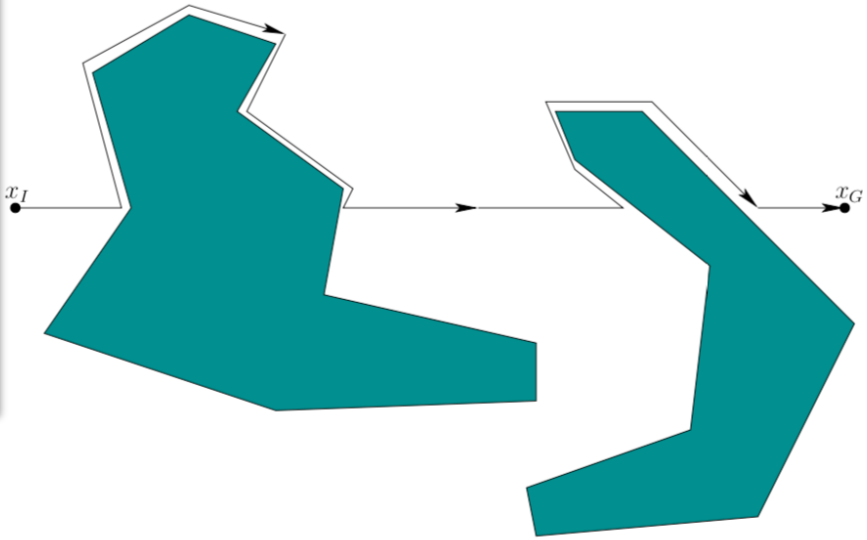
d is the Euclidean distance from the initial position to the goal position,
p_i is the perimeter of the i^{th} obstacle
M is the number of obstacles



Planning in Unknown Continuous Environments
Bug Algorithm

Bug2 Strategy

1. Move towards the goal along the line connecting initial and goal positions, until an obstacle or the goal is encountered. If the goal is reached, then stop.
2. The robot follows the perimeter only until the line is reached and it is able to move in the direction towards the goal. Then, *go to Step 1*.

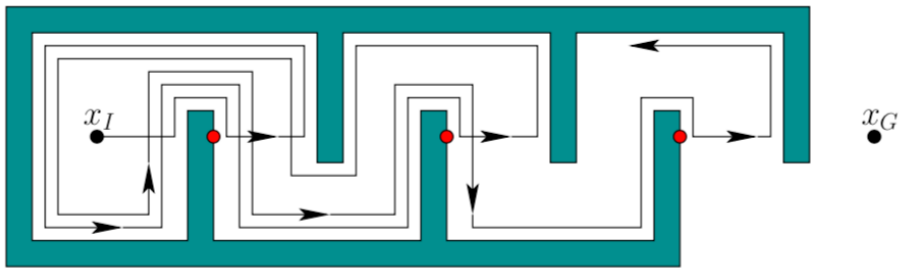


distance travelled $d + \frac{1}{2}\sum_{i=1}^{M} n_i p_i$

n_i is the number of times the i^{th} obstacle crosses the line segment between the initial and goal positions

Infinite cycles may occur

Modification (step 2):
The robot detects it is about to leave from a point on the boundary which it has already visited, it instead continues along the boundary until reaching a arriving at a new point on the line it has not yet visited.
This is applied iteratively until the goal is reached or it is deemed to be impossible.



Planning in Unknown Continuous Environments
Potential Fields

Robot is like an electrically-charged particle in an electrical field

U is the potential
F is the force acting over the robot

F pulls the robot towards the goal and pushes it away from the obstacles

Assumes free-flying robot

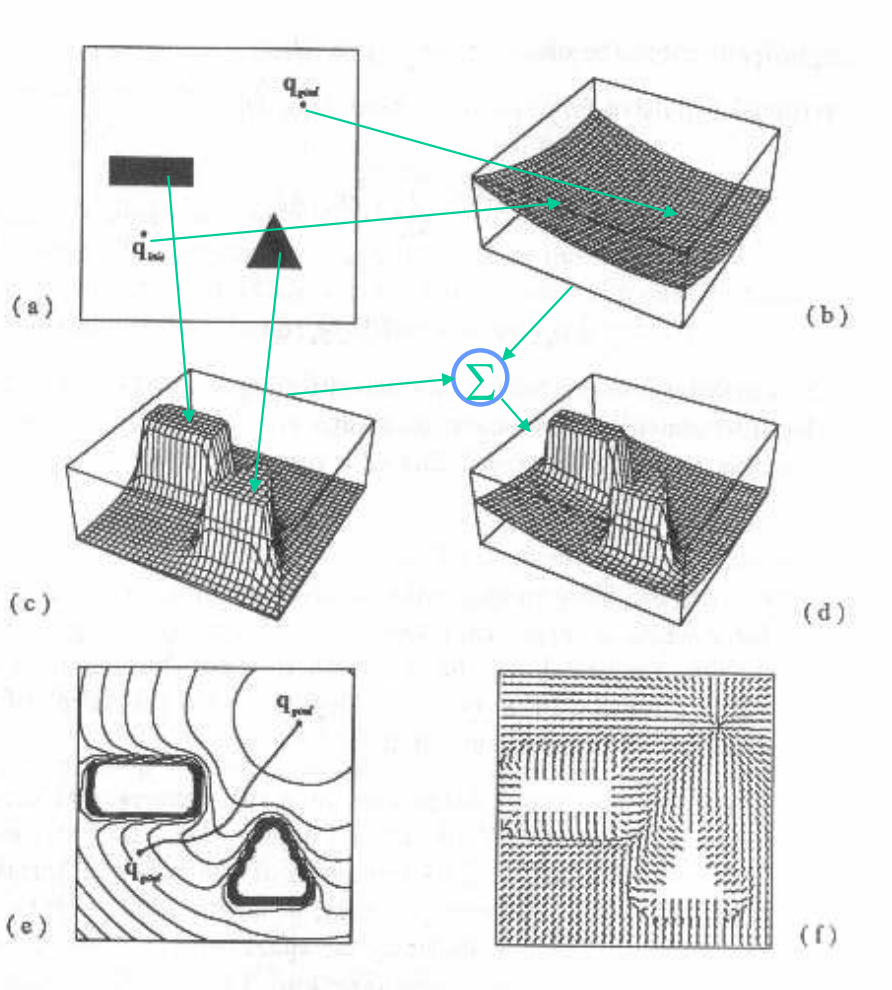
(Khatib, 1985)

$$(b)\ U_{att}(q) = \frac{1}{2}\xi\, d^2_{goal}(q),\ d_{goal}(q) = \left\| q - q_{goal} \right\|$$

$$(c)\ U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{d_{obs}(q)} - \frac{1}{d_0}\right)^2 & \text{if } d_{obs}(q) \le d_0 \\ 0 & \text{if } d_{obs}(q) > d_0 \end{cases}$$

$$(d)\ U(q) = U_{att}(q) + U_{rep}(q)$$

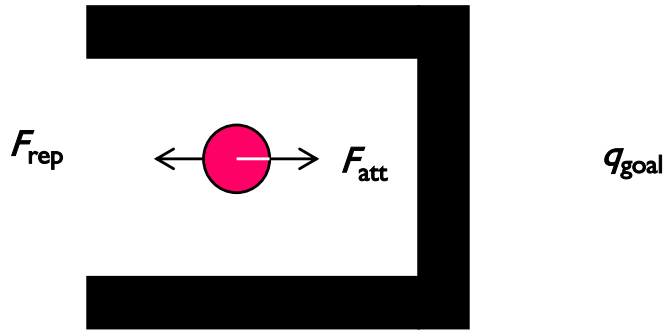
$$(e,f)\ F(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q)$$



in (Latombe, 1991)

Problems with Potential Fields

- oscillations in narrow passages
- robot gets stuck in local minima



Possible solutions:

- build potential function with a single minimum
- detect local minimum (robot oscillates) and escape (e.g., with random moves)
- change to other methods...

Path Planning
Kinematics Constraints

Kinematic Constraints

- In the basic path planning problem the only constraint of robot motion is due to obstacles
- There may occur other constraints – kinematic constraints (objects that cannot translate and rotate freely in the workspace)
- Two types of kinematic constraints
  - **Holonomic Constraints**
    - Do not fundamentally change the path planning problem
  - **Nonholonomic Constraints**
    - Much harder to handle regarding path planning

Holonomic Constraints

F is a smooth function with non-zero derivative

$$F(q,t) = 0$$

$$F(q_1, q_2, \ldots, q_m, t) = 0$$

1 holonomic constraint = Relation (equality or inequality) among the parameters of **C** that can be solved for one of them as a function of the others

k independent holonomic constraints → **dim C=m-k**

Path planning is done in a submanifold of C with dimension m-k as if there were no constraints

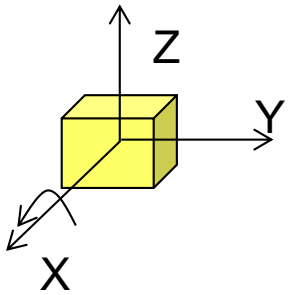
Kinematic Constraints
Holonomic Constraints

Holonomic Constraints - Example

A tridimensional object that:

Can freely translate

Has a rotation along a fixed axis (relative to F_A)



- Pitch angle = yaw angle = 0
- These two independent equations constrain the configuration

dim C = 6 $\quad q = (x, y, z, \theta, \varsigma, \psi)$

$\zeta = 0, \ \psi = 0$ holonomic constraints

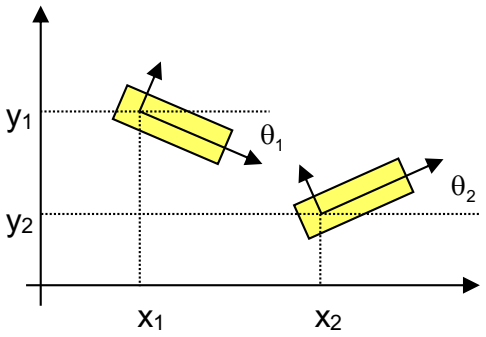
dim = 4

Kinematic Constraints
Holonomic Constraints

Holonomic Constraints - Example

2 Planar Robots

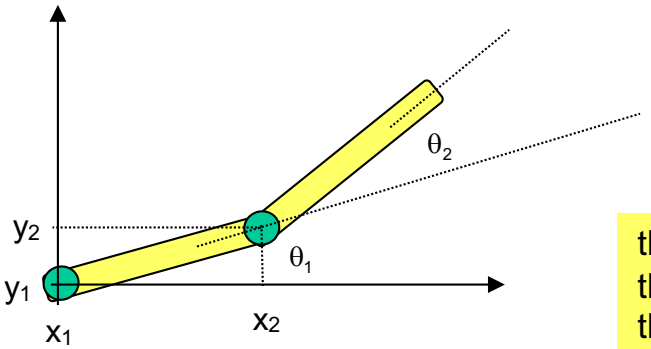
Articulated Robots



$$q = (x_1, y_1, \theta_1, x_2, y_2, \theta_2)$$

dim C = 6

there are no holonomic constraints



$x_1 = y_1 = 0$

$y_2 - x_2 \text{ tg } \theta_1 = 0$

there are two holonomic constraints
that remove 4 DoF
the configuration is fully described by
$C' = (\theta_1, \theta_2)$

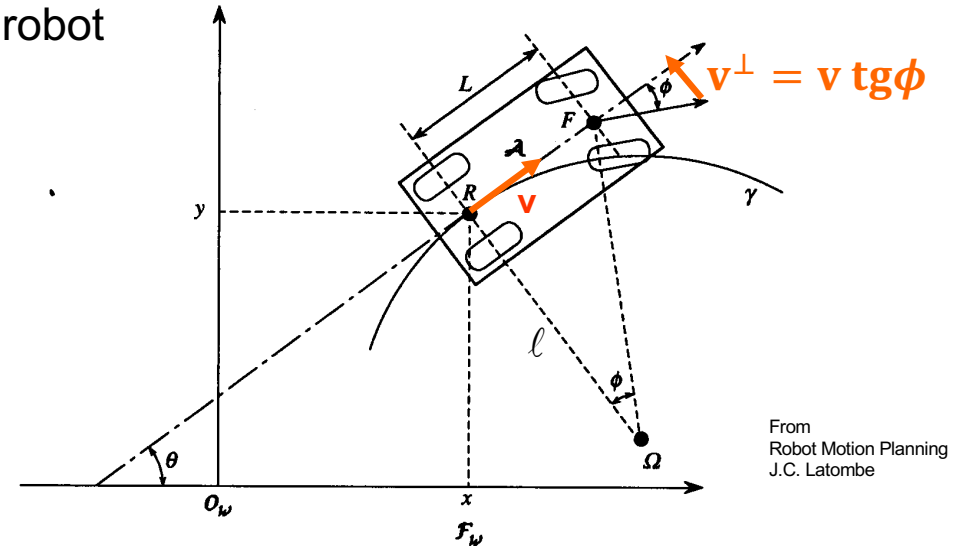
$$C' = (\theta_1, \theta_2)$$

$$\dim C' = \dim C - 4 = 2$$

Kinematic Constraints
Non-Holonomic Constraints

Nonholonomic Constraints – Example Car-like robot

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \end{cases}$$



From
Robot Motion Planning
J.C. Latombe

- $q = (x, y, \theta)$ – configuration (position + orientation)
- ϕ – steering angle
- C (configuration space) has dimension 3

If there is no slippage, the velocity of point R has to point along the main axis of **A**

$$\frac{dy}{dx} = \tan\theta \quad \rightarrow \quad -\sin\theta dx + \cos\theta dy = 0$$

Constraint cannot be written in the form
$F(q_1, q_2, \ldots, q_m, t) = 0$

Nonholonomic constraint

Kinematic Constraints

Non-Holonomic Constraints

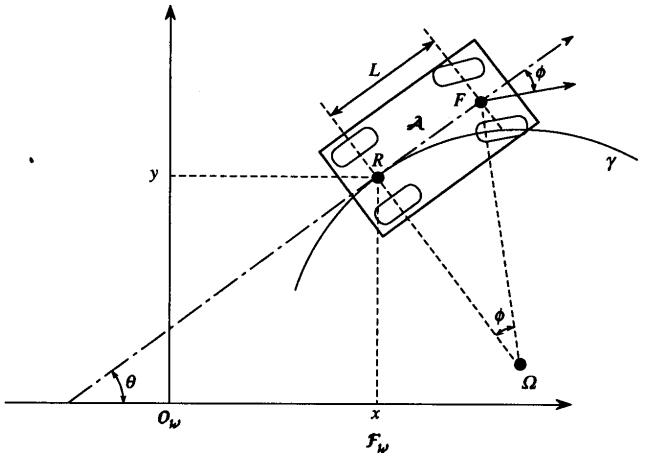
$$-\sin\theta\, dx + \cos\theta\, dy = 0$$

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$

$$\begin{cases} \dot{\theta} = \frac{v}{R} \\ \tan\phi = \frac{L}{R} \end{cases} \Rightarrow \quad \dot{\theta} = v\frac{\tan\phi}{L}$$

Curvature of the path
(inverse of the turning radius)

$$l = \frac{1}{R} = \frac{\tan\theta}{L}$$



The space of differential motions (dx, dy, dθ) of the robot at any configuration is a two-dimensional space.

If the robot was a free-flying object this space would be three-dimensional.

$$\phi \in [-\phi_{max}, \phi_{max}]$$
$$\phi_{max} < \pi/2$$
$$\rightarrow\ |\phi| \le \phi_{max} < \pi/2$$

Limited steering angle : additional **holonomic** constraint

$$l_{max} = \frac{\tan\phi_{max}}{L}$$

$$|\dot{\theta}| \le |v| l_{max} \rightarrow \dot{x}^2 + \dot{y}^2 - \frac{\dot{\theta}^2}{l_{max}} \ge 0 \rightarrow F\left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{d\theta}{dt}\right) \ge 0$$

Kinematic Constraints
Non-Holonomic Constraints

Nonholonomic Constraints

- Non-integrable equation involving the configuration parameters and their derivatives (velocity parameters)
- Nonholonomic constraints
  - **do not reduce** the dimension of the configuration space attainable by the robot
  - **reduce** the dimension of the possible differential motions (i.e., the space of the velocity directions) at any given configuration

$$F(q, \dot{q}, t) = 0$$

$$F(q_1, q_2, \ldots, q_m, \dot{q}_1, \dot{q}_2, \ldots, \dot{q}_m, t) = 0$$

F is a smooth function with non-zero derivative

- If F is integrable – the equation can be written as a holonomic constraint
- If F is non-integrable – the constraint is nonholonomic

- dim of **C** does not change
- dim (velocity or differential motion space) = dim C – nº of independent restrictions

Kinematic Constraints

Non-Holonomic Constraints

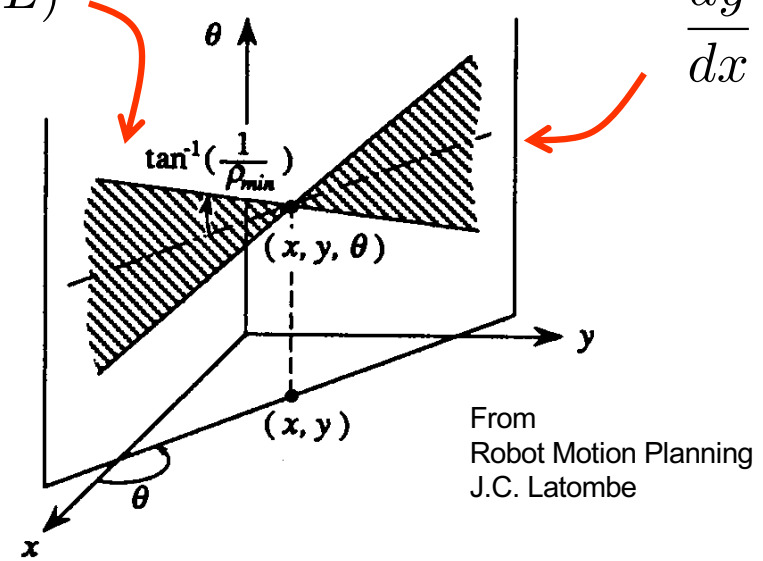
Holonomic Constraints - Example Car-like robot with minimum turning radius

$$\begin{cases} -\dot{x}\sin\theta + \dot{y}\cos\theta = 0 \\ \dot{x}^2 + \dot{y}^2 - \dfrac{\dot{\theta}^2}{l_{\max}} \geq 0 \end{cases}$$

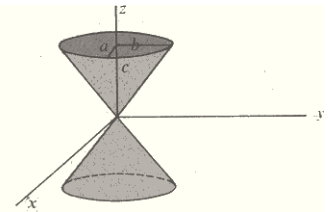
Velocity vector $(\dot{x}, \dot{y}, \dot{\theta})$ satisfies this inequality due to a limited turning angle

$$\phi_{\max} = \arctan(l_{\max} L)$$

$$\frac{dy}{dx} = \tan\theta$$



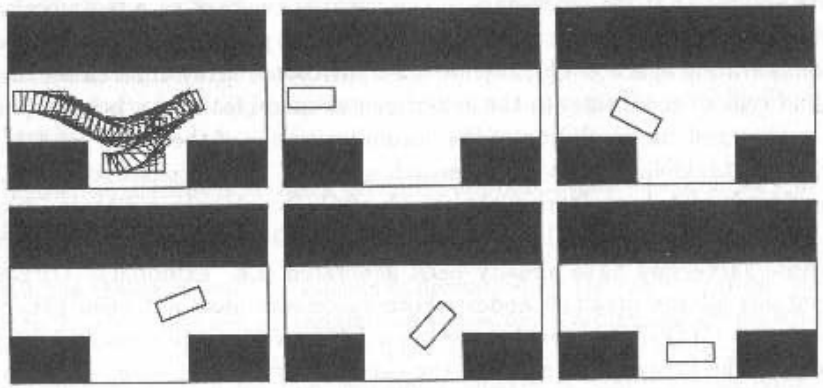
From
Robot Motion Planning
J.C. Latombe



$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0$$

Path Planning for Non-Holonomic Vehicles

Using Search Algorithm



parallel parking ($|\Phi_{max}| < 30°$)

in (Latombe,1991)

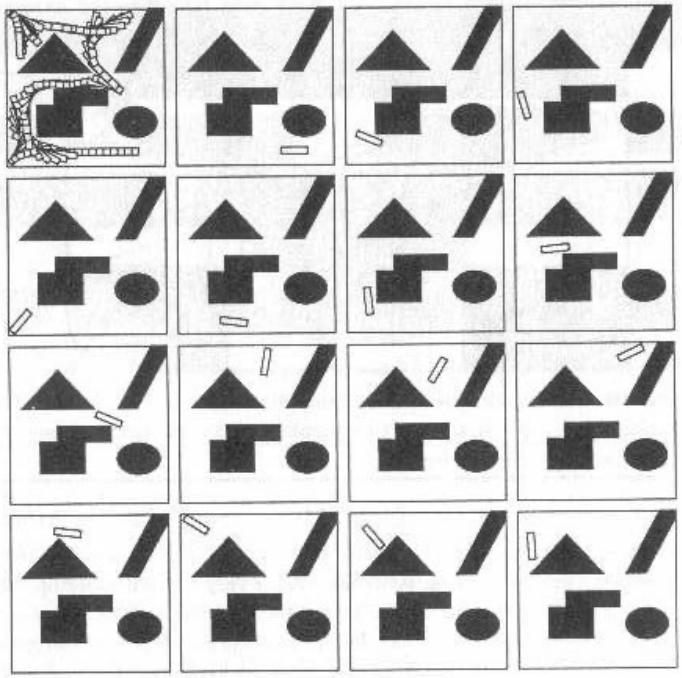
tricycle-like vehicle

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \frac{v}{L}\tan\Phi$$

A* search algorithm where each node generates at most 6 successors, i.e., postures which result from applying adequate pairs

$$(v,\Phi) \in \{-v_0, v_0\} \times \{-\Phi_{max}, 0, \Phi_{max}\}$$

cluttered workspace ($|\Phi_{max}| < 45°$)



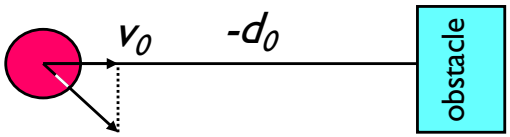
in (Latombe,1991)

Path Planning for Non-Holonomic Vehicles

Generalized Potential Fields

Repulsive potential takes into account the robot non-holonomy

(Krogh, 1984)



$$\begin{cases} v(t) &= v_0 - at \\ d(t) &= -d_0 + v_0 t - \frac{1}{2} a t^2 \end{cases}$$

$$\begin{cases} v(T) = 0 \\ a \leq \beta \end{cases} \Rightarrow T \geq T_{\min} = \frac{v_0}{\beta}$$

minimum time to stop at maximum de-acceleration

$$\begin{cases} v(T_{\max}) = 0 \\ d(T_{\max}) = 0 \end{cases} \Leftrightarrow \begin{cases} a = \frac{v_0^2}{2d_0} \\ T_{\max} = \frac{2d_0}{v_0} \end{cases}$$

maximum time that the robot may take to collide with obstacle, with constant de-acceleration

$$U_{\text{rep}}(v) = \frac{1}{T_{\max} - T_{\min}} = \begin{cases} \frac{\beta v_0}{2d_0\beta - v_0^2} & \text{if } v_0 > 0 \\ 0 & \text{if } v_0 \geq 0 \end{cases}$$

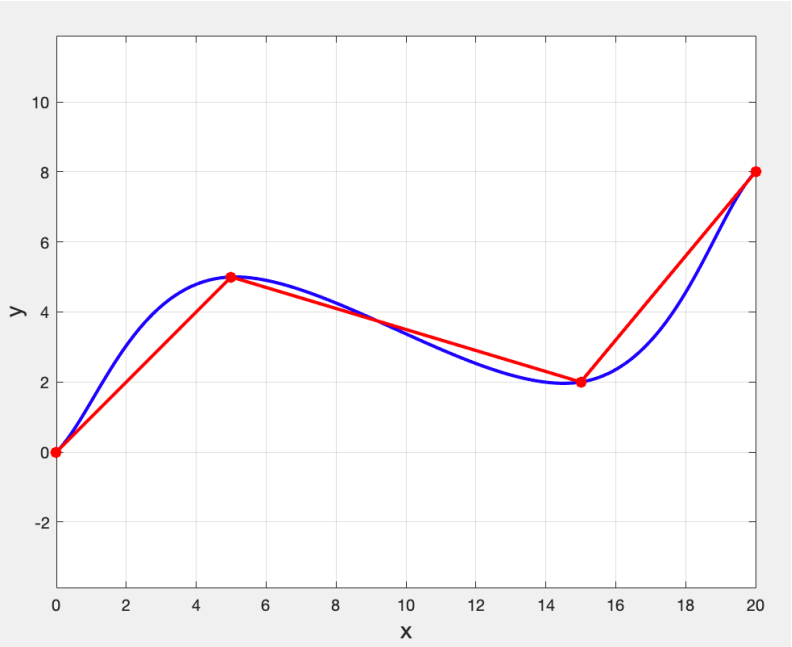
Path Planning Followed by Trajectory Optimization

Adapted from “VNAV course”, by Luca Carlone

Decoupled approach

1. Fast path planning with no kinematic constraints generates collision-free waypoints (e.g. RRT)
2. Trajectory optimization generates a ”smooth” polynomial trajectory

Example: generate trajectory that passes through 4 waypoints $[\bar{p}_0, \bar{p}_1, \bar{p}_2, \bar{p}_3]$



- Infinite dimensional optimization problem

$$\min_{p(t)} \int_{t_0}^{t_f} J(p(t), \dot{p}(t), \ldots) dt$$

$$\text{subject to } p(t_0) = \bar{p}_0, \ldots, p(t_3) = \bar{p}_3$$

$$\dot{p}(t_0) = \dot{\bar{p}}_0, \ldots, \dot{p}(t_3) = \dot{\bar{p}}_3$$

- Assume that p(t) has 3 segments described by polynomials → optimize the coefficients.

Path Planning Followed by Trajectory Optimization

Adapted from “VNAV course”, by Luca Carlone

Example: generate ”smooth” trajectory that passes through four waypoints

$$[\bar{p}_0, \bar{p}_1, \bar{p}_2, \bar{p}_3]$$

- 4 waypoints → 3 segments

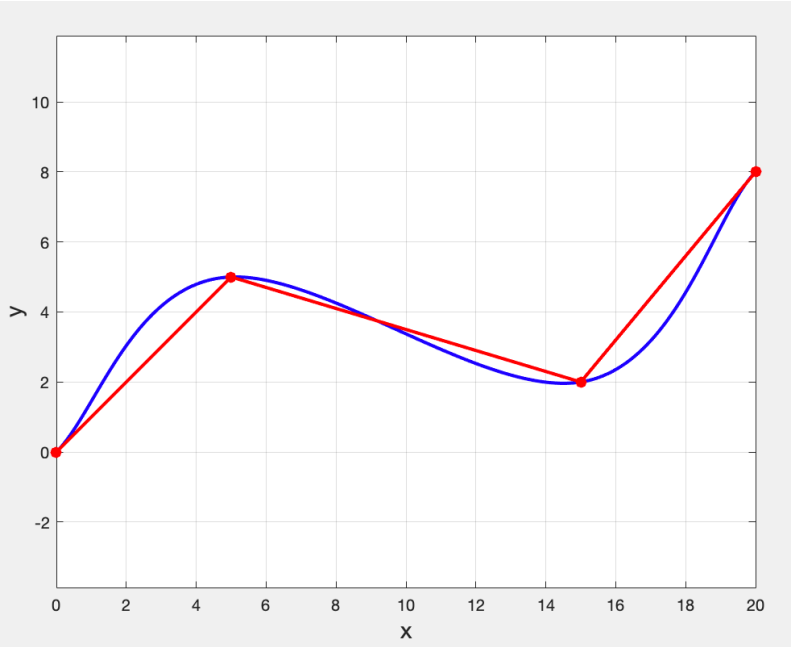
$$p(t) = \begin{cases} p_1(t), t_0 \leq t \leq t_1 \\ p_2(t), t_1 \leq t \leq t_2 \\ p_3(t), t_2 \leq t \leq t_3 \end{cases}$$

- polynomials of order 3 →

  minimum acceleration trajectories

- 3 polynomials of order 3 → 12 coefficients

$$p_i(t) = a_{0i} + a_{1i}t + a_{2i}t^2 + a_{3i}t^3 \quad \text{(per coordinate)}$$
$$\dot{p}_i(t) = a_{1i} + 2a_{2i}t + 3a_{3i}t^2$$
$$\ddot{p}_i(t) = 2a_{2i} + 6a_{3i}t$$



- 12 constraints → unique solution

Path Planning Followed by Trajectory Optimization

Adapted from “VNAV course”, by Luca Carlone

Example: generate ”smooth” trajectory that passes through four waypoints

$$[\bar{p}_0, \bar{p}_1, \bar{p}_2, \bar{p}_3]$$

- 12 coefficients and 12 constraints → unique solution

Waypoints (6)

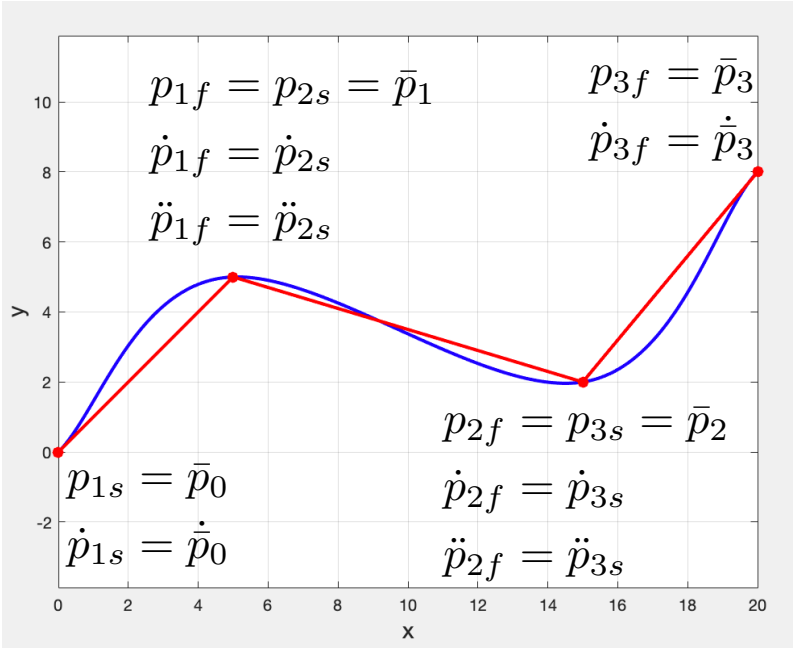
$$p_{is} = \bar{p}_{i-1}, \; i \in \{1,2,3\}$$
$$p_{jf} = \bar{p}_j, \quad j \in \{1,2,3\}$$

Initial and final velocity (2)

$$\dot{p}_{1s} = \dot{\bar{p}}_0$$
$$\dot{p}_{3f} = \dot{\bar{p}}_3$$

Intermediate velocities and accelerations (4)

$$\dot{p}_{if} = \dot{p}_{(i+1)s}$$
$$\ddot{p}_{if} = \ddot{p}_{(i+1)s}, i \in \{1,2\}$$



All constraints are linear on the coefficients,

e.g. $p_{1s} = \bar{p}_0 \Leftrightarrow \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \end{bmatrix} \begin{bmatrix} a_{00} \\ a_{10} \\ a_{12} \\ a_{13} \end{bmatrix} = \bar{p}_0$, such that $\mathbf{a} = M^{-1}\mathbf{b}$

Path Planning Followed by Trajectory Optimization

Adapted from “VNAV course”, by Luca Carlone

Example: generate ”smooth” trajectory that passing through four waypoints

$$[\bar{p}_0, \bar{p}_1, \bar{p}_2, \bar{p}_3]$$

- 12 coefficients and 12 constraints → unique solution

$$\mathbf{a} = M^{-1}\mathbf{b}$$

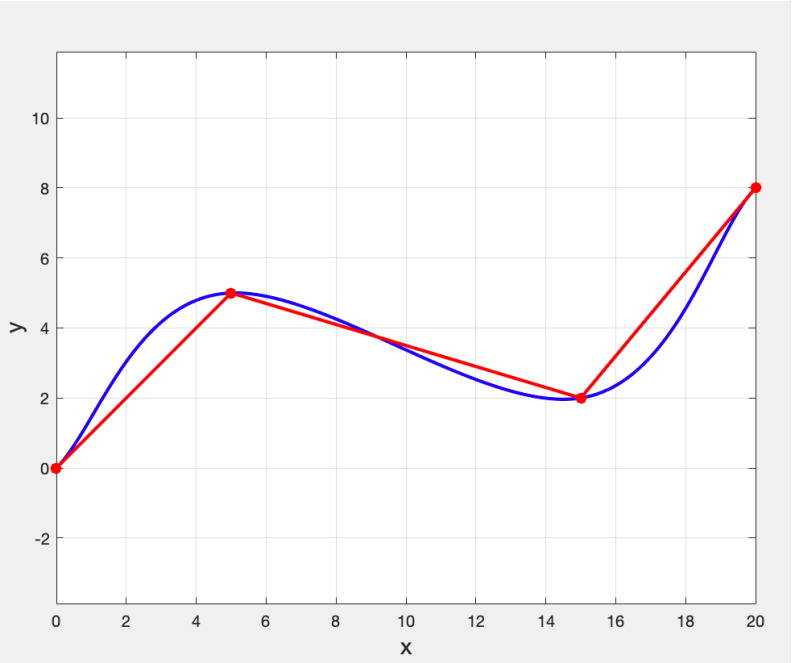
- In general (with more coefficients) →

optimization problem

$$\min_{p(t)} \int_{t_0}^{t_f} J(p(t), \dot{p}(t), \ldots) dt$$

$$\text{subject to } p(t_0) = \bar{p}_0, \ldots, p(t_3) = \bar{p}_3$$

$$\dot{p}(t_0) = \dot{\bar{p}}_0, \ldots, \dot{p}(t_3) = \dot{\bar{p}}_3$$



written as a QP

$$\min_{\mathbf{a}} \mathbf{a}^T Q \mathbf{a}$$

$$\text{subject to } M\mathbf{a} = \mathbf{b}$$

Motion Planning: Extensions

Extensions of the Basic Problem

- Multiple moving objects
  - mobile obstacles (e.g., opponents in robotic soccer) - *include time as a new dimension of C*
  - multiple robots sharing the same workspace (e.g., teammates in robotic soccer) – *planning in* $A_1 \times A_2 \times \ldots \times A_p$
  - articulated robots – *e.g., manipulator path planning*
- Uncertainty (e.g., errors in the measured distance to obstacles)
- Movable objects (e.g., the ball in robot soccer)
- Kinematic constraints – holonomic *vs* non-holonomic

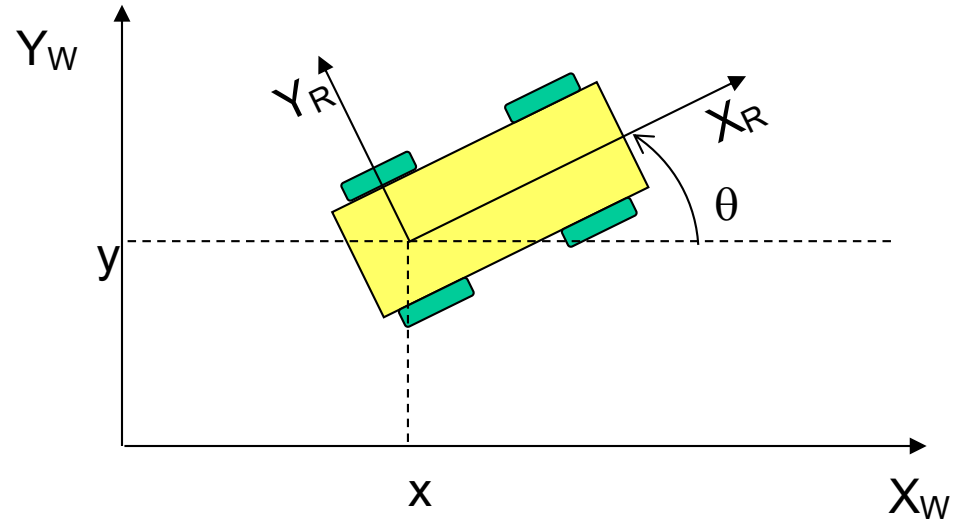
Mobile Robots Kinematics

- What is a **kinematic** model ?
- What is a **dynamic** model ?
- Which is the difference between kinematics and dynamics?

- **Locomotion** is the process of causing an autonomous robot to move.
  - In order to produce motion, forces and torques must be applied to the vehicle

- **Dynamics** – the study of motion equations in which these forces and torques are modeled
- **Kinematics** – study of motion equations withouth considering the forces that affect the motion.
  - Deals with the geometric relationships that govern the system
  - Deals with the relationship between control parameters and the beahvior of a system in state space.

Notation



$\{X_R, Y_R\}$ – moving frame

$\{X_W, Y_W\}$ – world (reference) frame

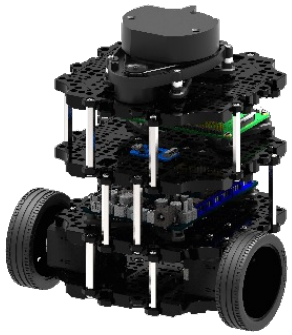
$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

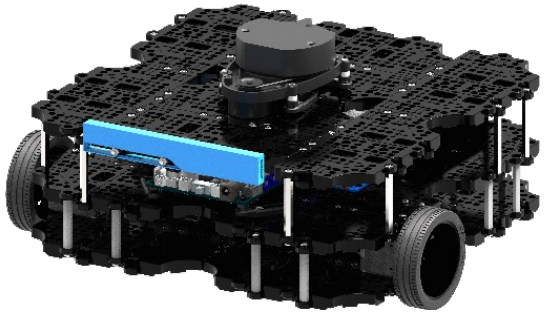
Robot posture/pose in world frame

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation matrix expressing the orientation of the robot frame with respect to the world frame

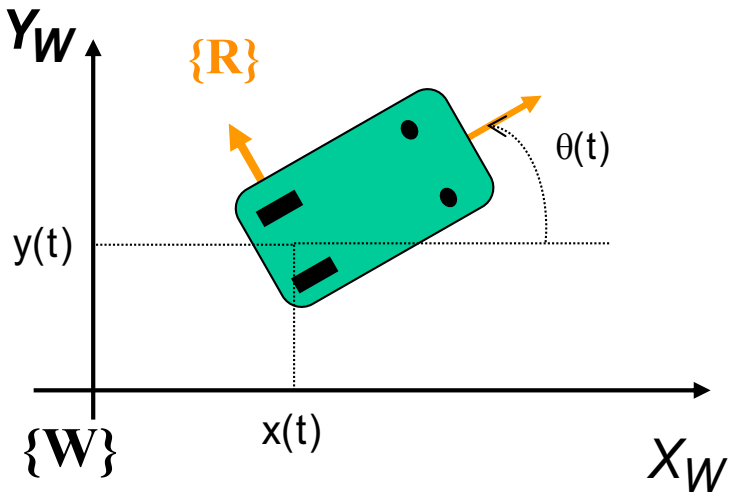
Differential Drive Kinematics





a.k.a. Unicycle

Differential Drive Kinematics



Kinematics in the world frame

$$\begin{cases} \dot{p}(t) = R(\theta(t))v(t) \\ \dot{\theta}(t) = \omega(t) \end{cases}$$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$v = \begin{bmatrix} v_x \\ 0 \end{bmatrix}$$

Kinematics in the world frame

$$\begin{cases} \dot{x} = \cos(\theta)v_x \\ \dot{y} = \sin(\theta)v_x \\ \dot{\theta} = \omega \end{cases}$$

Kinematics in the body frame

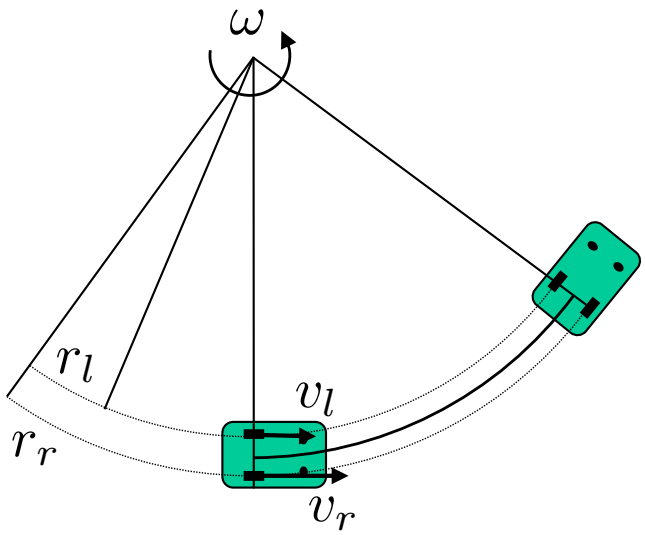
$$v = \begin{bmatrix} v_x \\ 0 \end{bmatrix}$$

$$\dot{\theta} = \omega$$

$$\begin{cases} v_x = ? \\ \omega = ? \end{cases}$$

Differential Drive Kinematics

Instantaneous radius of rotation $r = \dfrac{v_x}{\omega}$



$$\begin{cases} v_r + v_l = \omega(r_r + r_l) = 2v_x \\ v_r - v_l = \omega(r_r - r_l) = \omega L \end{cases}$$

$$\Rightarrow \begin{cases} v_x = \frac{1}{2}(v_r + v_l) = \frac{R}{2}(\omega_r + \omega_l) \\ \omega = \frac{1}{L}(v_r - v_l) = \frac{R}{L}(\omega_r - \omega_l) \end{cases}$$

Also valid when $\omega = 0,\ r = \infty$

$$\begin{cases} r = \frac{1}{2}(r_r + r_l) \\ L = r_r - r_l \end{cases}$$

$$\begin{cases} v_x = \omega r \\ v_r = \omega r_r \\ v_l = \omega r_l \end{cases}$$

Kinematics in the world frame

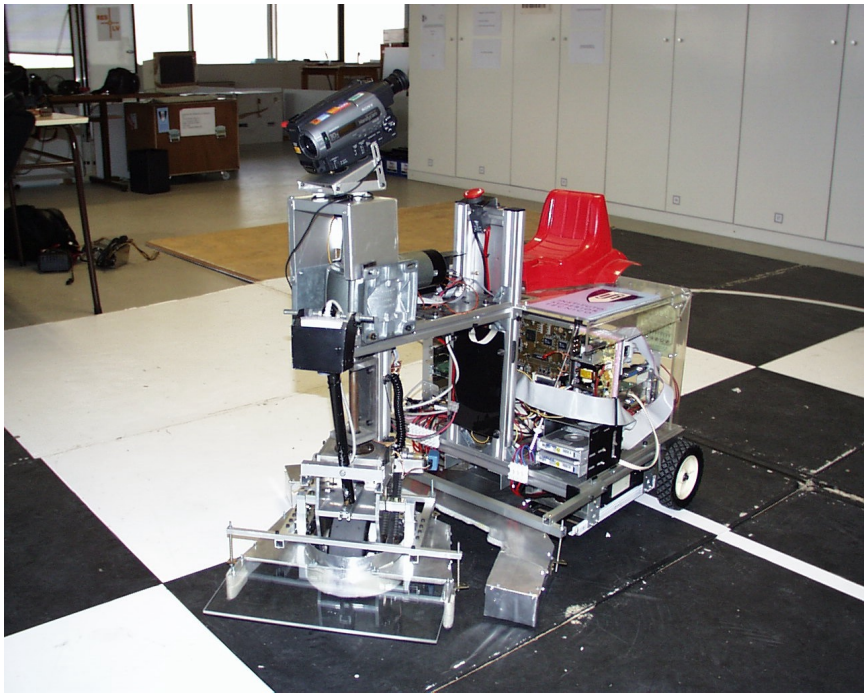
$$\begin{cases} \dot{p}(t) = R(\theta(t))v(t) \\ \dot{\theta}(t) = \omega(t) \end{cases}$$

Velocities in the body frame

$$v = \begin{bmatrix} \frac{R}{2}(\omega_r + \omega_l) \\ 0 \end{bmatrix}$$

$$\omega = \frac{R}{L}(\omega_r - \omega_l)$$

Tricycle-like Kinematics

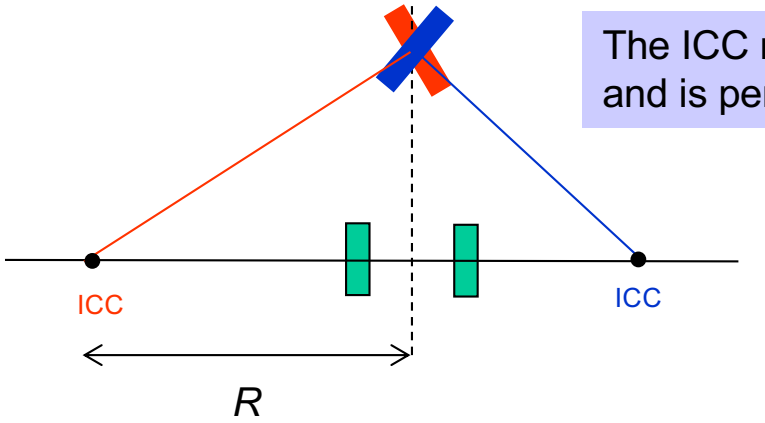


Tricycle-like Kinematics

- Three wheels and odometers on the two rear wheels
- Steer and drive are provided through the front wheel

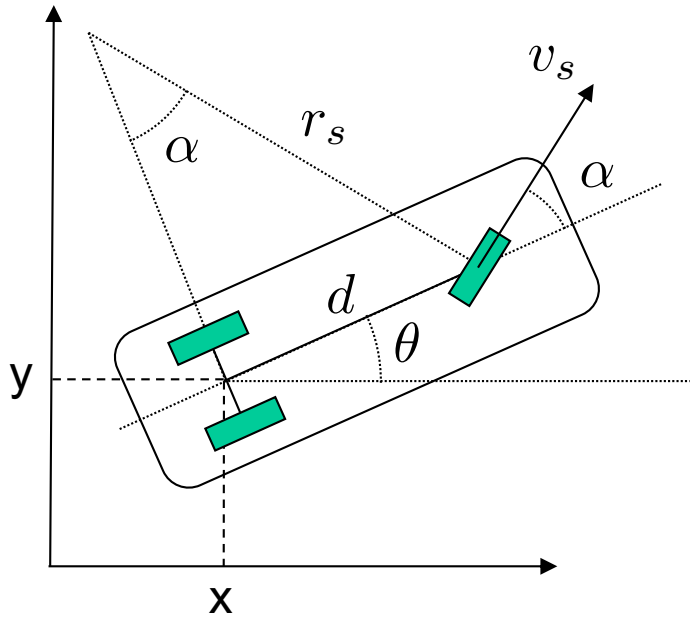
- control variables:
  - steering direction $\alpha(t)$
  - velocity of steering wheel $v_s(t)$

ICC: Instantaneous Center of Curvature



The ICC must lie on the line that passes through, and is perpendicular to, the fixed rear wheels

Tricycle-like Kinematics



If the steering wheel is set to an angle $\alpha(t)$ from the straight-line direction, the tricycle will rotate with angular velocity w(t) about a point lying a distance r_s along the line perpendicular to, and passing through, the front wheel.

$$\omega = \frac{v_s}{r_s}$$

angular velocity of the robot frame with respect to the world frame

$$\sin\alpha = \frac{d}{r_s}$$

$$\omega = \frac{v_s}{d}\sin\alpha$$

Kinematics in the world frame

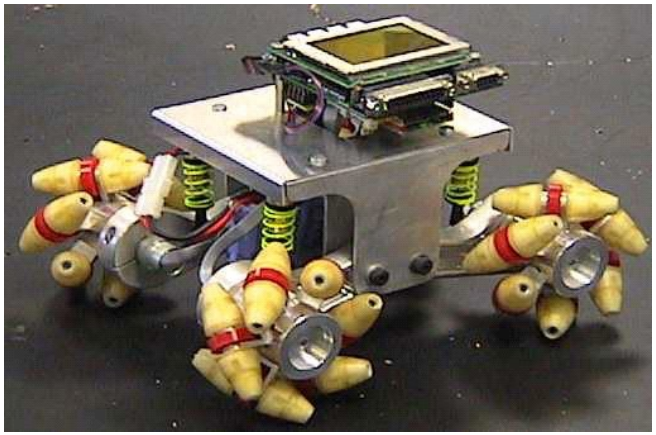
$$\begin{cases} \dot{p}(t) = R(\theta(t))v(t) \\ \dot{\theta}(t) = \omega(t) \end{cases}$$

Velocities in the body frame

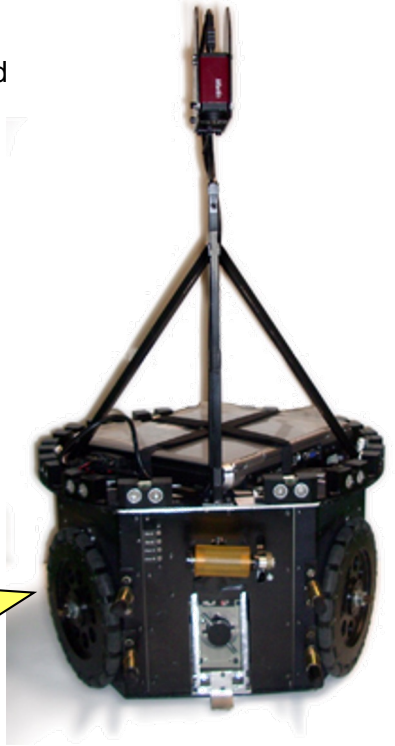
$$v = \begin{bmatrix} v_s\cos\alpha \\ 0 \end{bmatrix}$$

$$\omega = \frac{v_s}{d}\sin\alpha$$

Omnidirectional Kinematics

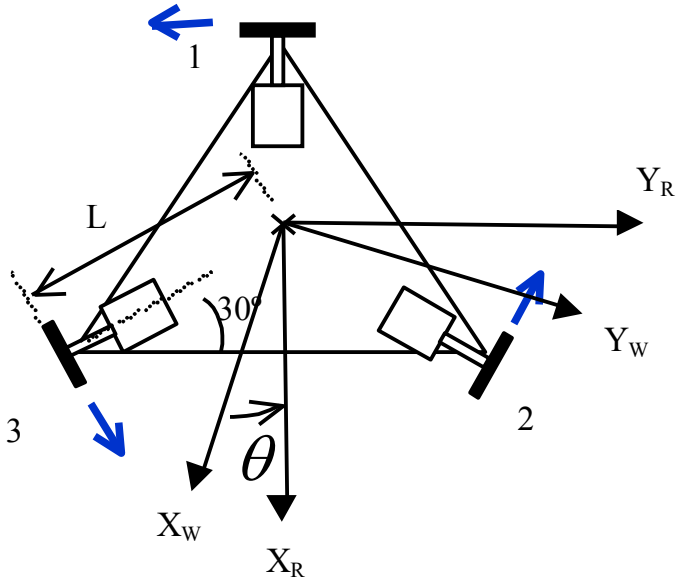


EyeBot Omni-Directional Vehicles. Designed by Gordon Menck, Richard Meager and Thomas Braunl.



Swedish wheel

Omnidirectional Kinematics



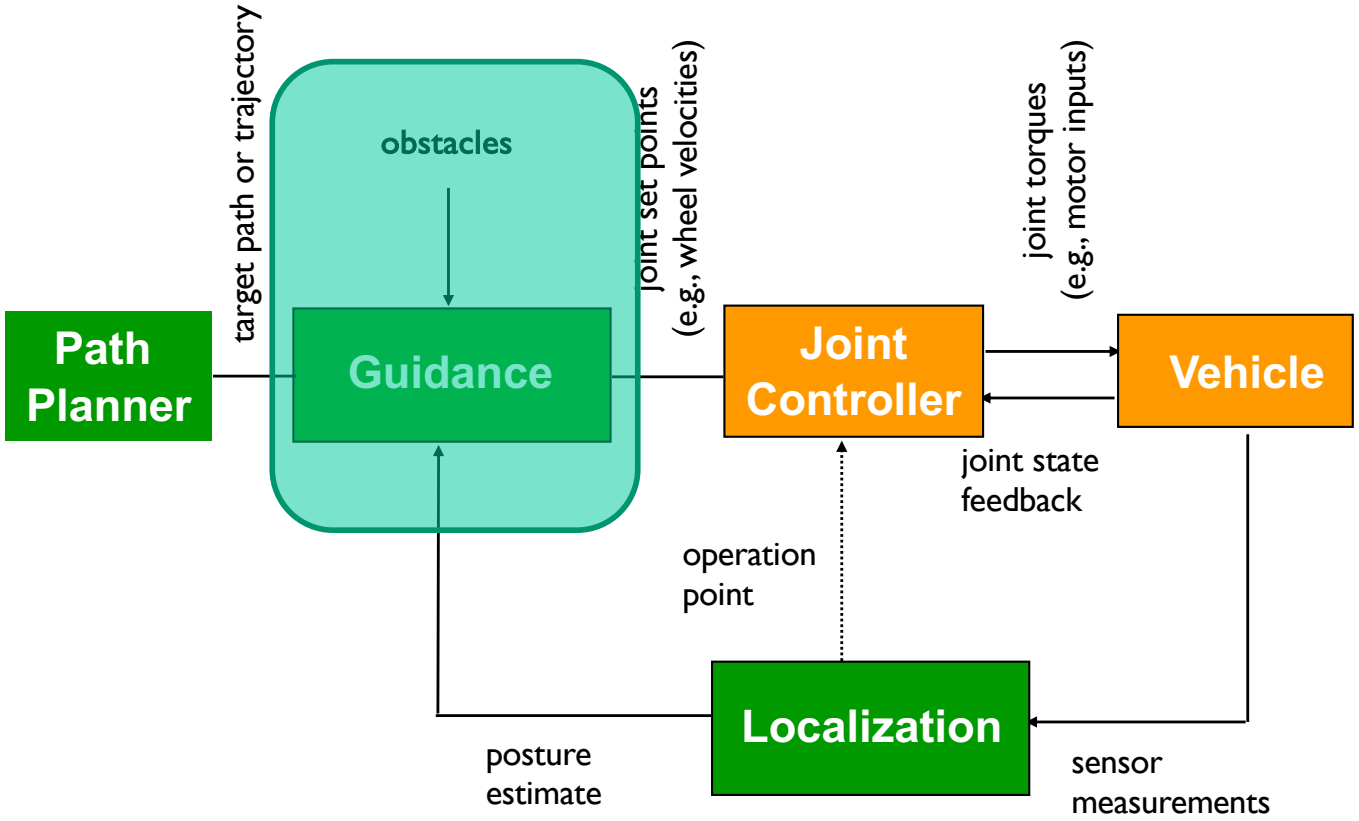
Kinematic model in the robot frame

$$\begin{bmatrix} V_x \\ V_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{\sqrt{3}}r & \frac{1}{\sqrt{3}}r \\ -\frac{2}{3}r & \frac{1}{3}r & \frac{1}{3}r \\ \frac{r}{3L} & \frac{r}{3L} & \frac{r}{3L} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Control variables

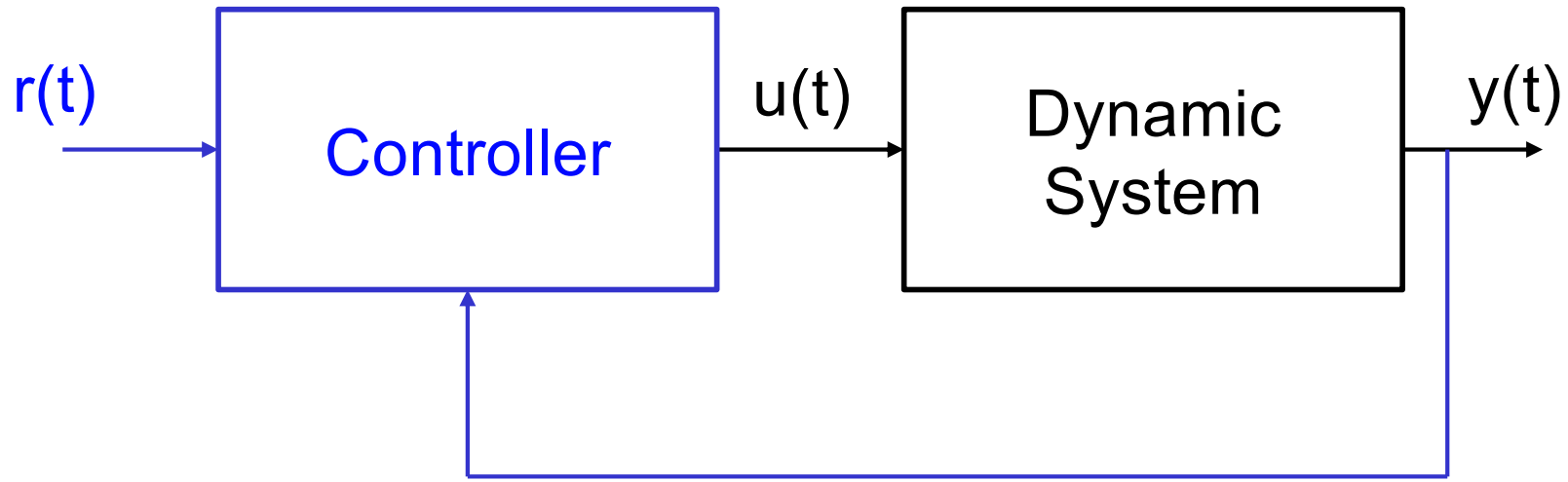
w_1, w_2, w_3 – angular velocities of the three swedish wheels

The Navigation Loop



Closed Loop Control

Input/Output Model



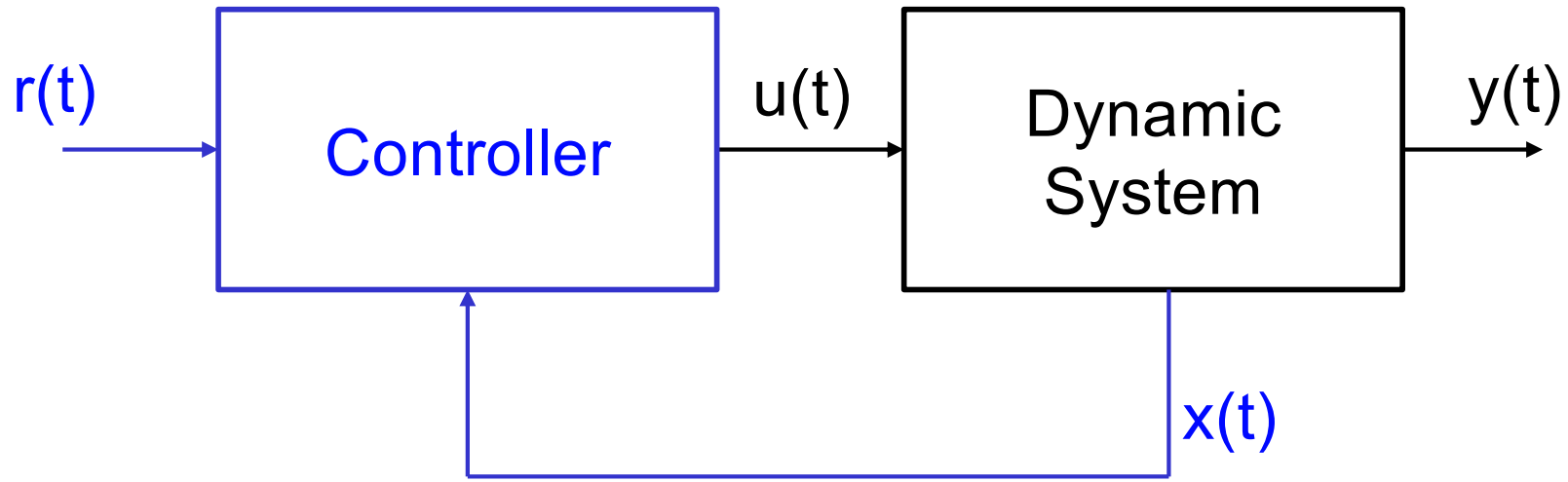
in general: $y(t) = f(u(t))$

linear case:

$$a_n \frac{d^n}{dt} y(t) + a_{n-1} \frac{d^{n-1}}{dt} y(t) + \cdots + a_1 \frac{d}{dt} y(t) + a_0 y(t) = b_m \frac{d^m}{dt} u(t) + b_{m-1} \frac{d^{m-1}}{dt} u(t) + \cdots + b_1 \frac{d}{dt} u(t) + b_0 u(t)$$

Closed Loop Control

State Space Model



in general:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{g}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)$$
$$\mathrm{y}(t) = \boldsymbol{h}(\boldsymbol{x}(t)\,)$$

linear case:

$$\dot{\boldsymbol{x}}(t) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\boldsymbol{x}(t)$$

$\boldsymbol{x}$ is n X 1

Mobile Robot Guidance

Guidance is a closed loop control problem where the mobile robot control variables are controlled to make it follow a reference trajectory

Three distintic problems:

- Virtual Vehicle Tracking / Trajectory Tracking / Posture Tracking
- Path Following
- Posture Stabilization

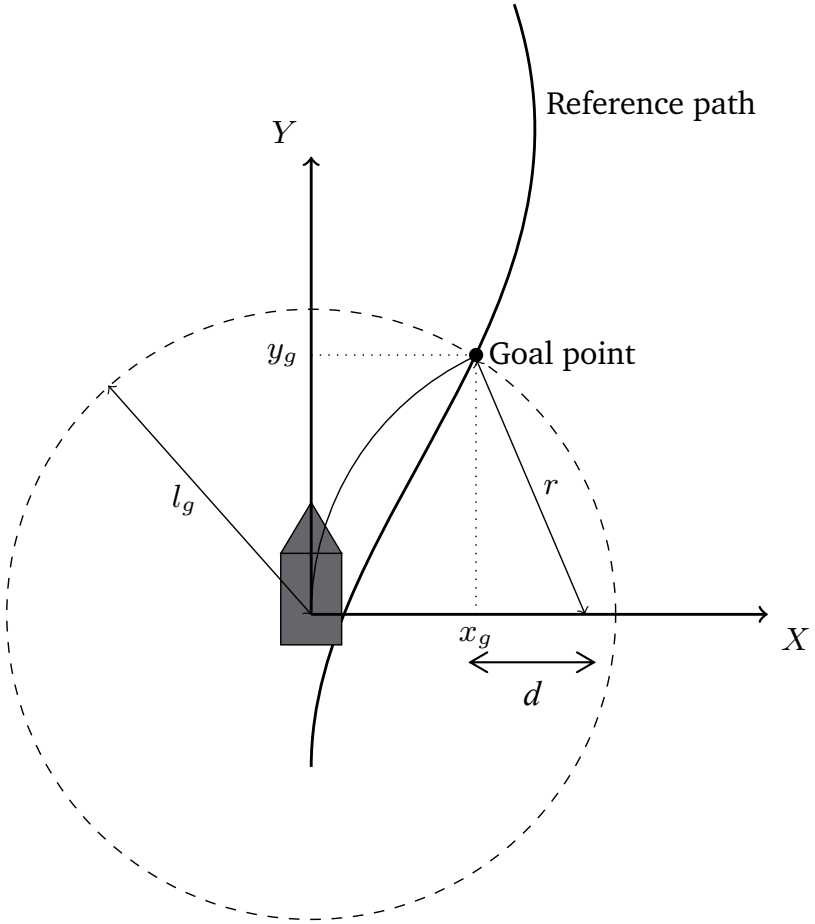
Guidance / Local Planning algorithms for Path Following:

- Pure-pursuit
- Dynamic Window Approach
- ...

Pure-Pursuit Path Following

References

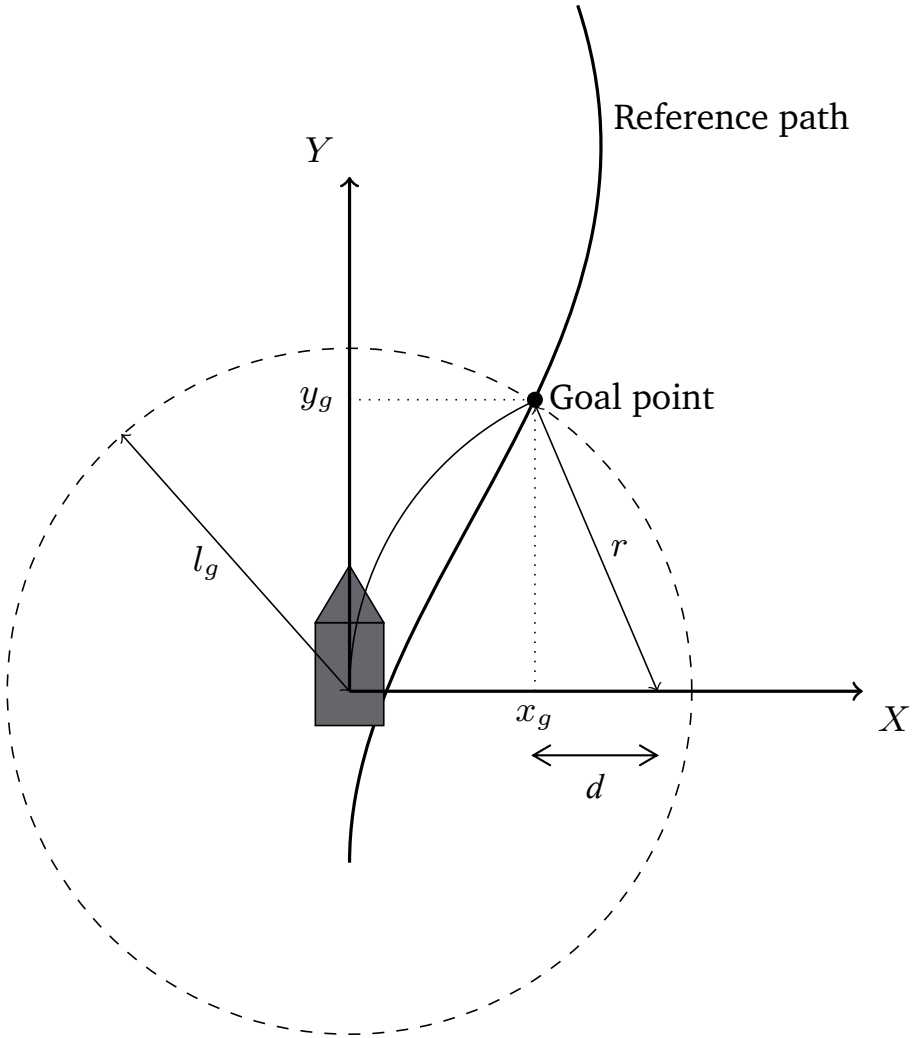
- Coulter, R. Implementation of the pure-pursuit path tracking algorithm. Technical report, Carnegie Mellon University, 1992.
- Wallace, R. S., Stentz, A., Thorpe, C. E., Moravec, H. P., Whittaker, W., and Kanade, T. First results in robot road-following. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 1089–1095. Citeseer, 1985.



inspired by the way humans drive:

- focus on a goal position (x_g, y_g) ahead on the path
- user-defined look-ahead distance l_g
- chasing the point on the path, i.e., *pursuing* the point

Pure-Pursuit Path Following



$$x_g^2 + y_g^2 = l_g^2$$

$$d^2 + y_g^2 = r^2$$

$$x_g + d = r$$

$$(r - x_g)^2 + y_g^2 = r^2$$

$$r^2 - 2rx_g + x_g^2 + y_g^2 = r^2$$

$$\boxed{r = \frac{l_g^2}{2x_g}}$$

Mobile Robot Guidance
Pure-Pursuit Path Following

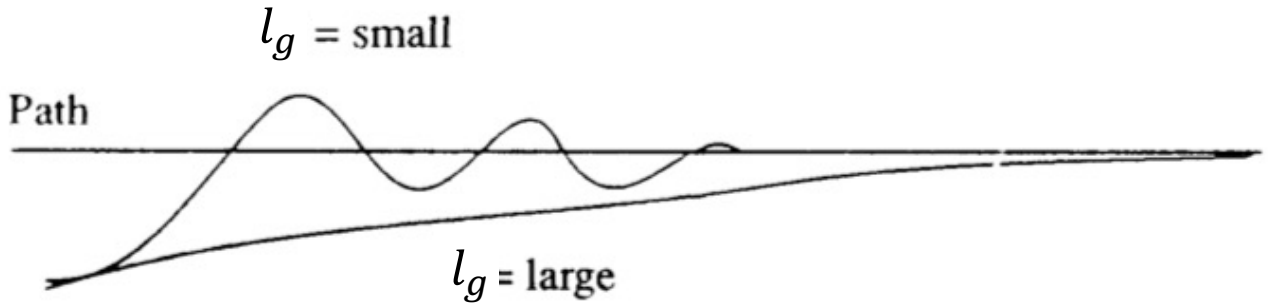
Pure-Pursuit Algorithm

1. Determine the current location of the vehice
2. Find the path point closest to the vehicle, one lookahead distance of the vehicle
3. Determine the goal point (x_g, y_g)
4. Represent the goal point in the vehicle frame {*R*}
5. Calculate the curvature ($= 1/r$) and make the vehicle set its steering to that curvature, i.e., make $v_y^R = wr$
6. Update the vehicle's position and go to step 2.

Mobile Robot Guidance
Pure-Pursuit Path Following

Pure-Pursuit Algorithm Properties

1. Regaining a path far-away from the vehicle



2. Maintaining the vehicle on the path
   - if the lookahead distance is too long, low-radius *r* (high-curvature) arcs can not be followed

TÉCNICO LISBOA

Mobile Robot Local Planning

Dynamic Window Approach

Motion Generation for mobile robots:

- planning algorithms: used to plan a path towards a goal position in known static environments (global approach, e.g., roadmap methods)
  - can be computed offline, but are slow and assume static environment
- real-time obstacle avoidance methods: reactive motion behavior in dynamic and unknown environments (local approach, e.g., potential field, bug algorithm, methods)
  - only need local (sensed) information, react fast, but may have local minima

References

- Fox, Dieter, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance." IEEE Robotics & Automation Magazine 4.1 (1997): 23-33.
- Brock, O., & Khatib, O. (1999). High-speed navigation using the global dynamic window approach. In Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on (Vol. 1, pp. 341-346). IEEE.

Mobile Robot Local Planning

Dynamic Window Approach

DWA summary description:

- algorithm considers only a short-time interval when computing the next steering command, and does it periodically
- during that time interval, trajectories are approximated by circular curvatures, resulting in a two-dimensional search space of translational and rotational velocities
- search space is constrained to
  - the admissible velocities allowing the robot to stop safely
  - velocities that can be reached within the next time interval (due to robot acceleration constraints)
- velocities form the *dynamic window,* which is centered around the current velocities of the robot in the velocity space

Mobile Robot Local Planning
Dynamic Window Approach

DWA algorithm components

reprinted from (Fox et al, 1997)

1. **Search space:** The search space of the possible velocities is reduced in three steps:

   (a) **Circular trajectories:** The dynamic window approach considers only circular trajectories (curvatures) uniquely determined by pairs (v, ω) of translational and rotational velocities. This results in a two-dimensional velocity search space.

   (b) **Admissible velocities:** The restriction to admissible velocities ensures that only safe trajectories are considered. A pair (v, ω) is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the corresponding curvature.

   (c) **Dynamic window:** The dynamic window restricts the admissible velocities to those that can be reached within a short time interval given the limited accelerations of the robot.

Mobile Robot Local Planning
Dynamic Window Approach

DWA algorithm components

2. **Optimization:** The objective function

$$G(v,\omega) = \sigma(\alpha \cdot \text{heading}(v,\omega) + \beta \cdot \text{dist}(v,\omega) + \gamma \cdot \text{vel}(v,\omega)) \quad (13)$$

is maximized. With respect to the current position and orientation of the robot this function trades off the following aspects:

(a) **Target heading:** *heading* is a measure of progress towards the goal location. It is maximal if the robot moves directly towards the target.

(b) **Clearance:** *dist* is the distance to the closest obstacle on the trajectory. The smaller the distance to an obstacle the higher is the robot's desire to move around it.

(c) **Velocity:** *vel* is the forward velocity of the robot and supports fast movements.

The function σ smoothes the weighted sum of the three components and results in more side-clearance from obstacles.

Mobile Robot Local Planning
Dynamic Window Approach

Search space

V_s space of possible velocities (circular arcs)

Each curvature is uniquely determined by (v_i, w_i), defined as the *velocity*

to generate a trajectory point for the next *n* time intervals the robot has to determine velocities (v_i, w_i) for each of the n intervals between t_0 and t_n, under the constraint of no intersection with obstacles

search space is exponential in the number of time intervals considered!

dynamic window approach

- considers exclusively the first time interval
- assumes that the velocities in the remaining *n-1* time intervals are constant (i.e., assumes zero accelerations in $[t_1, t_n]$)

reduced search space is 2-D and tractable
search is repeated after each time interval
velocities will automatically stay constant if no new commands are given

Mobile Robot Local Planning
Dynamic Window Approach

Search space

V_a set of velocities that allow the robot to stop without colliding with an obstacle

$$V_a = \left\{ (v, \omega) \mid v \leq \sqrt{2 \cdot \text{dist}(v, \omega) \cdot \dot{v}_b} \ \wedge \ \omega \leq \sqrt{2 \cdot \text{dist}(v, \omega) \cdot \dot{\omega}_b} \right\}$$

where

$\dot{v}_b$ linear breakage acceleration
$\dot{w}_b$ angular breakage acceleration

Mobile Robot Local Planning

Dynamic Window Approach

Search space

V_d dynamic window representing the reachable velocities during next time interval

$$V_d = \{(v, \omega) \mid v \epsilon [v_a - \dot{v} \cdot t, v_a + \dot{v} \cdot t] \wedge \omega \epsilon [\omega_a - \dot{\omega} \cdot t, \omega_a + \dot{\omega} \cdot t]\}$$

where

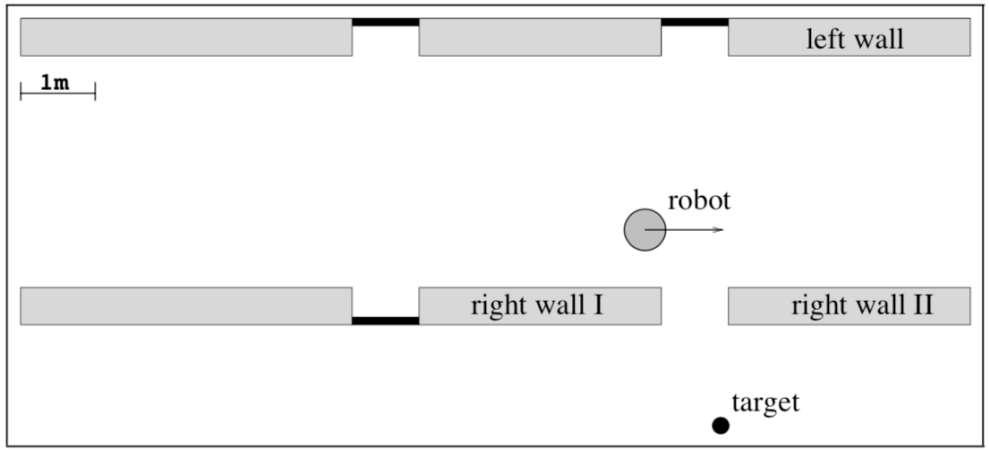
$\dot{v}$	linear acceleration applied during time interval *t*
$\dot{w}$	angular acceleration applied during time interval *t*
(v_a, w_a)	current velocity

Dynamic Window Approach

Resulting Search space $V_r = V_s \cap V_a \cap V_d$

Example

(Fox et al, 1997)



positive w

$\dot{v}_b$ = 50cm/s^2

$\dot{w}_b$ = 60º/s^2