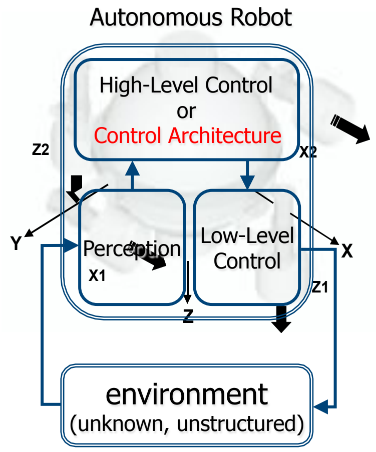
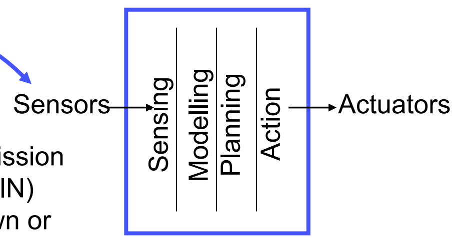
1. Overview of Control Architectures



- Components:
  - **Perception** → sensing + feature extraction + localization
  - **Low-Level Controller** → actuator control to achieve the desired movement in each DOF
  - **High-Level Controller** → To select the best action at each state in order to fulfill the mission

Deliberative architectures

- **Centralized** architecture
- **Sequential** processing
- **Symbolic representation** of the world
- **Hierarchical** division of the mission (Goal= SubGoal1, ..., SubGoalN)
- ✔ Suitable for structured or known or static environments

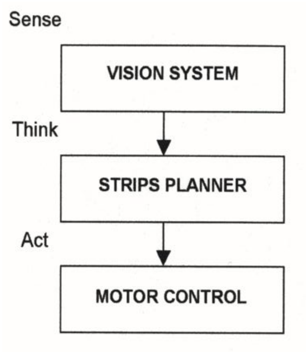


Problems in unstructured or changing environments

- ✗ Symbolic Model of the world (precision and maintenance)
- ✗ Symbolic assignment
- ✗ Real time

UdG **A**utonomous **R**obots

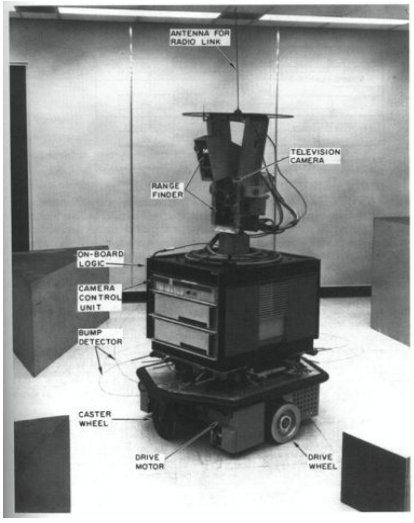
Deliberative architectures



•Shakey (1969), from the Stanford research Institute.

•"sense-think-act" paradigm

•Thinking was accomplished with the STRIPS planner.



UdG **A**utonomous **R**obots

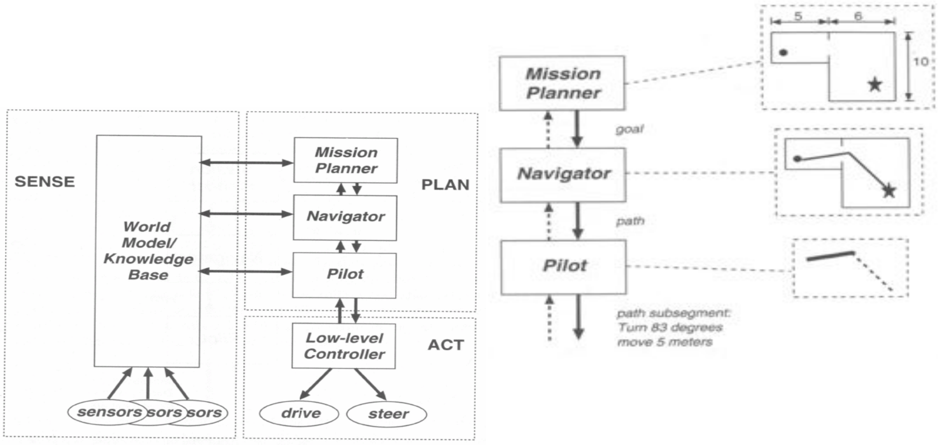
Deliberative architectures

- **World model**:
  - All sensor information is fused into one global data structure
  - It contains:
    - An a priori representation of the environment the robot is operating in
    - Sensing information
    - Any additional cognitive knowledge needed to accomplish the goal
  - Problems:
    - **Closed world assumption**: it contains everything the robot needs to know; there can be no surprises.
    - **Frame problem**: representing a real-world situation in a way that is computationally tractable; which part of the environment must be considered?
- **Task/mission planner**
  - To divide the goal mission in a set of tasks
- **Path planning**
  - To plan a trajectory that accomplishes a task
- **Low-level controller**
  - To execute the trajectory in the real world

UdG Autonomous Robots

Deliberative architectures

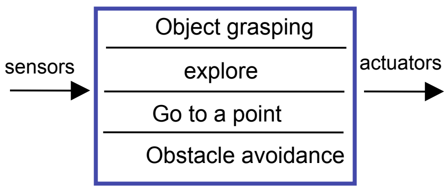
- Nested Hierarchical Controller [Meystel 1990]



UdG Autonomous Robots

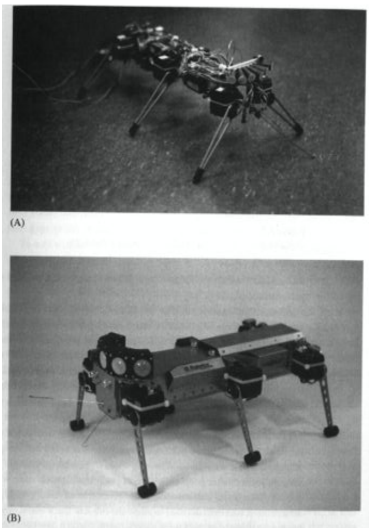
Behaviour-based Architectures

- **Decentralized** Architecture
- Mission division in **simple behaviours**
- **Parallel** processing
- **Reactivity** to the **perceived** environment



- **Advantages**
  - ✔ **No** use of a symbolic model of the world
  - ✔ **Real Time**
  - ✔ Suitable for **changing and unstructured** environments
- **Problems**
  - ✘ Selection and merging of behaviours (maximizing robustness and efficiency)
  - ✘ Decomposition of complex missions

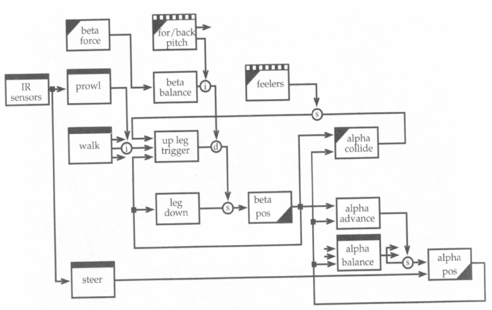
Autonomous Robots — Behaviour-based Architectures



(A)

(B)

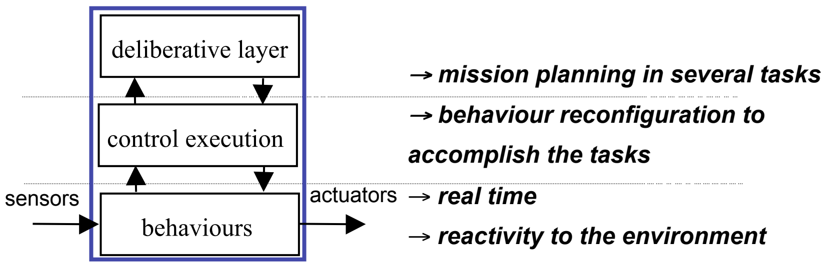
Figure 3.6
(A) Original Genghis. (Photograph courtesy of Rodney Brooks.) (B) Genghis II—a robotic hexapod, commercial successor to the original Genghis. (Photograph courtesy of IS Robotics, Somerville, MA.)



- Genghis robotic hexapod (1989)
- 57 augmented finite state machines implemented the Subsumption control architecture

Autonomous Robots — Hybrid Architectures

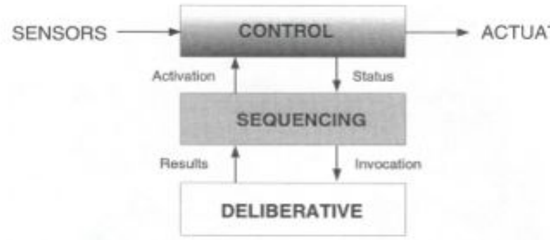
Organization in three layers:

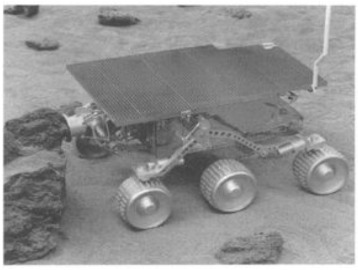


- **Most used architectures**
- ✔ **Advantages from both architecture philosophies**
- ✘ **Disadvantage: more complexity**

Hybrid Architectures

- Atlantis hybrid architecture from Jet Propulsion Laboratory [Gat 1991]

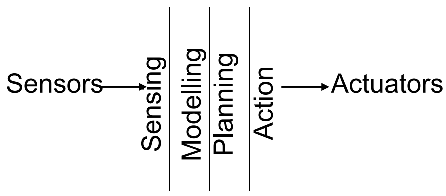




Sojourner, Mars microrovers from NASA

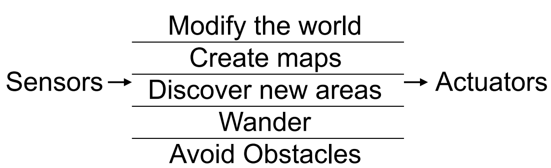
Classification

Deliberative



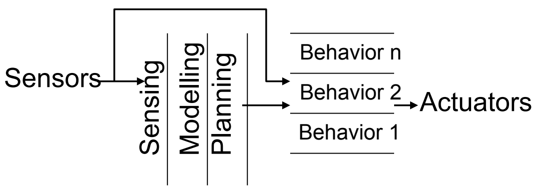
- Functional decomposition
- World model
- Planning
- Based on sense-model-plan-act
- **Predictable behavior**
- **Slow reaction**.

Reactive



- Decomposition in parallel robot Behaviors
- Absence of World model
- Based on sense-react
- **Fast reaction**
- **a bit random behavior**

Hybrid

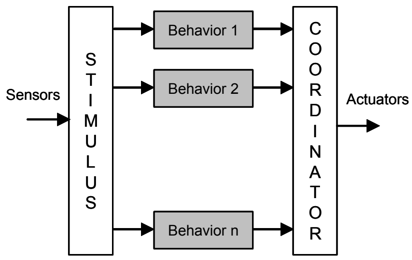


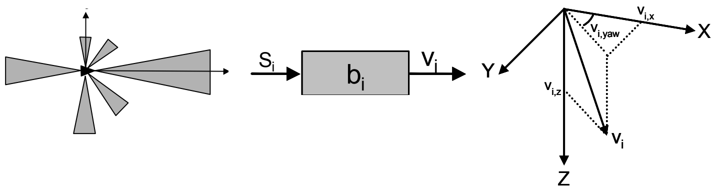
- World model
- Planning
- Reactive execution
- **Prediction**
- **Fast reaction**

2. Behavior-based architectures

Main features:

- Independent Behaviors: “go to”, “avoid obstacles”, …
- **Input**: perceived state of the environment
- **Output**: robot desired velocity
- **Coordination** of behaviors





Behavior-based architectures

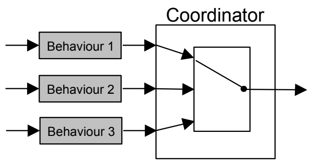
More features

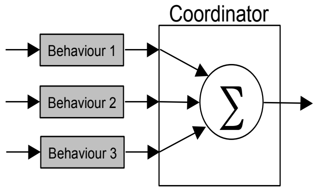
- Set of **simple** behaviours (i.e.: hardware implemented)
- Each behaviour acts **independently**: asynchronously, in its own hardware.
- Each behaviour represents and **intention of the robot**: ”go to a point”, “avoid obstacles”, “follow the corridor”,...
- Inspiration from **nature.**
- A **coordinator** selects at each time step the appropriate behaviour response.
- **Input**: Information from sensors. The perceived environment is used as the **best** representation of the world.
- **Internal states**: behaviour can have different internal states, acting differently according to them.
- **Output**: n-dimensional (n DOFs) vector indicating the direction and speed to be followed by the robot.

Coordination

The **coordinator** chooses the **best** behaviour **output** from all active behaviours.

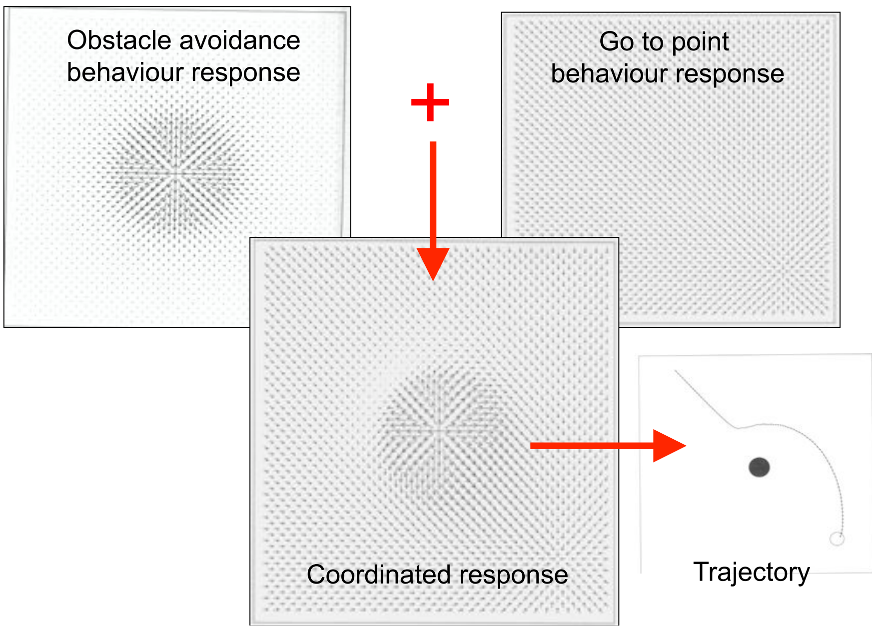
Coordination mechanisms can be **classified** in **2 groups**:





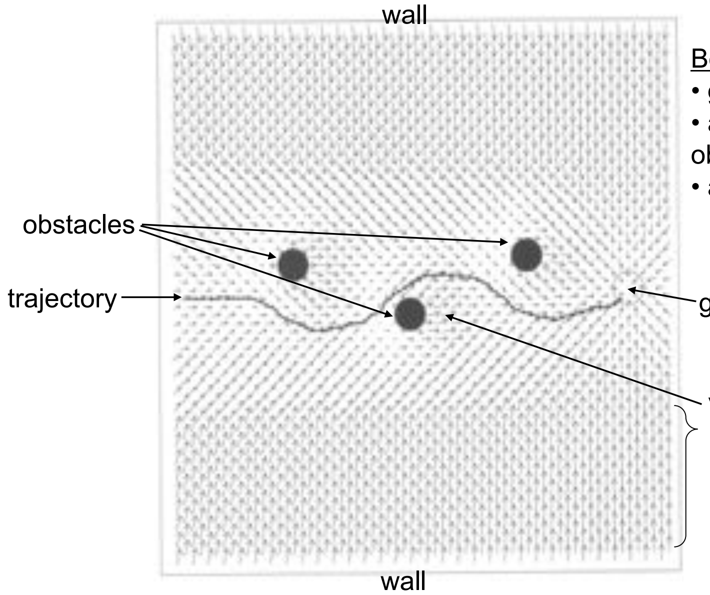
- **Competitive** Coordination
  - Only one response is chosen
- **Cooperative** Coordination
  - The final response is a merging of all the behaviours

Emergence



Autonomous Robots

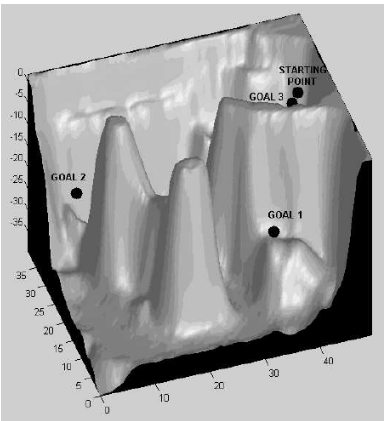
Emergence



Behaviours:
- go to point
- avoid moving obstacle
- avoid wall

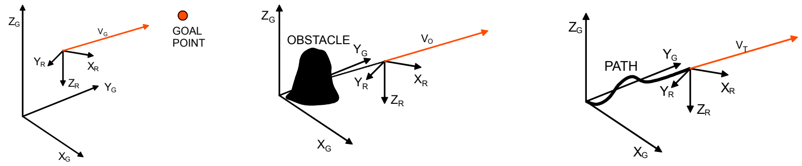
Autonomous Robots

Task example

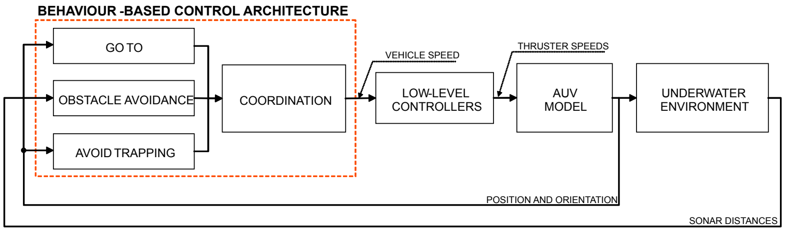


- **Evaluation task:** "To reach 3 goal-points avoiding obstacles"
- Simulated **environment using the** dynamics **model of GARBI underwater robot**
- **Predefined set of Behaviours to fulfil the task.**

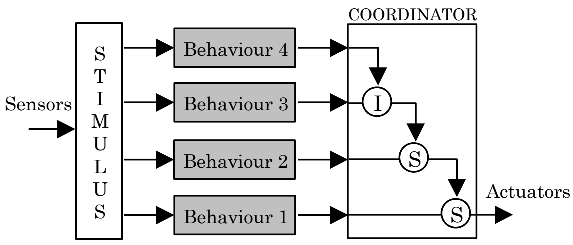
Task example



"Go To" Behavior **Obstacle Avoidance** **Avoid Trapping**



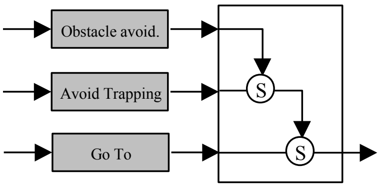
Subsumption Architecture



- **Competitive** coordination system.
- Each behaviour (layer) belongs to a hierarchy.
- When top layers are active, they cancel (**inhibition nodes**) or substitute (**suppression nodes**) the responses of lower layers.
- The layers are implemented with **Augmented Finite State Machine** (FSM with registers and timers) or with behavioural libraries.
- Principal developer: Rodney Brooks (M.I.T.).

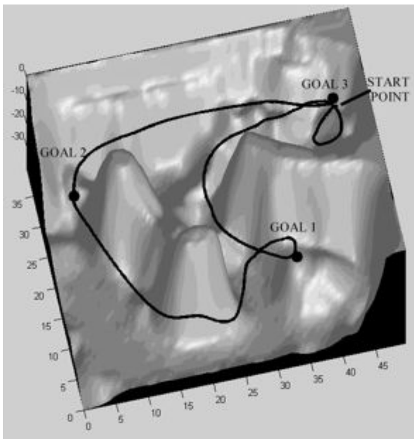
Autonomous Robots

Subsumption Architecture



- Hierarchy of behaviours:
  1st. avoid obstacle
  2nd. avoid trapping
  3rd. go to goal

- Implementation with **suppression** nodes.



Autonomous Robots

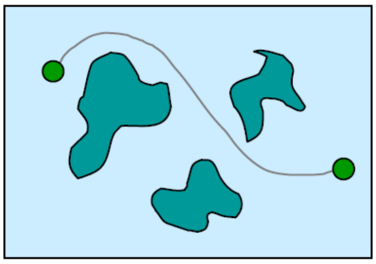
More examples

More examples

UdG Autonomous Robots

More examples

3. Path Planning



Outline

- Bug algorithms
- Configuration space
- Potential functions – Wavefront planner
- Topological maps – Visibility graph
- Graph search - A* algorithm
- Cell decompositions
- Sampling-based algorithms

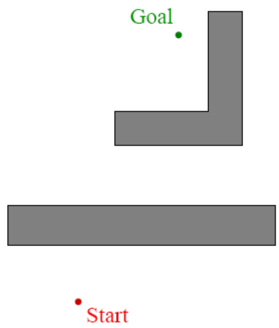
Bug algorithms

- They are inspired from insects
- Simple Bug behaviours:
  - follow a wall
  - move toward a goal
- Assumptions:
  - the direction to the goal is known
  - tactile sensors

Autonomous Robots

Bug algorithms

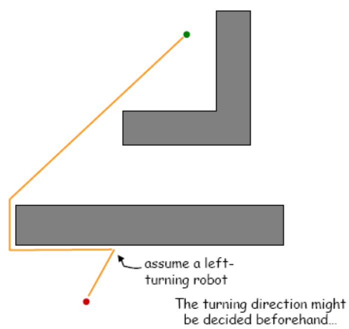
- Bug 0 algorithm:
  1. head toward goal
  2. follow obstacle (left or right) until you can head toward the goal again
  3. continue



Autonomous Robots

Bug algorithms

- Bug 0 algorithm:
  1. head toward goal
  2. follow obstacle (left or right) until you can head toward the goal again
  3. continue



Bug algorithms

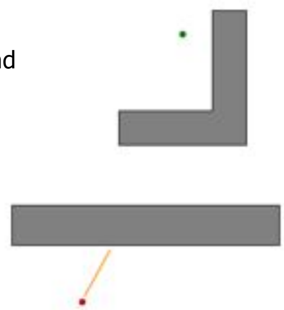
- Bug 0 algorithm:
  1. head toward goal
  2. follow obstacle (left or right) until you can head toward the goal again
  3. continue

What is the trajectory in this environment?

Bug algorithms

Adding some memory, it is possible to improve Bug 0

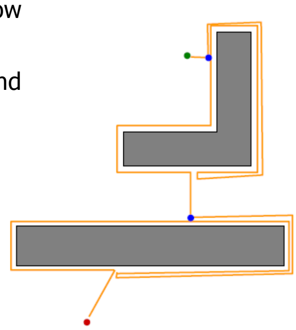
- Bug 1 algorithm:
  1. head toward goal
  2. if an obstacle is encountered circumnavigate it and remember how close you get to the goal
  3. return to that closest point and continue



Bug algorithms

Adding some memory, it is possible to improve Bug 0

- Bug 1 algorithm:
  1. head toward goal
  2. if an obstacle is encountered circumnavigate it and remember how close you get to the goal
  3. return to that closest point and continue

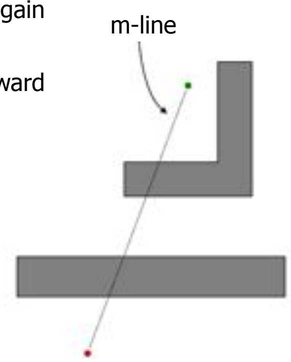


UdG Autonomous Robots

Bug algorithms

Another possibility

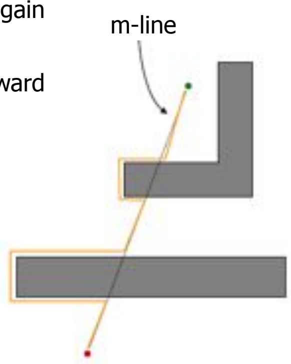
- Bug 2 algorithm:
  1. head toward goal on the m-line
  2. if an obstacle is in the way, follow it until you encounter the m-line again closer to the goal
  3. leave the obstacle and continue toward the goal



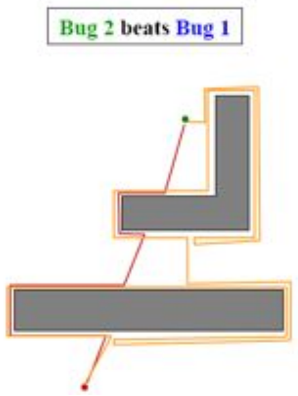
Bug algorithms

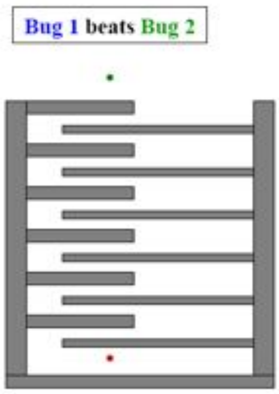
Another possibility

- Bug 2 algorithm:
  1. head toward goal on the m-line
  2. if an obstacle is in the way, follow it until you encounter the m-line again closer to the goal
  3. leave the obstacle and continue toward the goal



Bug algorithms





Bug 1 is an exhaustive search algorithm: *it looks first all choices*

Bug 2 is a greedy algorithm: *it takes the first thing that looks better*

Bug algorithms

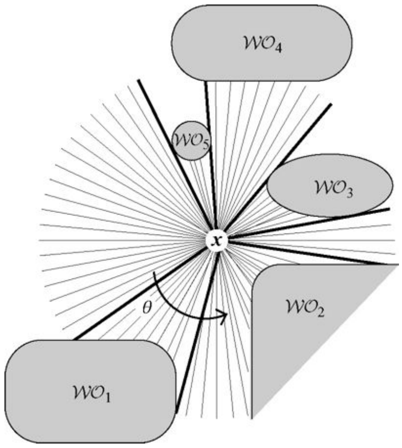
having range sensors...

- Tangent Bug algorithm:

$$\rho(x,\theta) = \min_{\lambda\in[0,\infty]} d(x, x+\lambda[\cos\theta, \sin\theta]^T),$$
$$\text{such that } x+\lambda[\cos\theta,\sin\theta]^T \in \bigcup_i \mathcal{WO}_i.$$

$$\rho_R : \mathbb{R}^2 \times S^1 \to \mathbb{R}$$

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta), & \text{if } \rho(x,\theta) < R \\ \infty, & \text{otherwise.} \end{cases}$$



Autonomous Robots UdG

Bug algorithms

- Tangent Bug algorithm:

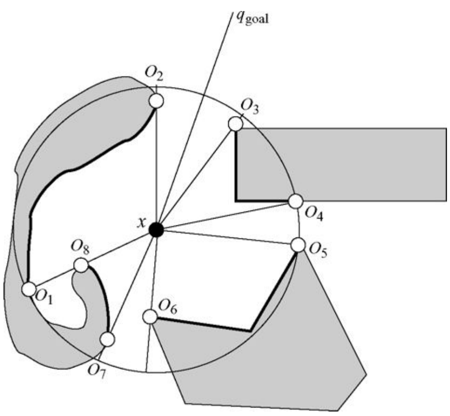
Discontinuity points:

O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8

Continuity intervals

$O_1 \rightarrow O_2$, $O_3 \rightarrow O_4$

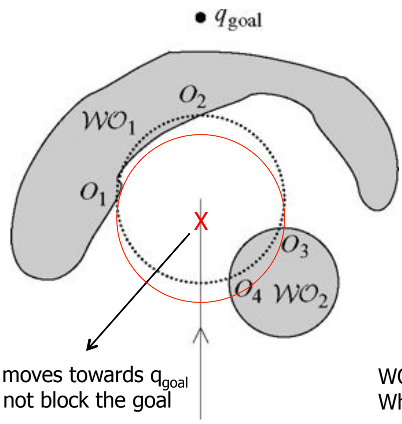
$O_5 \rightarrow O_6$, $O_7 \rightarrow O_8$



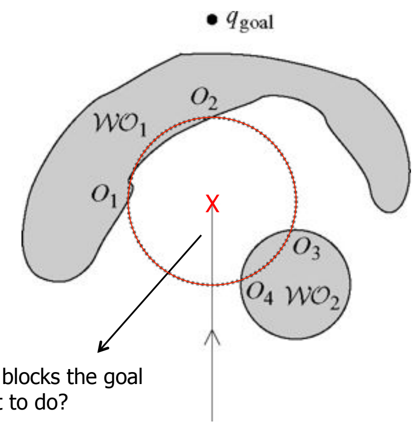
Autonomous Robots

Bug algorithms

- Tangent Bug algorithm:



The robot moves towards q_{goal}
WO_2 does not block the goal



WO_1 blocks the goal
What to do?

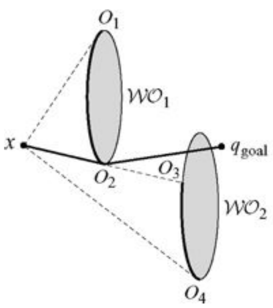
Autonomous Robots

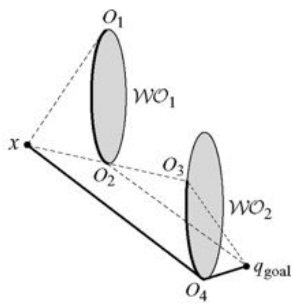
Bug algorithms

- Tangent Bug algorithm:

The robot then moves toward the Oi that maximally decreases a heuristic distance to the goal.

choose O_i that minimizes: $d(x, Oi) + d(Oi, q_{goal})$

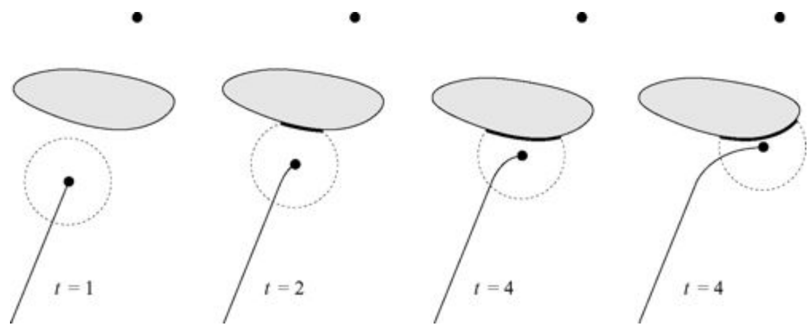




Bug algorithms

- Tangent Bug algorithm:

Avoiding the obstacle:
PART 1: MOTION TO GOAL BEHAVIOUR

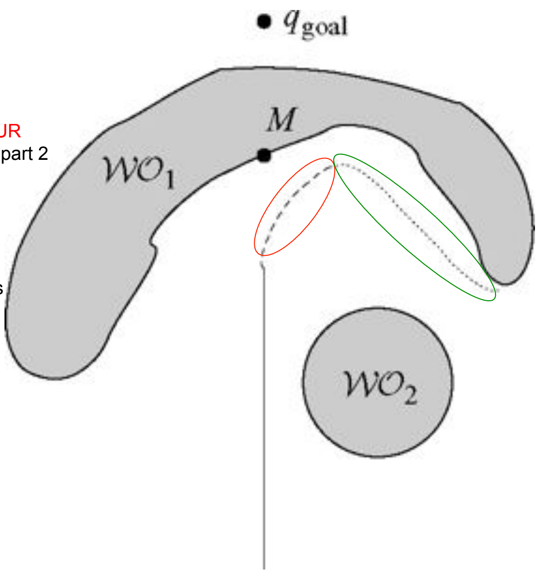


Bug algorithms

- Tangent Bug algorithm:

Avoiding the obstacle:
PART 1: MOTION TO GOAL BEHAVIOUR
... until d starts increasing, then part 2

PART 2: BOUNDARY FOLLOWING BEHAVIOUR
Follow the boundary until the distance to goal from one reachable point O_i (d_{reach}) is less than the distance to goal from any past followed point.
Then, part 1.



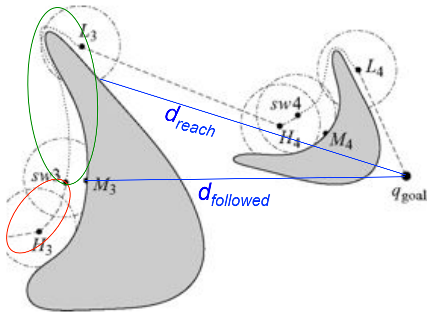
Autonomous Robots

UdG

Bug algorithms

- Tangent Bug algorithm:

$d_{followed}$ *is the shortest distance between the boundary which had been sensed and the goal.*



d_{reach} *is the distance between the goal and the closest point on the followed obstacle that is within line of sight of the robot*

$$d_{\text{reach}} = \min_{c \in \Lambda} d(q_{\text{goal}}, c).$$

Autonomous Robots

UdG

Bug algorithms

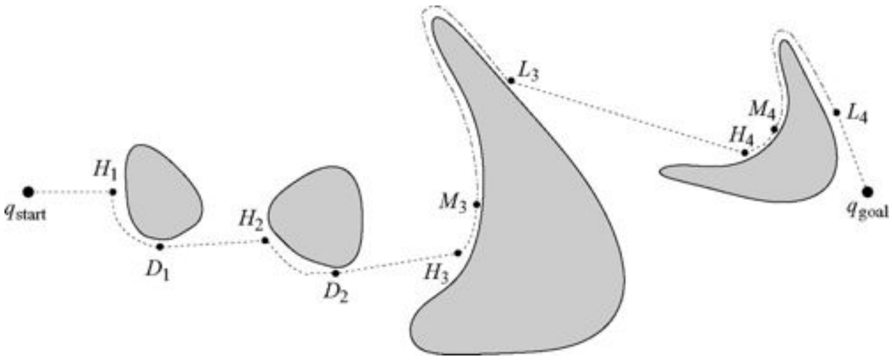
- Tangent Bug algorithm:


```
Input: A point robot with a range sensor
Output: A path to the q_goal or a conclusion no such path exists
while True do
  repeat
    Continuously move toward the point n ∈{T, O_i} which minimizes d(x, n) + d(n, q_goal)
  until
•   the goal is encountered or
•   The direction that minimizes d(x, n) + d(n, q_goal) begins to increase d(x, q_goal), i.e., the
  Chose a boundary following direction which continues in the same direction as the most recent
  repeat
    Continuously update d_reach, d_followed, and {O_i}.
    Continuously moves toward n ∈{O_i } that is in the chosen boundary direction.
  until
•   The goal is reached.
•   The robot completes a cycle around the obstacle in which case the goal cannot be achieved.
•   d_reach < d_followed
end while
```

Bug algorithms

- Tangent Bug algorithm:

Tangent Bug with zero sensor range

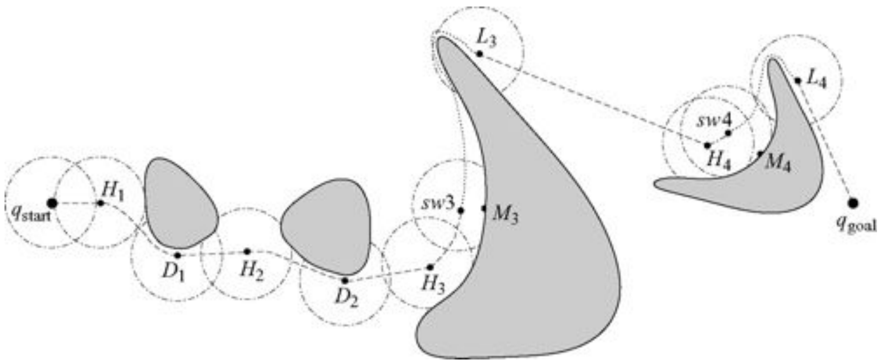


Autonomous
UdG Robots

Bug algorithms

- Tangent Bug algorithm:

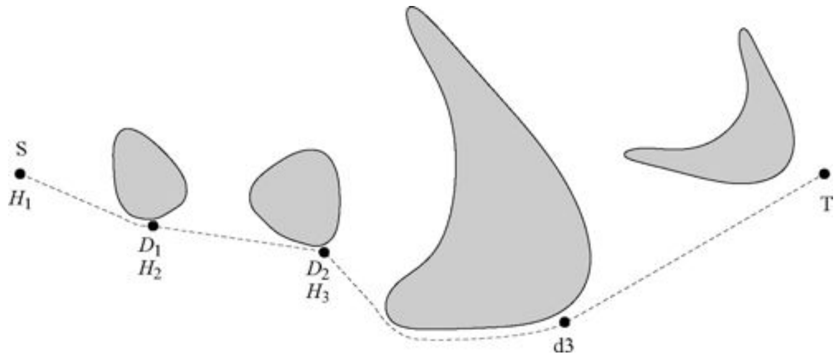
Tangent Bug with finite sensor range



Autonomous Robots — Bug algorithms

- Tangent Bug algorithm:

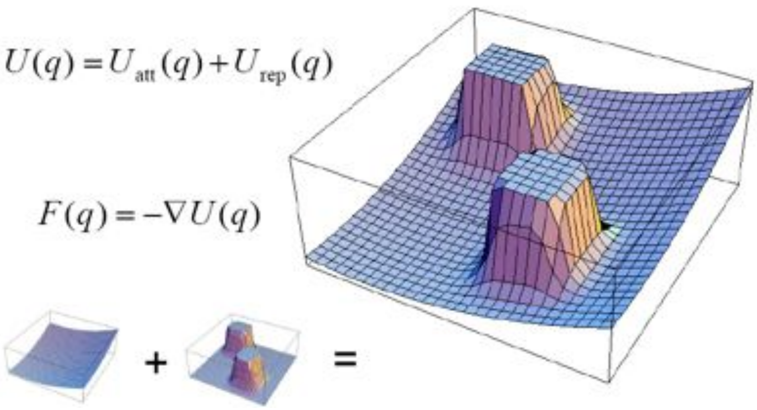
Tangent Bug with infinite sensor range



Autonomous Robots — Potential functions

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

$$F(q) = -\nabla U(q)$$



Autonomous
UdG Robots

Potential functions

- Finding the minimum:
  - The gradient of the total potential function indicates the way to the goal:

$$\dot{c}(t) = -\nabla U(c(t)).$$

  - since the total potential function depends on the number, position and shape of the obstacles, there can be local minimums!!

  - Solutions:
    - to operate mathematically the functions to eliminate local minimums → navigation functions
    - to divide the space into a grid → brushfire algorithm and wavefront planner

Autonomous
UdG Robots

Potential functions

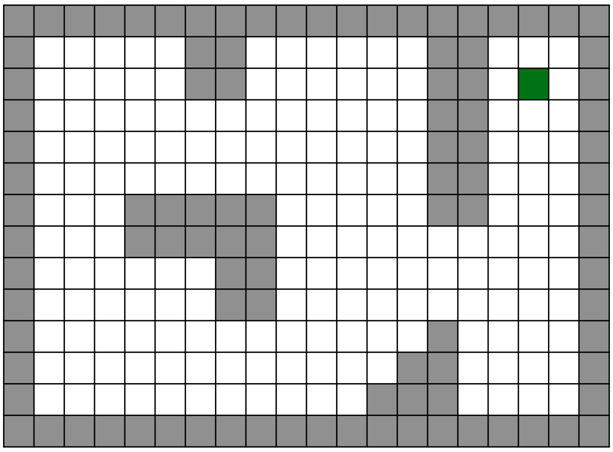
- Brushfire algorithm:
  - To compute the gradient of the repulsive functions
  - Define a grid on the space
  - Choose 4 or 8 point connectivity

4 8

  - Obstacles start with a 1; free space zero
  - Until all cells >0; assign to all **connected cells** the minimum non-zero value plus 1
  - The result is a map where each cell holds the minimum distance to an obstacle
  - The gradient of distance is easily found by taking differences with all neighbouring cells

Potential functions

- Brushfire algorithm:
  2D finite environment, 20x14 cells



Autonomous
UdG Robots

Potential functions

- Brushfire algorithm:
  with 4-point connectivity, 1st iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Potential functions

- Brushfire algorithm:
  with 4-point connectivity, 2nd iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	2	2	1	1	2	2	2	2	2	2	1	1	2	2	2	1
1	2	0	0	0	2	1	1	2	0	0	0	0	2	1	1	2	0	2	1
1	2	0	0	0	0	2	2	0	0	0	0	0	2	1	1	2	0	2	1
1	2	0	0	0	0	0	0	0	0	0	0	0	2	1	1	2	0	2	1
1	2	0	0	2	2	2	2	2	0	0	0	0	2	1	1	2	0	2	1
1	2	0	2	1	1	1	1	1	2	0	0	0	2	1	1	2	0	2	1
1	2	0	2	1	1	1	1	1	2	0	0	0	0	2	2	0	0	2	1
1	2	0	0	2	2	2	1	1	2	0	0	0	0	0	0	0	0	2	1
1	2	0	0	0	0	2	1	1	2	0	0	0	0	2	0	0	0	2	1
1	2	0	0	0	0	0	2	2	0	0	0	0	2	1	2	0	0	2	1
1	2	0	0	0	0	0	0	0	0	0	0	2	1	1	2	0	0	2	1
1	2	2	2	2	2	2	2	2	2	2	2	1	1	1	2	2	2	2	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Potential functions

- Brushfire algorithm:
  with 4-point connectivity, 5th iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	2	2	1	1	2	2	2	2	2	2	1	1	2	2	2	1
1	2	3	3	3	2	1	1	2	3	3	3	3	2	1	1	2	3	2	1
1	2	3	4	4	3	2	2	3	4	4	4	3	2	1	1	2	3	2	1
1	2	3	4	3	3	3	3	3	4	5	4	3	2	1	1	2	3	2	1
1	2	3	3	2	2	2	2	2	3	4	4	3	2	1	1	2	3	2	1
1	2	3	2	1	1	1	1	1	2	3	4	3	2	1	1	2	3	2	1
1	2	3	2	1	1	1	1	1	2	3	4	4	3	2	2	3	3	2	1
1	2	3	3	2	2	2	1	1	2	3	4	5	4	3	3	4	3	2	1
1	2	3	4	3	3	2	1	1	2	3	4	4	3	2	3	4	3	2	1
1	2	3	4	4	4	3	2	2	3	4	4	3	2	1	2	3	3	2	1
1	2	3	3	3	3	3	3	3	3	3	3	2	1	1	2	3	3	2	1
1	2	2	2	2	2	2	2	2	2	2	2	1	1	1	2	2	2	2	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Autonomous Robots — UdG

Potential functions

- Brushfire algorithm:
  with 8-point connectivity, 4th iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	2	2	1	1	2	2	2	2	2	2	1	1	2	2	2	1
1	2	3	3	3	2	1	1	2	3	3	3	3	2	1	1	2	3	2	1
1	2	3	4	3	2	2	2	2	3	4	4	3	2	1	1	2	3	2	1
1	2	3	3	3	3	3	3	3	3	3	4	3	2	1	1	2	3	2	1
1	2	3	2	2	2	2	2	2	2	3	4	3	2	1	1	2	3	2	1
1	2	3	2	1	1	1	1	1	2	3	4	3	2	1	1	2	3	2	1
1	2	3	2	1	1	1	1	1	2	3	4	3	2	2	2	2	3	2	1
1	2	3	2	2	2	2	1	1	2	3	4	3	3	3	3	3	3	2	1
1	2	3	3	3	3	2	1	1	2	3	3	3	2	2	2	3	3	2	1
1	2	3	4	4	3	2	2	2	2	3	3	2	2	1	2	3	3	2	1
1	2	3	3	3	3	3	3	3	3	3	2	2	1	1	2	3	3	2	1
1	2	2	2	2	2	2	2	2	2	2	2	1	1	1	2	2	2	2	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Autonomous Robots — UdG

Potential functions

- Wavefront planner:
  - Planner based on the brushfire algorithm
  - The algorithm starts from the goal position (labelled with a 2)
  - The “1” cells are not considered
  - The result is the distance to the goal (-2)
  - Gradient descent indicates the direction to go
  - Drawbacks
    - The planner has to search the entire space
    - Does not scale well in higher dimensions or big spaces!! Computationally intractable. In 3D,

      4-point connectivity → 6-point connectivity

      8-point connectivity → 26-point connectivity

Potential functions

- Wavefront planner:

  with 4-point connectivity, 1st iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	2	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1
1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Potential functions

- Wavefront planner:

  with 4-point connectivity, 10th iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	4	3	4	1
1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	3	2	3	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	4	3	4	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	5	4	5	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	6	5	6	1
1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	7	6	7	1
1	0	0	0	1	1	1	1	1	0	0	0	0	11	10	9	8	7	8	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	11	10	9	8	9	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	11	10	9	10	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	11	10	11	1
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	11	0	1
1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Potential functions

- Wavefront planner:
  with 4-point connectivity, 27th iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	29	28	27	26	25	1	1	22	21	20	19	18	17	1	1	4	3	4	1
1	28	27	26	25	24	1	1	21	20	19	18	17	16	1	1	3	2	3	1
1	27	26	25	24	23	22	21	20	19	18	17	16	15	1	1	4	3	4	1
1	26	25	24	23	22	21	20	19	18	17	16	15	14	1	1	5	4	5	1
1	25	24	23	22	21	20	19	18	17	16	15	14	13	1	1	6	5	6	1
1	26	25	24	1	1	1	1	1	16	15	14	13	12	1	1	7	6	7	1
1	27	26	25	1	1	1	1	1	15	14	13	12	11	10	9	8	7	8	1
1	28	27	26	25	24	23	1	1	16	15	14	13	12	11	10	9	8	9	1
1	27	26	25	24	23	22	1	1	17	16	15	14	13	12	11	10	9	10	1
1	26	25	24	23	22	21	20	19	18	17	16	15	14	1	12	11	10	11	1
1	27	26	25	24	23	22	21	20	19	18	17	16	1	1	13	12	11	12	1
1	28	27	26	25	24	23	22	21	20	19	18	1	1	1	14	13	12	13	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Potential functions

- Wavefront planner:
  with 4-point connectivity, one shortest trajectory

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	29	28	27	26	25	1	1	22	21	20	19	18	17	1	1	4	3	4	1
1	28	27	26	25	24	1	1	21	20	19	18	17	16	1	1	3	2	3	1
1	27	26	25	24	23	22	21	20	19	18	17	16	15	1	1	4	**3**	4	1
1	26	25	24	23	22	21	20	19	18	17	16	15	14	1	1	5	**4**	5	1
1	25	24	23	22	21	20	19	18	17	16	15	14	13	1	1	6	**5**	6	1
1	26	25	24	1	1	1	1	1	16	15	14	13	12	1	1	7	**6**	7	1
1	27	26	25	1	1	1	1	1	15	14	**13**	**12**	**11**	**10**	**9**	**8**	**7**	8	1
1	28	27	26	25	24	23	1	1	16	15	**14**	13	12	11	10	9	8	9	1
1	27	26	25	24	23	22	1	1	17	16	**15**	14	13	12	11	10	9	10	1
1	26	25	24	23	22	21	20	19	18	17	**16**	15	14	1	12	11	10	11	1
1	27	26	25	24	23	22	21	20	19	18	**17**	16	1	1	13	12	11	12	1
1	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	1	1	1	14	13	12	13	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

From starting point, gradient descent indicates direction to goal.

Potential functions

- Wavefront planner:

  with 8-point connectivity, 20th iteration

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	21	20	19	18	18	1	1	14	14	14	14	14	14	1	1	3	3	3	1
1	21	20	19	18	17	1	1	14	13	13	13	13	13	1	1	3	2	3	1
1	21	20	19	18	17	16	15	14	13	12	12	12	12	1	1	3	3	3	1
1	21	20	19	18	17	16	15	14	13	12	11	11	11	1	1	4	4	4	1
1	21	20	19	18	17	16	15	14	13	12	11	10	10	1	1	5	5	5	1
1	21	20	19	1	1	1	1	1	13	12	11	10	9	1	1	6	6	6	1
1	21	20	19	1	1	1	1	1	13	12	11	10	9	8	7	7	7	7	1
1	21	20	19	18	17	17	1	1	13	12	11	10	9	8	8	8	8	8	1
1	21	20	19	18	17	16	1	1	13	12	11	10	9	9	9	9	9	9	1
1	21	20	19	18	17	16	15	14	13	12	11	10	10	1	10	10	10	10	1
1	21	20	19	18	17	16	15	14	13	12	11	11	1	1	11	11	11	11	1
1	21	20	19	18	17	16	15	14	13	12	12	1	1	1	12	12	12	12	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

UdG Autonomous Robots

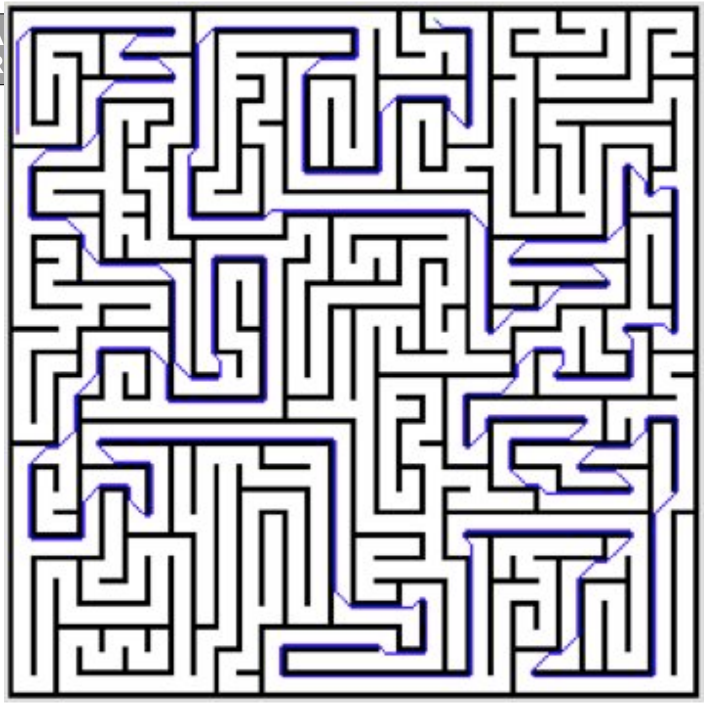
Potential functions

- Wavefront planner:

  with 8-point connectivity, one shortest trajectory

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	21	20	19	18	18	1	1	14	14	14	14	14	14	1	1	3	3	3	1
1	21	20	19	18	17	1	1	14	13	13	13	13	13	1	1	3	2	3	1
1	21	20	19	18	17	16	15	14	13	12	12	12	12	1	1	3	3	3	1
1	21	20	19	18	17	16	15	14	13	12	11	11	11	1	1	4	4	4	1
1	21	20	19	18	17	16	15	14	13	12	11	10	10	1	1	5	5	5	1
1	21	20	19	1	1	1	1	1	13	12	11	10	9	1	1	6	6	6	1
1	21	20	19	1	1	1	1	1	13	12	11	10	9	8	7	7	7	7	1
1	21	20	19	18	17	17	1	1	13	12	11	10	9	8	8	8	8	8	1
1	21	20	19	18	17	16	1	1	13	12	11	10	9	9	9	9	9	9	1
1	21	20	19	18	17	16	15	14	13	12	11	10	10	1	10	10	10	10	1
1	21	20	19	18	17	16	15	14	13	12	11	11	1	1	11	11	11	11	1
1	21	20	19	18	17	16	15	14	13	12	12	1	1	1	12	12	12	12	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

From starting point, gradient descent indicates direction to goal.

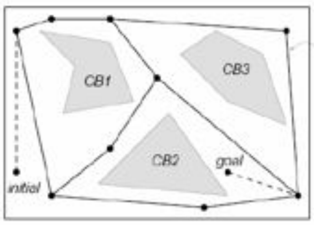


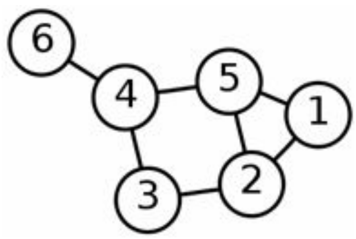
Autonomous
Robots

Topological maps

Planning in topological maps

- Topological map: simplified map with only relationship between points. It can be represented as a graph:
  - nodes are real positions
  - edges join positions in the free space, they include the distance
- It is easy to find a path in a topological map. How to build a topological map?
  - Visibility graph
  - Voronoi diagram
- How to solve the graph?
  - A* algorithm



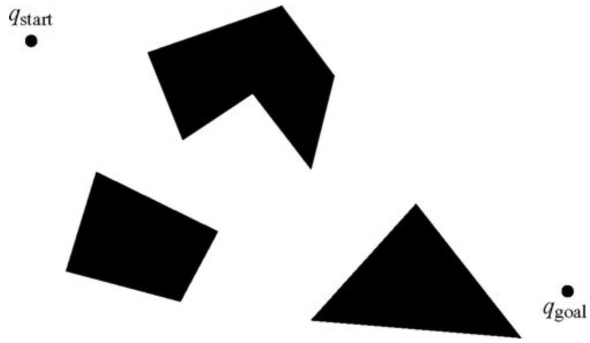


Top. Maps: Visibility Graph

Defined for a 2D polygonal configuration space

- The nodes v_i of the visibility graph include the start location, the goal location, and all the vertices of the configuration space obstacles.
- The graph edges e_{ij} are straight-line segments that connect two line-of-sight nodes v_i and v_j, i.e.,

$$e_{ij} \neq \emptyset \iff s v_i + (1-s) v_j \in \mathrm{cl}(\mathcal{Q}_{\text{free}}) \;\; \forall s \in [0, 1].$$

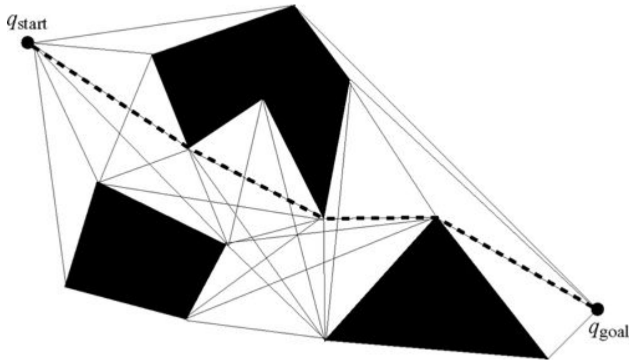


Top. Maps: Visibility Graph

- Construction of the visibility graph with n nodes has complexity n^3

  for all nodes; for all potential edges; for all obstacle edges

  wich can be reduced with the Rotational Plane Sweep Algorithm ($n^2 \log n$).
- Using the euclidean distance, the graph can be searched to find the shortest distance.

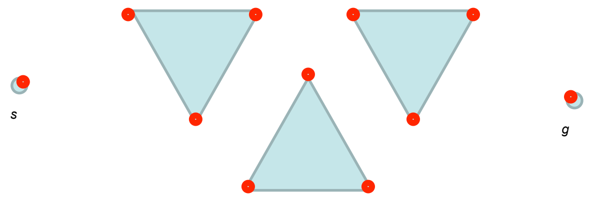


Top. Maps: Visibility Graph

Visibility graph construction with brute force

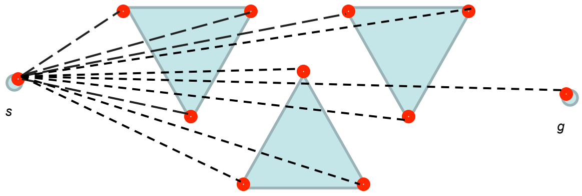
Autonomous
UdG Robots

Top. Maps: Visibility Graph



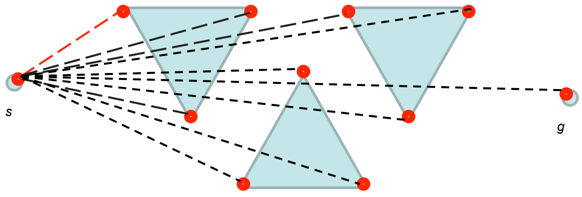
Autonomous UdG Robots

Top. Maps: Visibility Graph



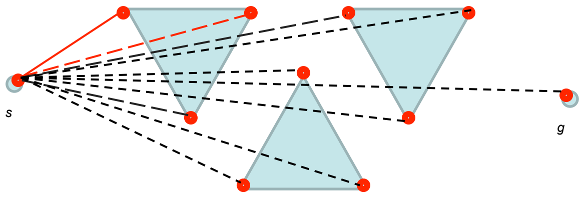
Autonomous UdG Robots

Top. Maps: Visibility Graph



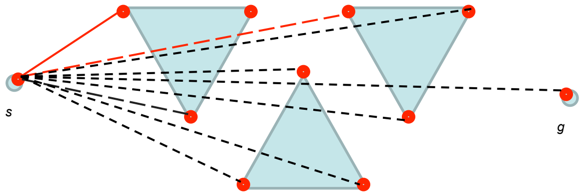
Top. Maps: Visibility Graph

Autonomous Robots



Top. Maps: Visibility Graph

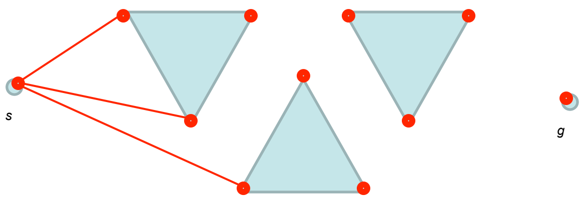
Autonomous Robots



Autonomous Robots

UdG

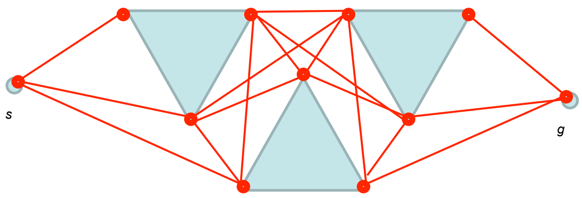
Top. Maps: Visibility Graph



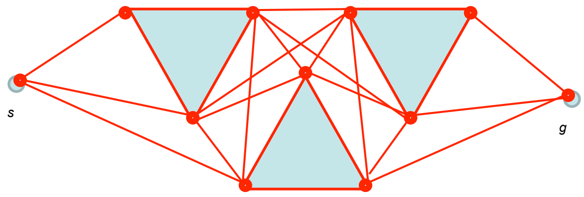
Autonomous Robots

UdG

Top. Maps: Visibility Graph



Top. Maps: Visibility Graph



Autonomous
UdG Robots

Top. Maps: Visibility Graph

Rotational plane sweep algorithm

Algorithm for building the visibility graph in a total time complexity of $n^2 \log n$:

- A rotating half-line emanating from any vertex will be used to determine the vertices which are visible.
- The half-line has to stop only in the directions in which there is a vertex.
- At each vertex angle, a list of edges which intersect the beam will be updated (list S).
- Since the line rotates following the sorted list of vertex angles, list ε, the updating of the S list consists only on adding or removing the edges that contain the candidate vertex.
- Then, to determine if the vertex is visible, only intersection with lines contained in the S list, that are closer than the candidate vertex, have to be checked.

Top. Maps: Visibility Graph

Rotational plane sweep algorithm

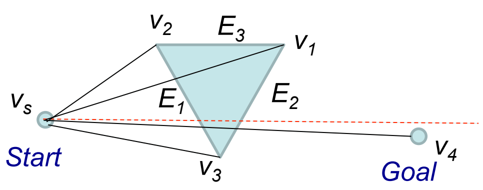
Algorithm 5: Rotational Plane Sweep Algorithm

Input: A set of vertices $\{v_i\}$ (whose edges do not intersect) and a vertex v
Output: A subset of vertices from $\{v_i\}$ that are within line of sight of v
1: For each vertex v_i, calculate α_i, the angle from the horizontal axis to the line segment $\overline{vv_i}$.
2: Create the vertex list $\mathcal{E}$, containing the α_i 's sorted in increasing order.
3: Create the active list $\mathcal{S}$, containing the sorted list of edges that intersect the horizontal half-line emanating from v.
4: **for all** α_i **do**
5:   **if** v_i is visible to v **then**
6:     Add the edge (v, v_i) to the visibility graph.
7:   **end if**
8:   **if** v_i is the beginning of an edge, E, not in $\mathcal{S}$ **then**
9:     Insert the E into $\mathcal{S}$.
10:   **end if**
11:   **if** v_i is the end of an edge in $\mathcal{S}$ **then**
12:     Delete the edge from $\mathcal{S}$.
13:   **end if**
14: **end for**

Autonomous
UdG Robots

Top. Maps: Visibility Graph

Rotational plane sweep algorithm

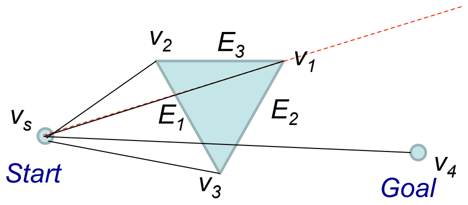


$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

Initialization:
$S=\{E_1,E_2\}$

Top. Maps: Visibility Graph

Rotational plane sweep algorithm



$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

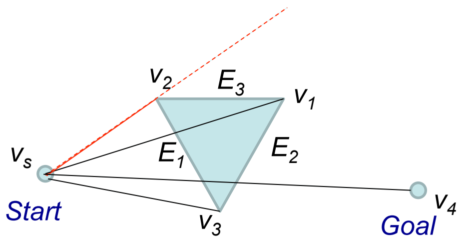
Iteration 1, stop at α_1 :
$S=\{E_1,E_3\}$

V_sV_1 intersects with E_1!

Autonomous
UdG Robots

Top. Maps: Visibility Graph

Rotational plane sweep algorithm



$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

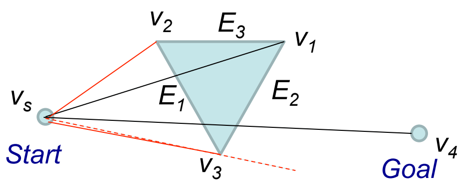
Iteration 2, stop at α_2 :
$S=\{\}$

V_sV_2 is visible!

Autonomous Robots UdG

Top. Maps: Visibility Graph

Rotational plane sweep algorithm



ε={α_1, α_2, α_3, α_4}

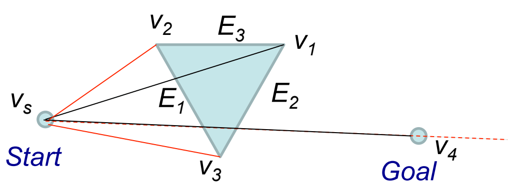
Iteration 3, stop at α_3 :
S={E_1, E_2}

V_sV_3 does not intersect with E_1, it is visible!

Autonomous Robots UdG

Top. Maps: Visibility Graph

Rotational plane sweep algorithm



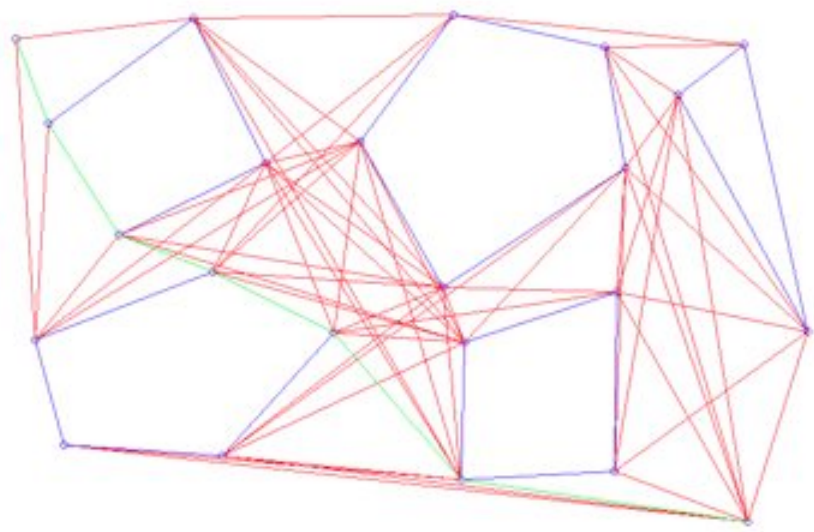
ε={α_1, α_2, α_3, α_4}

Iteration 4, stop at α_4 :
S={E_1, E_2}

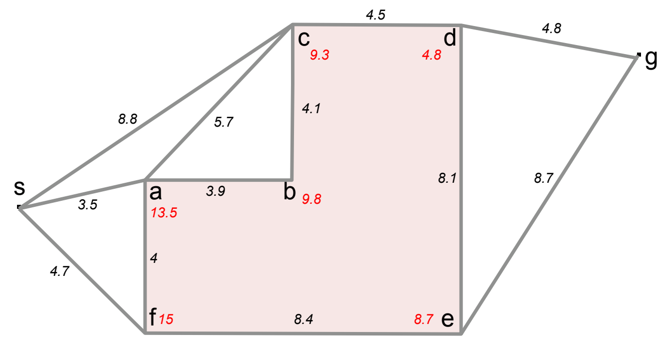
V_sV_4 intersects with E_1 and E_2!

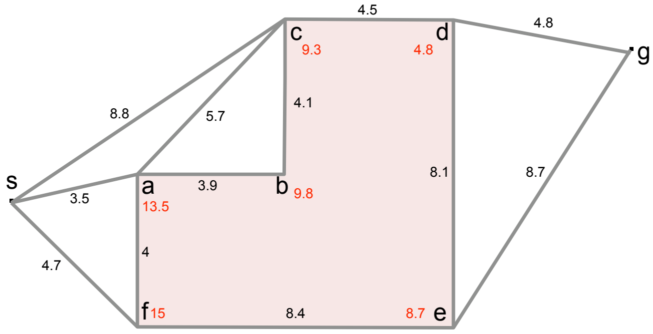
Top. Maps: Visibility Graph

Rotational plane sweep algorithm + A*



Graph search - A algorithm*



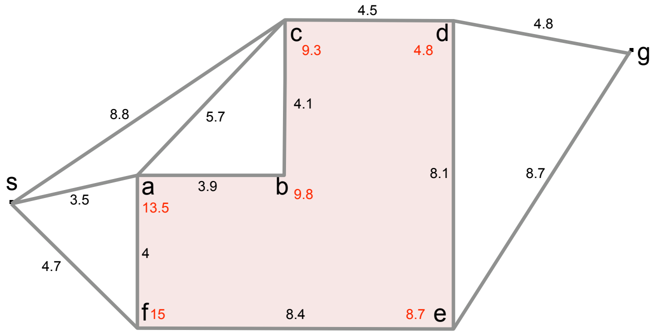


O list

Nodes	Cost
a	17
c	18.1
f	19.7

C list

Nodes	Backpointer
s	-

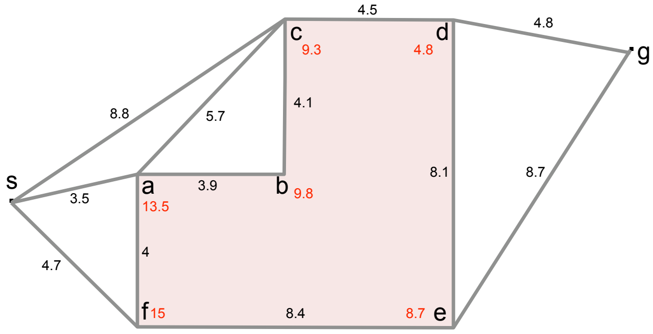


O list

Nodes	Cost
b	17.2
c	18.1
f	19.7

C list

Nodes	Backpointer
s	-
a	s

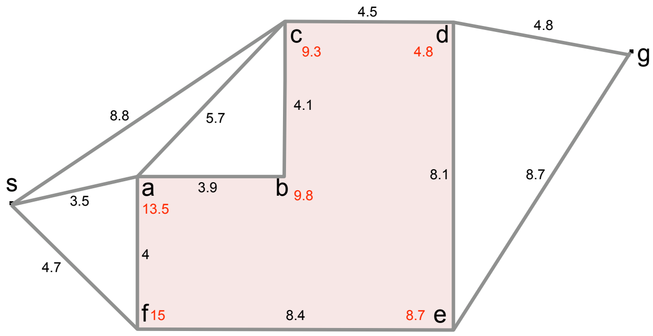


O list

Nodes	Cost
c	18.1
f	19.7

C list

Nodes	Backpointer
s	-
a	s
b	a

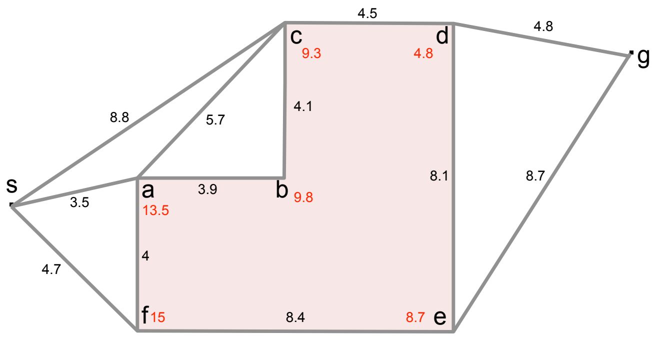


O list

Nodes	Cost
d	18.1
f	19.7

C list

Nodes	Backpointer
s	-
a	s
b	a
c	s

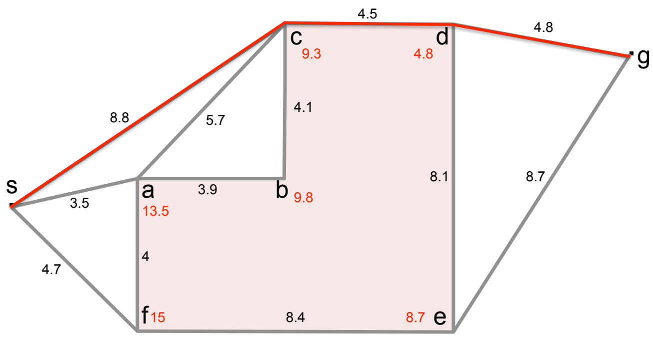


O list

Nodes	Cost
g	18.1
f	19.7
e	30.1

C list

Nodes	Backpointer
s	-
a	s
b	a
c	s
d	c



O list

Nodes	Cost
f	19.7
e	30.1

C list

Nodes	Backpointer
s	-
a	s
b	a
c	s
d	c
g	d

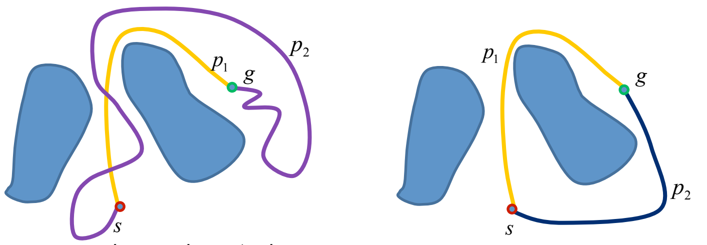
→ *2D Path planning from bathymetric maps for goal achievement using topological information based on homotopies.*

Bathymetric map — *2D map* — *Path in the workspace for each homotopic class*

UdG Autonomous Robots Research project: AUV topological path planning

Homotopy Definition

- Let p_1, p_2: [0, 1] → R^2 be two continuous paths. Then p_1 and p_2 are homotopic with respect to a set of obstacles $V \subseteq R^2$ if p_1 can be continuously deformed into p_2 while avoiding the obstacles.
- Example

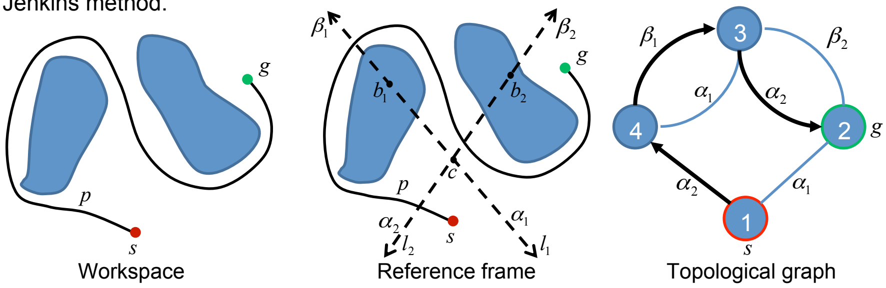


p_1 and p_2 are homotopic — *p_1 and p_2 are not homotopic*

Autonomous Robots UdG **Research project: AUV topological path planning**

From the workspace to the topological graph

Conversion of the metric workspace into a topological one using an extension of the Jenkins method.

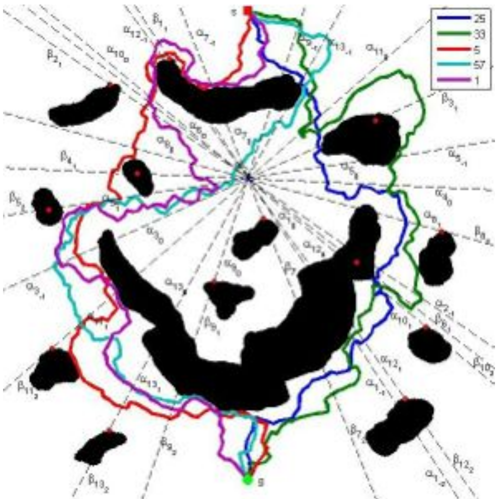


The reference frame is the link between the metric workspace and the topological graph.
Any path can be described by the ordered sequence of the traversed segments in the reference frame.
The topological graph is used to generate systematically, all the topological paths (homotopy classes) discarding the duplicates and those which are ensured to self-cross.

Autonomous Robots UdG **Research project: AUV topological path planning**

Topologically guided path planning

- Extension of the Jenkins method for allowing any class to be followed.
- A lower bound of the optimal path can be calculated for each homotopy class.
- The classes with smaller lower bound can be explored with a modified version of the RRT algorithm (HRRT) or the A* algorithm (HA*) to find the global optimal path.

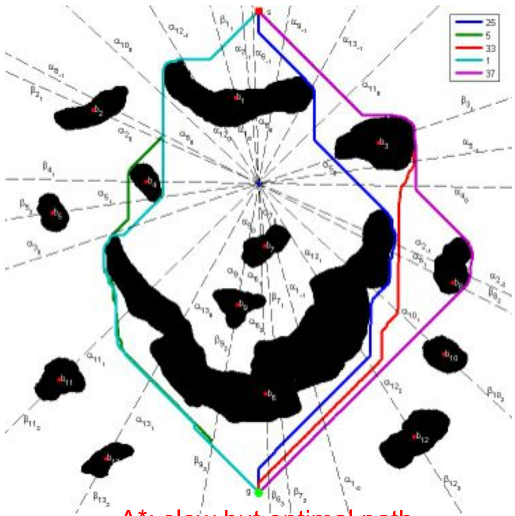


HRRT: fast suboptimal path

Autonomous Robots UdG *Research project: AUV topological path planning*

Topologically guided path planning

- Extension of the Jenkins method for allowing any class to be followed.
- A lower bound of the optimal path can be calculated for each homotopy class.
- The classes with smaller lower bound can be explored with a modified version of the RRT algorithm (HRRT) or the A* algorithm (HA*) to find the global optimal path.

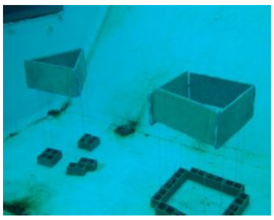


A*: slow but optimal path

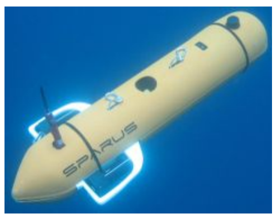
Autonomous Robots UdG *Research project: AUV topological path planning*

Experimental results

• Test of the proposal in real conditions with SPARUS^{AUV} in a controlled unknown environment to test its applicability to real applications.



Set up in the water tank of the UdG

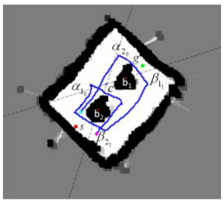


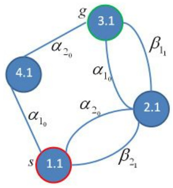
SPARUS^{AUV}

- MSIS configuration: 360º sector, 5m range with a 0.1m resolution and a 1.8 angular step.
- Dead-reckoning computed using the velocity readings coming from the DVL and the heading data obtained from the MRU sensor, both merged with an EKF.

Preliminary experimental results

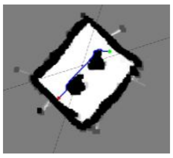
- Resultant OGM map with its reference frame and topological graph

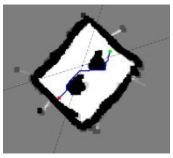


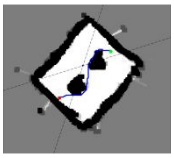


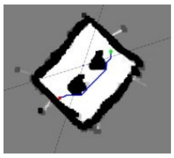
Homotopy class	Length (m)
$\alpha_{1_0}\alpha_{2_0}$	8.38
$\alpha_{2_0}\beta_{1_1}$	8.76
$\beta_{2_1}\alpha_{1_0}$	9.58
$\beta_{2_1}\beta_{1_1}$	8.40

- Homotopy classes and their paths in the workspace

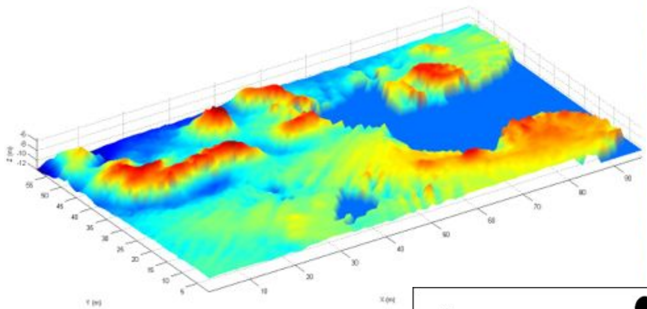

$\alpha_{1_0}\alpha_{2_0}$


$\alpha_{2_0}\beta_{1_1}$

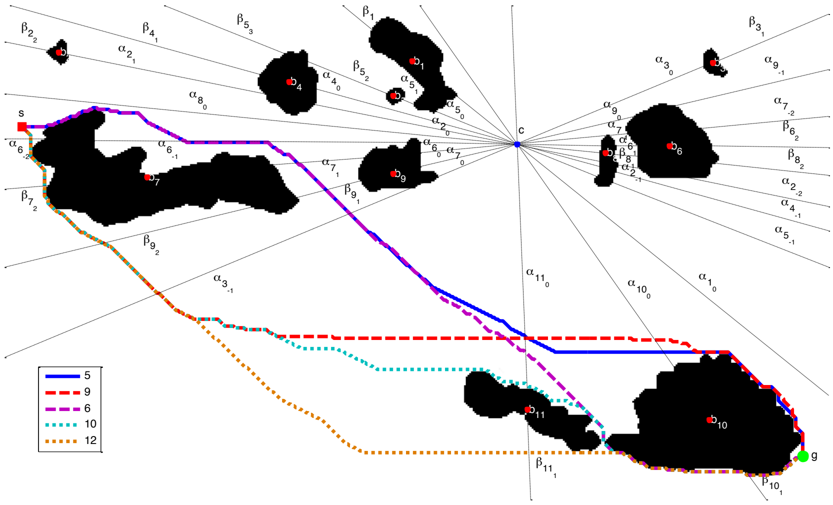

$\beta_{2_1}\alpha_{1_0}$


$\beta_{2_1}\beta_{1_1}$

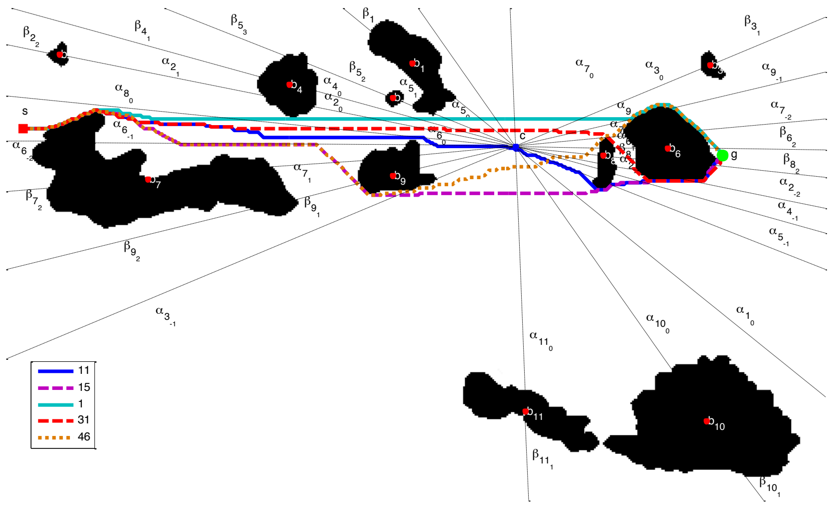
Autonomous Robots UdG
Research project: AUV topological path planning



Autonomous Robots UdG **Research project: AUV topological path planning**

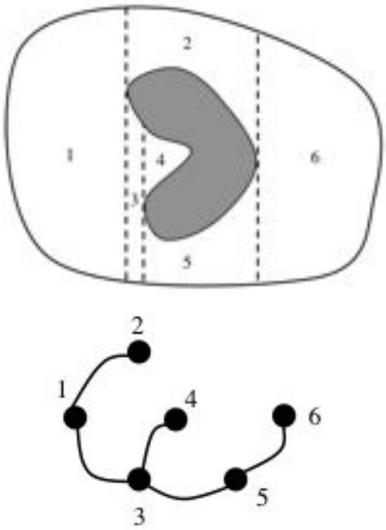


Autonomous Robots UdG **Research project: AUV topological path planning**



Cell decomposition

- Division of the free space in a set of **cells**.
- Adjacent cells share a **boundary**, and based on this, an **adjacency graph** can be built.
- **Path planning** is done by first determining the cells that contain the start and goal positions, and then finding a path within the adjacency graph. The A* or other graph search algorithms can be used.
- The adjacency graph can also be considered as a **topological map**.
- Cell decomposition is often used for **coverage path planning**.

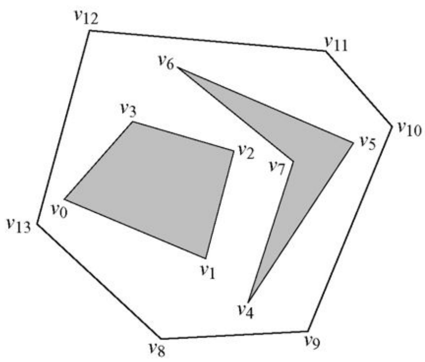


Autonomous
UdG **R**obots

Cell decomposition

Trapezoidal Decomposition

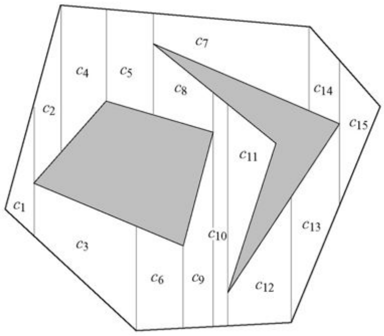
- Cells that are shaped like trapezoids: 4 sides, and also triangles (1 side has 0-length).
- Only for polygonal obstacles, which will have a set of vertices.
- At each vertex v_i, an upper and/or lower vertical edges appear, which will generate the boundaries between adjacent cells.
- At each vertex v_i, the adjacency graph is also updated accordingly.

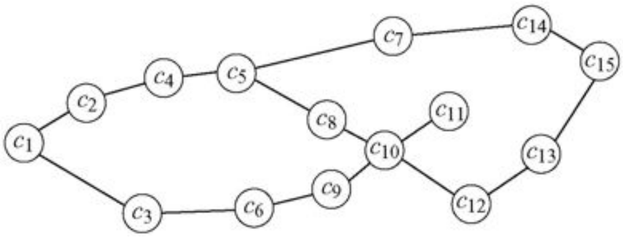


Cell decomposition

Trapezoidal Decomposition

- Each cell has its corresponding graph node.
- Cells which contain the start and goal positions must be found.
- Planning will take place at the adjacency graph.
- Midpoints will be used to translate the plan found in the graph into the free space.

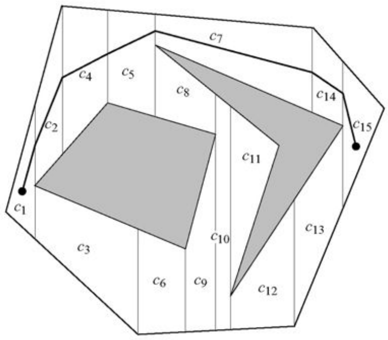


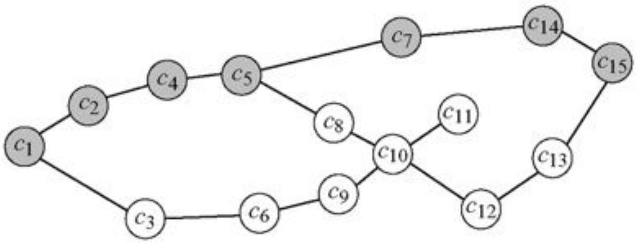


Cell decomposition

Trapezoidal Decomposition

- Each cell has its corresponding graph node.
- Cells which contain the start and goal positions must be found.
- Planning will take place at the adjacency graph.
- Midpoints will be used to translate the plan found in the graph into the free space.



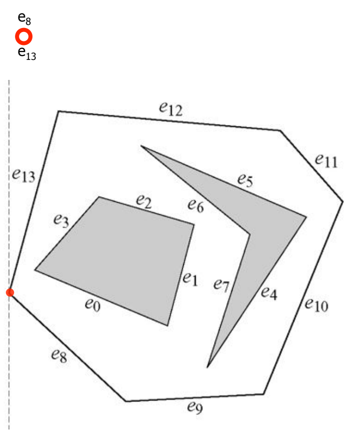


Cell decomposition

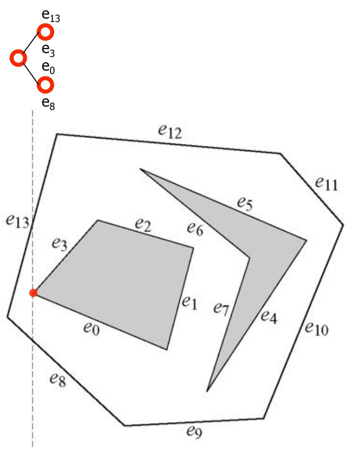
Trapezoidal Decomposition

- In order to build cells, a vertical sweep line from left to right is used.
- All vertices are sorted from left to right.
- The sweep line stops at each vertex v_i and a list L, containing intersected edges, is updated.
- By calculating the y coordinate of the intersection between the sweep line and each vertex contained in L, we can easily know the upper (e_{UPPER}) and lower (e_{LOWER}) edges of the current vertex.
- The update of L is done considering the 2 edges that belong to v_i. If an edge belongs to the list, it is removed; and if it is not in the list, it is added.
- If both are not in L, the second vertex of each edge is used to sort them.
- The edge on the left of the vertex edges in L will be e_{LOWER}, and the edge on the rigth will be e_{UPPER}.

Cell decomposition

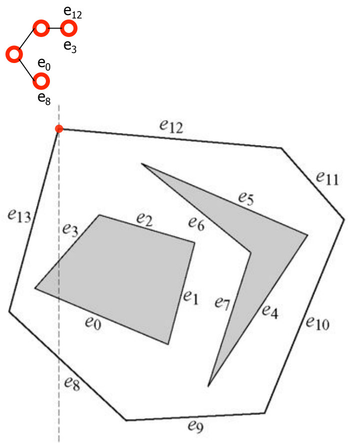


L:{e_8,e_{13}}
e_{UPPER}: -
e_{LOWER}: -

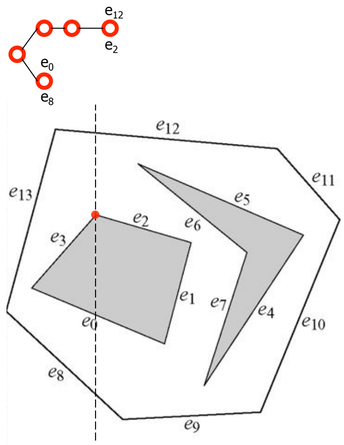


L:{e_8,e_0,e_3,e_{13}}
e_{UPPER}: e_{13}
e_{LOWER}: e_8

Cell decomposition

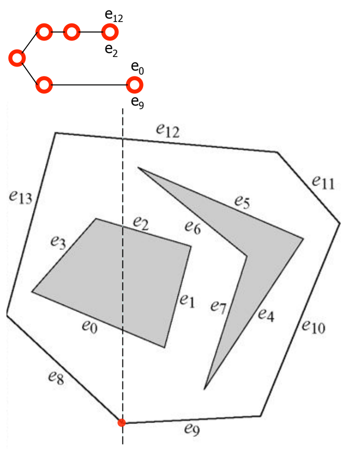


L:{e_8,e_0,e_3,e_{12}}
e_{UPPER}: -
e_{LOWER}: e_3

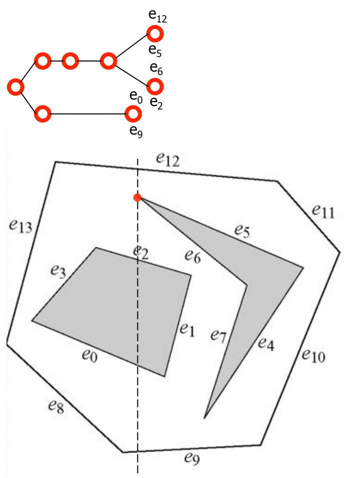


L:{e_8,e_0,e_2,e_{12}}
e_{UPPER}: e_{12}
e_{LOWER}: e_0

Cell decomposition



L:{e_9,e_0,e_2,e_{12}}
e_{UPPER}: e_0
e_{LOWER}: -

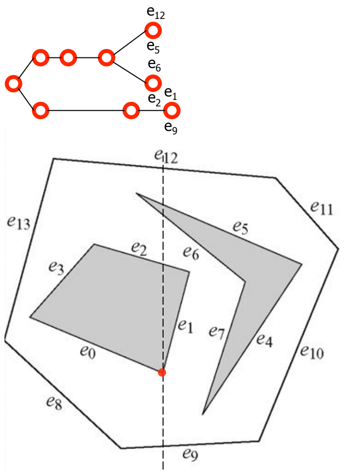


L:{e_9,e_0,e_2,e_6,e_5,e_{12}}
e_{UPPER}: e_{12}
e_{LOWER}: e_2

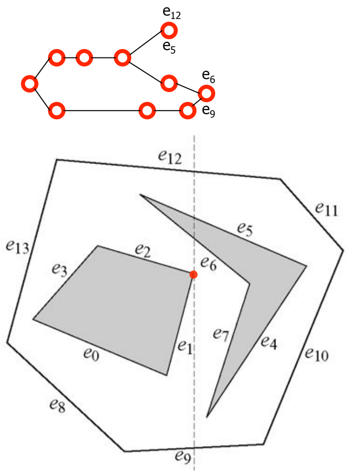
Autonomous Robots

UdG

Cell decomposition



L:{e_9,e_1,e_2,e_6,e_5,e_{12}}
e_{UPPER}: e_2
e_{LOWER}: e_9

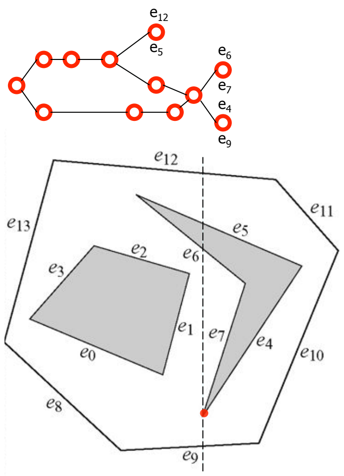


L:{e_9,e_6,e_5,e_{12}}
e_{UPPER}: e_6
e_{LOWER}: e_9

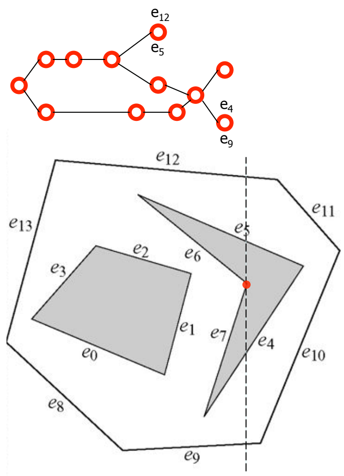
Autonomous Robots

UdG

Cell decomposition



L:{e_9,e_4,e_7,e_6,e_5,e_{12}}
e_{UPPER}: e_6
e_{LOWER}: e_9

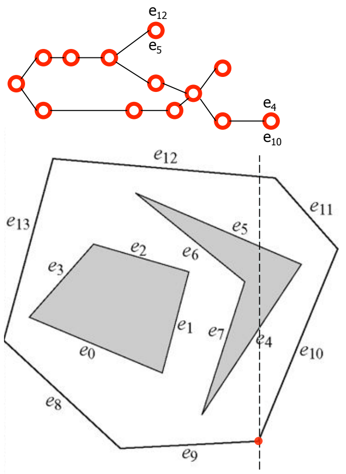


L:{e_9,e_4,e_5,e_{12}}
e_{UPPER}: e_5
e_{LOWER}: e_4

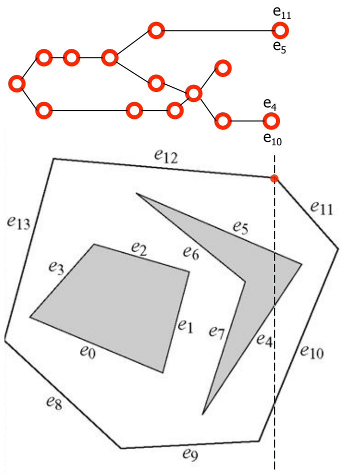
Autonomous Robots

UdG

Cell decomposition



L:$\{e_{10}, e_4, e_5, e_{12}\}$
e_{UPPER}: e_4
e_{LOWER}: -

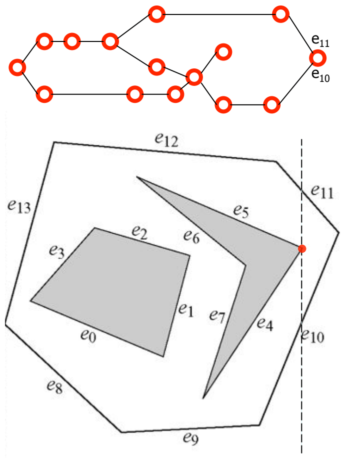


L:$\{e_{10}, e_4, e_5, e_{11}\}$
e_{UPPER}: -
e_{LOWER}: e_5

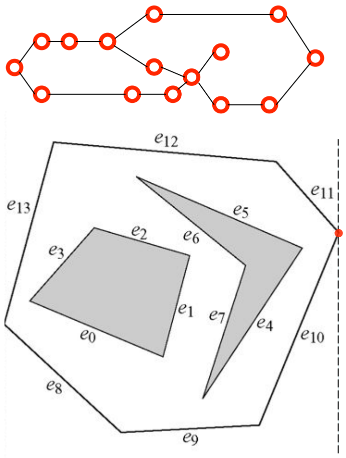
Autonomous Robots

UdG

Cell decomposition



L:$\{e_{10}, e_{11}\}$
e_{UPPER}: e_{11}
e_{LOWER}: e_{10}

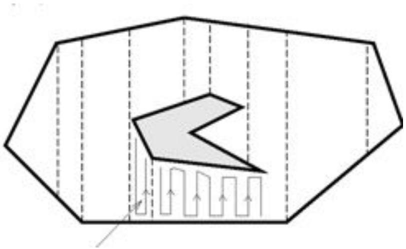


L:{}
e_{UPPER}: -
e_{LOWER}: -

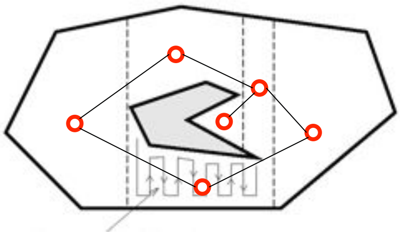
Cell decomposition

Boustrophedon decomposition

- Similar than trapezoidal decomposition but only vertices at which vertical line can be extended up and down are considered.
- Cells are bigger, not trapezoidal.
- Used for coverage path planning (i.e. cleaning robots).
- A lawnmower trajectory is followed inside each cell.



Trapezoidal decomposition



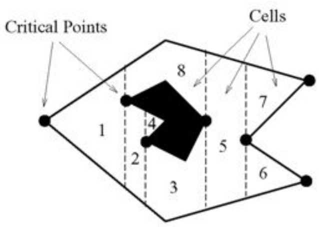
Boustrophedon decomposition

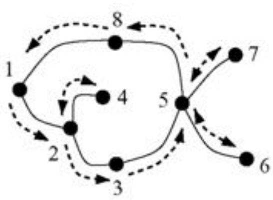
Autonomous Robots UdG

Cell decomposition

Boustrophedon decomposition

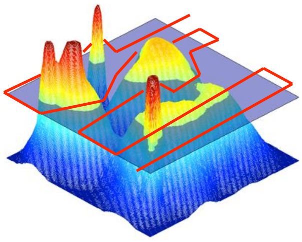
- Once the graph is generated, an exhaustive walk is first determined (depth-first search algorithm).
- Then, explicit robot motions are determined within each cell: straight lines separated by one robot width and short segments connecting them.

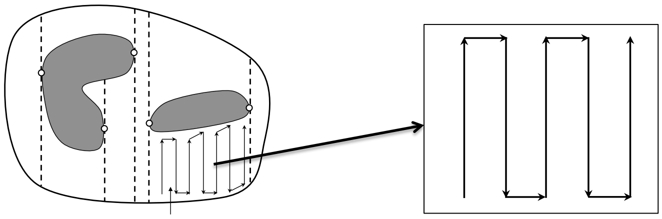




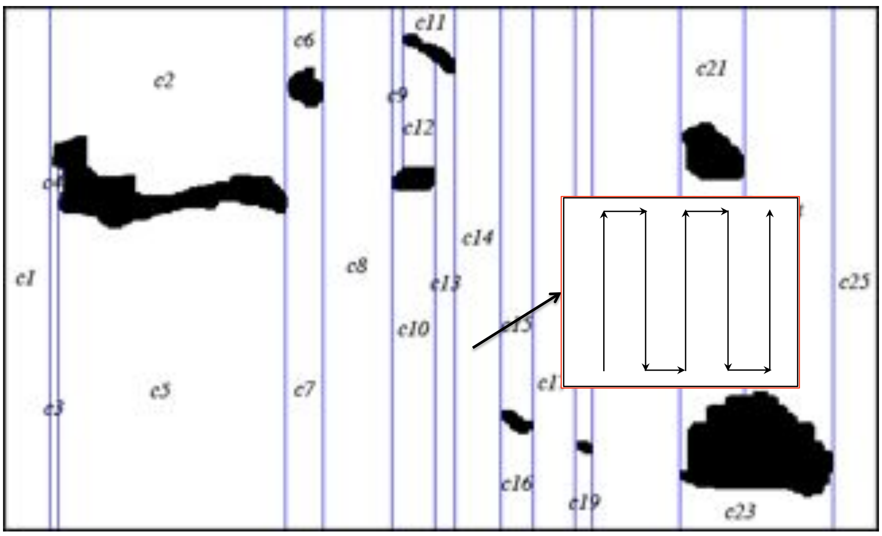
Autonomous Robots UdG — *Research project: AUV coverage path planning*

→ Coverage path planning **from bathymetric maps for surveying trajectories**. Cell decomposition.



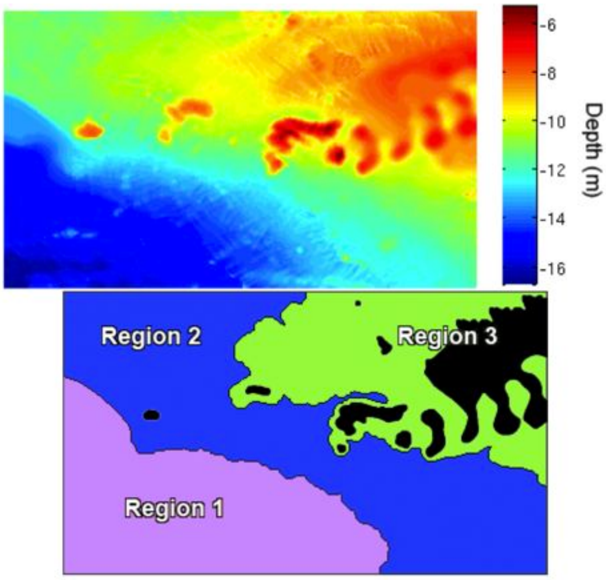


Autonomous Robots UdG — *Research project: AUV coverage path planning*



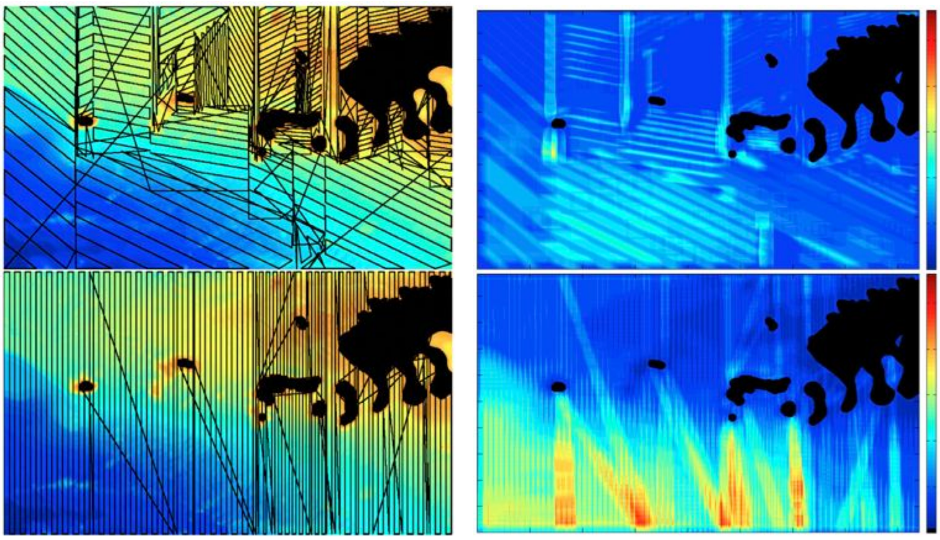
Autonomous
UdG Robots

Research project: AUV coverage path planning



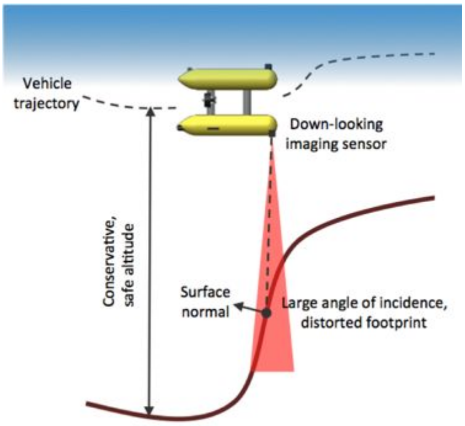
Autonomous
UdG Robots

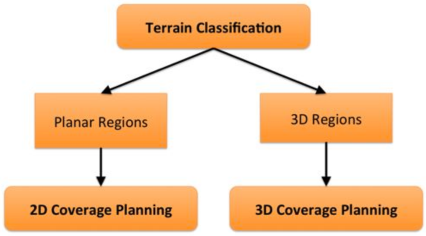
Research project: AUV coverage path planning

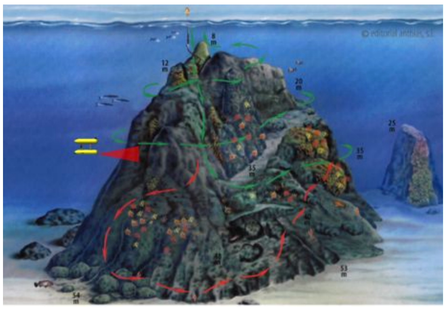


Autonomous Robots UdG

Research project: AUV coverage path planning



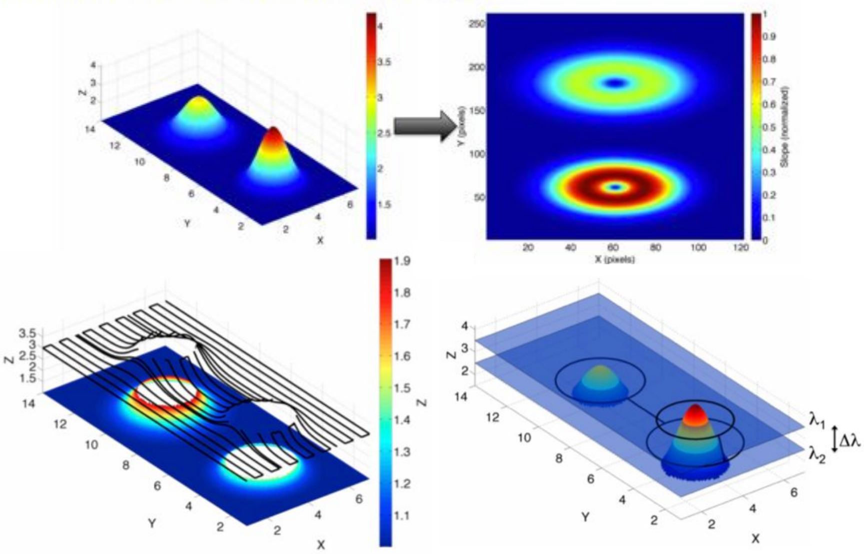




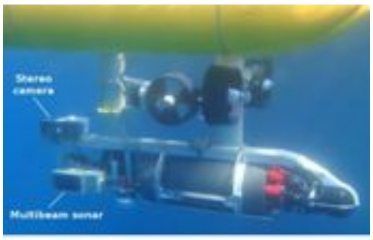
Autonomous Robots UdG

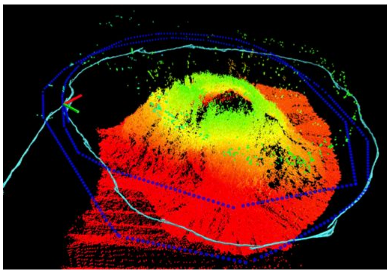
Research project: AUV coverage path planning

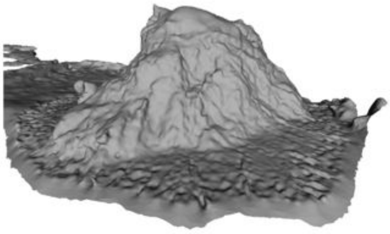
Identification of 2D and 3D Regions



Autonomous Robots **Research project: AUV coverage path planning**

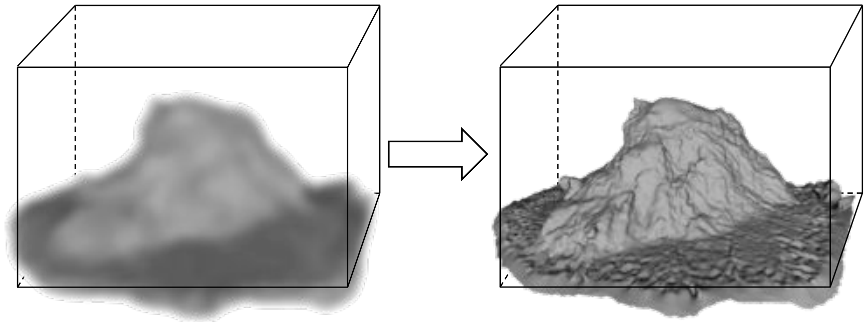






Autonomous Robots **Research project: AUV coverage path planning**

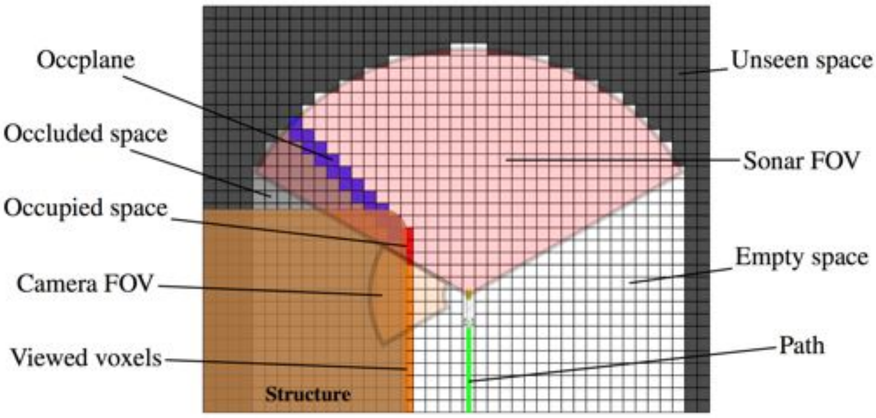
Research project: Online view planning



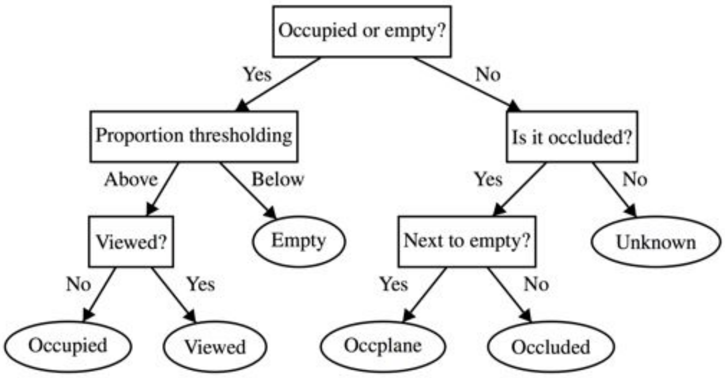
Mapping unknown structures without having prior information

Research project: Online view planning

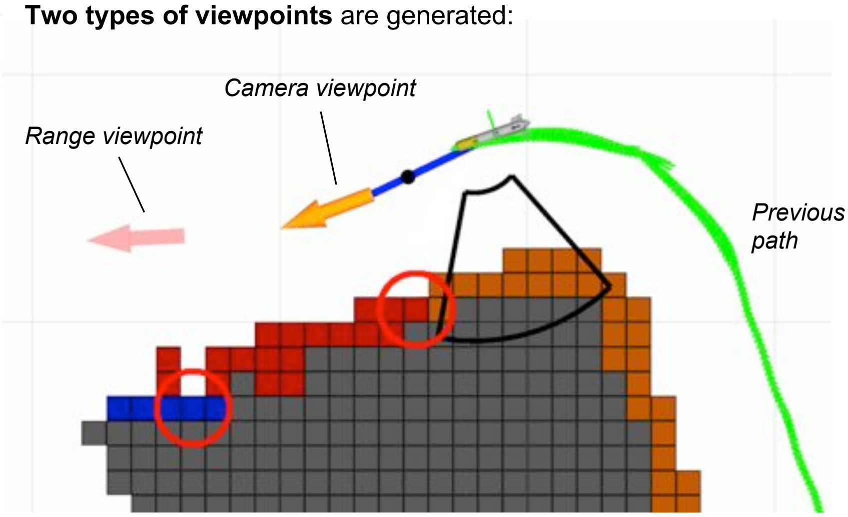
- World is represented using a **labeled 2D grid map**:



- Voxel **labeling strategy**:



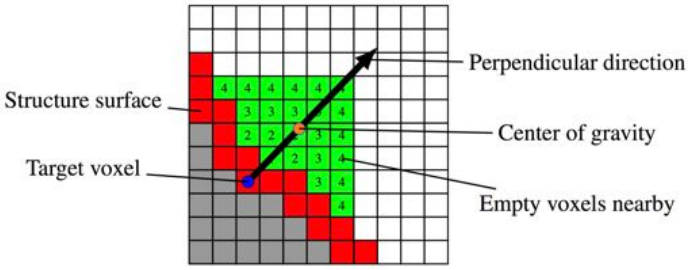
- **Two types of viewpoints** are generated:



UdG Autonomous Robots

Research project: Online view planning

- **Surface normal** computation:



- **Viewpoint selection** based on **distance and orientation**
- Automatic **sonar beam orientation**

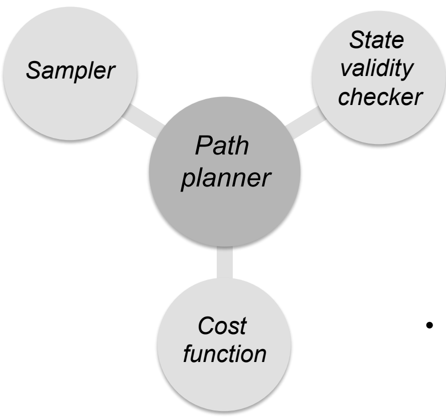
174

UdG Autonomous Robots

Research project: Online view planning

- To plan safe paths that allow the vehicle to achieve desired views, the **Open Motion Planning Library (OMPL)** has been used:

OMPL

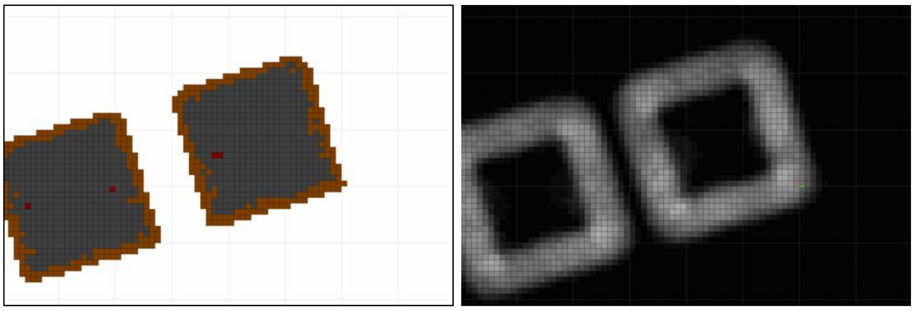


- *Final solution uses RRT* sampling based planner in R2 space*

175

Research project: Online view planning

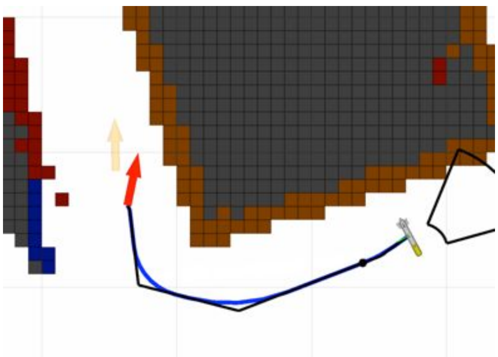
- The **cost of a path** corresponds to the **integral of the risk** function along the path:



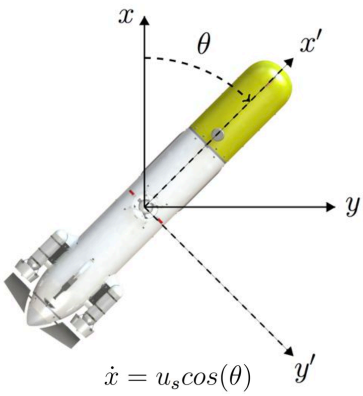
Risk map representation. Comparison between real map (left) and its corresponding risk map representation (right)

Research project: Online view planning

- **Path smoothing** is applied:



Example of path smoothing. Original path (black) and smoothed path (blue)



$$\dot{x} = u_s cos(\theta)$$

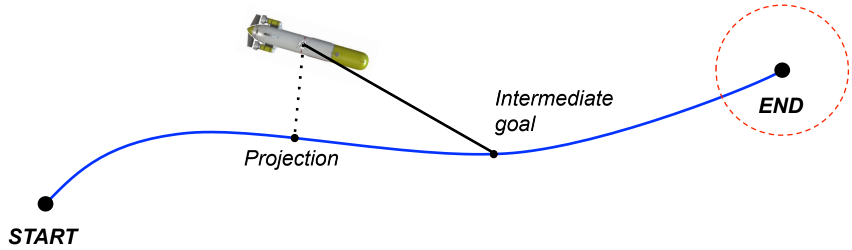
$$\dot{y} = u_s sin(\theta)$$

$$\dot{\theta} = u_y$$

Top view of the robot. World and robot coordinate frames

Autonomous Robots UdG — Research project: Online view planning

- ***Line Of Sight (LOS)*** algorithm:



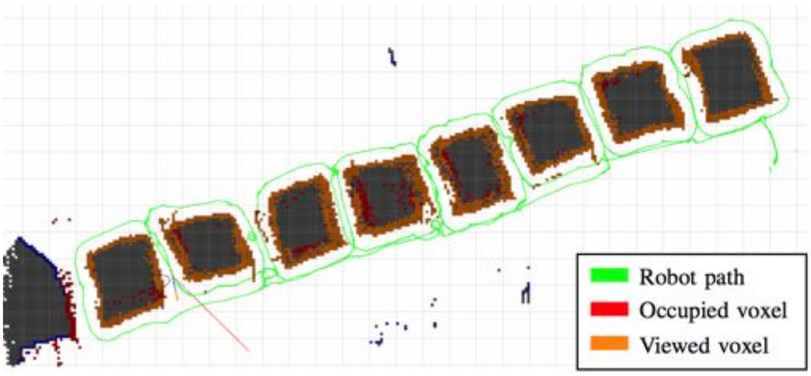
Line Of Sight algorithm: the vehicle orients and moves towards an intermediate goal

178

Autonomous Robots UdG — Research project: Online view planning

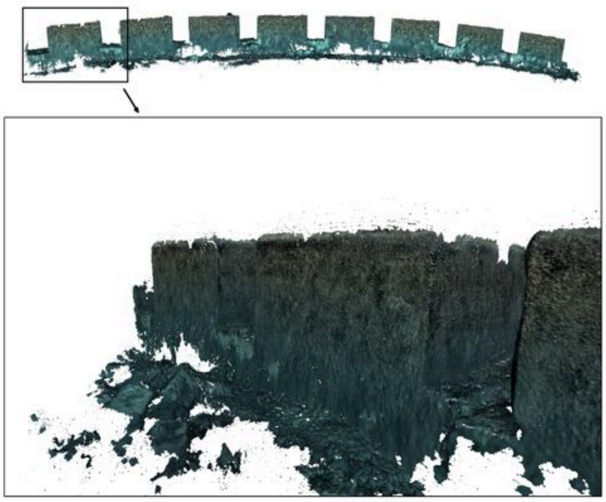
Research project: Online view planning

Autonomous Robots UdG



Research project: Online view planning

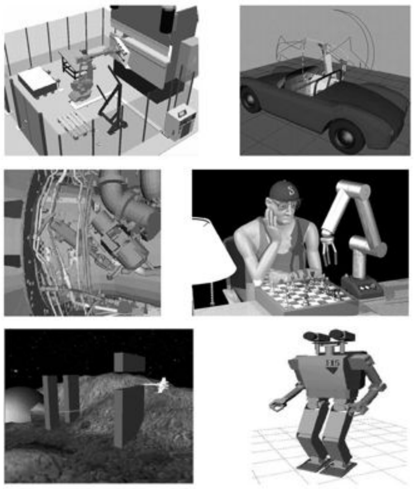
Autonomous Robots UdG



UdG **A**utonomous **R**obots

Sampling-based algorithms

- For problems with a lot of Degrees of Freedom or constraints (kinematic and dynamic).
- Instead of finding an optimal solution considering the whole environment, only few samples are considered.
- Each sample is a robot configuration.
- Solution to path planning will be a sequence of connected samples which all bellong to Q_{free} and connect the start and goal positions.
- A procedure is used to determine if a configuration is in Q_{free} or not.
- Algorithms can also guarantee the finding of the solution (completeness), they are probabilistic completeness.

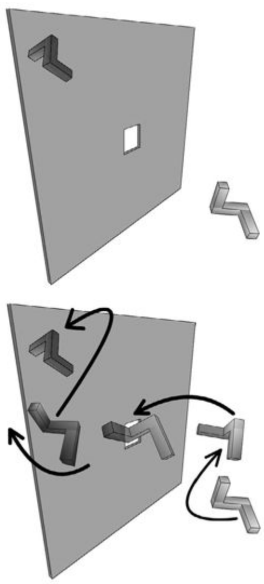


UdG **A**utonomous **R**obots

Sampling-based algorithms

Probabilistic RoadMap planner

- It is a multiple-query planner that creates a roadmap in Q_{free}.
- Coarse sampling using a uniform random distribution is used to obtain the nodes of the roadmap.
- The edges between nodes are planned, by a local planner, with fine sampling to ensure that all configurations belong to Q_{free}.
- Phases:
  - Learning phase, to create the roadmap.
  - Query phase, to plan particular paths between a start and a goal configurations.
- Roadmap is represented by a graph G=(V,E); V: vertices or nodes; E: edges generated by the local planner that correspond to a collision-free path from q_1 to q_2. Simplest form of the local planner: the straight line.
- In the query phase, q_{init} and q_{goal} are connected to two nodes q' and q'' respectively. The planner searches G for connecting q' and q'', and generates the path.



Sampling-based algorithms

Algorithm 6: Roadmap Construction

```
Input:
n : number of nodes to put in the roadmap
k : number of closest neighbors to examine for each configuration
Output:
A roadmap G = (V, E)
V ← Ø
E ← Ø
while |V| < n do
  repeat
    q ← a random configuration in Q
  until q is collision-free
  V ← V ∪{q}
end while
for all q ∈ V do
  N_q ← the k closest neighbors of q chosen from V according to dist
  for all q' ∈ N_q do
    if (q, q') ∉ E and Δ(q, q') ≠ NIL then
      E ← E ∪{(q, q')}
    end if
  end for
end for
```

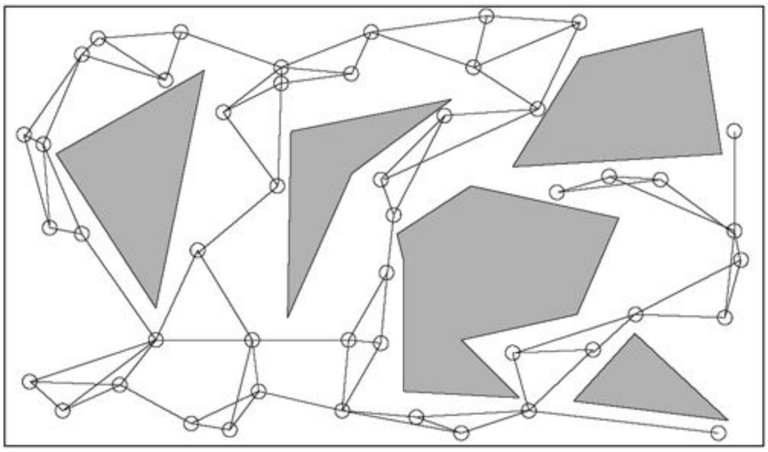
- *Being Δ the local planner and dist a metric function to measure distance between two configurations*

Autonomous
UdG Robots

Sampling-based algorithms

Probabilistic RoadMap planner

- Roadmap in a 2D space, local planner: straight line planner, n=50, k=3.



Autonomous
Robots

Algorithm 7: Solve Query Algorithm

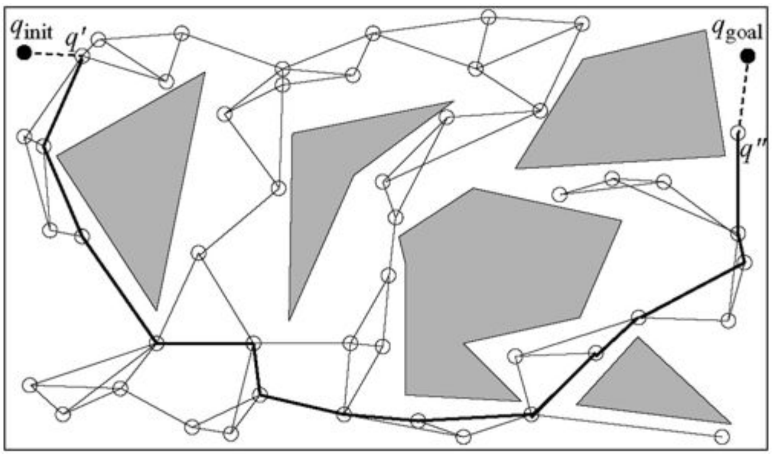
```
Input:
q_init: the initial configuration
q_goal: the goal configuration
k: the number of closest neighbors to examine for each configuration
G = (V, E): the roadmap computed by algorithm 6
Output:
A path from q_init to q_goal or failure
N_qinit ← the k closest neighbors of q_init from V according to dist
N_qgoal ← the k closest neighbors of q_goal from V according to dist
V ← {q_init} ∪ {q_goal} ∪ V
set q' to be the closest neighbor of q_init in N_qinit
repeat
  if Δ(q_init, q') ≠ NIL then
    E ← (q_init, q') ∪ E
  else
    set q' to be the next closest neighbor of q_init in N_qinit
  end if
until a connection was succesful or the set N_qinit is empty
set q' to be the closest neighbor of q_goal in N_qgoal
repeat
  if Δ(q_goal, q') ≠ NIL then
    E ← (q_goal, q') ∪ E
  else
    set q' to be the next closest neighbor of q_goal in N_qgoal
  end if
until a connection was succesful or the set N_qgoal is empty
P ← shortest path(q_init, q_goal, G)
if P is not empty then
  return P
else
return failure
end if
```

Autonomous
Robots

Sampling-based algorithms

Probabilistic RoadMap planner

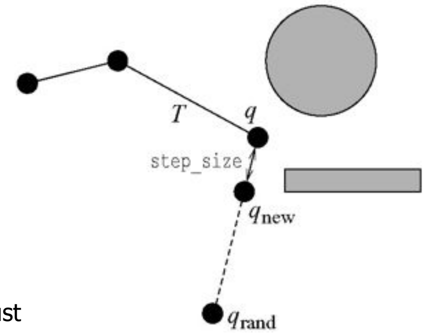
- Query solved with a graph-search algorithm (i.e. A*)



Sampling-based algorithms

Single-Query Sampling-Based Planners

- Different approaches to build directly, without the roadmap, the path between two configurations.
- For large number of degrees of freedom, or kinematic and dynamic constraints.
- **RRT algorithm** (Rapidly-Exploring Random Trees)
  - Most well known sampling algorithm
  - 2 trees, T_{init} and T_{goal}, grow rooted at q_{init} and q_{goal} respectively.
  - A random configuration q_{rand} is sampled uniformly in Q_{free}.
  - The nearest configuration q_{near} is found, and a new configuration q_{new} is generated at a step_size distance towards q_{rand}.
  - q_{new} and the edge (q_{near}, q_{new}) must belong to Q_{free}.

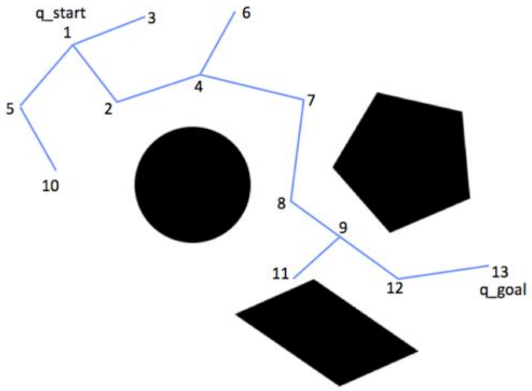


Autonomous
UdG Robots

Sampling-based algorithms

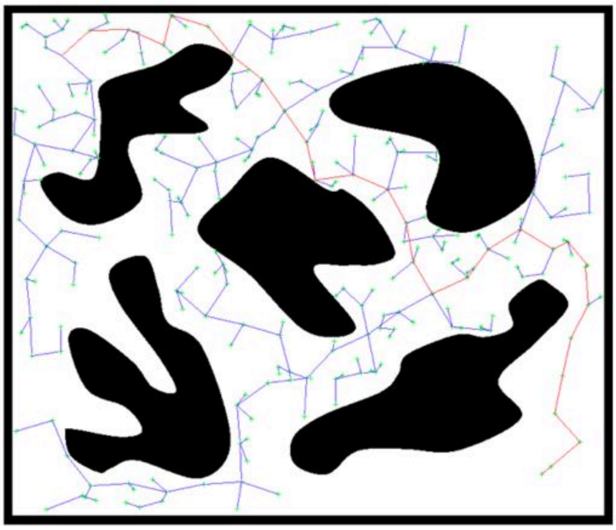
Single-Query Sampling-Based Planners

- **RRT algorithm** (Rapidly-Exploring Random Trees)



Single-Query Sampling-Based Planners

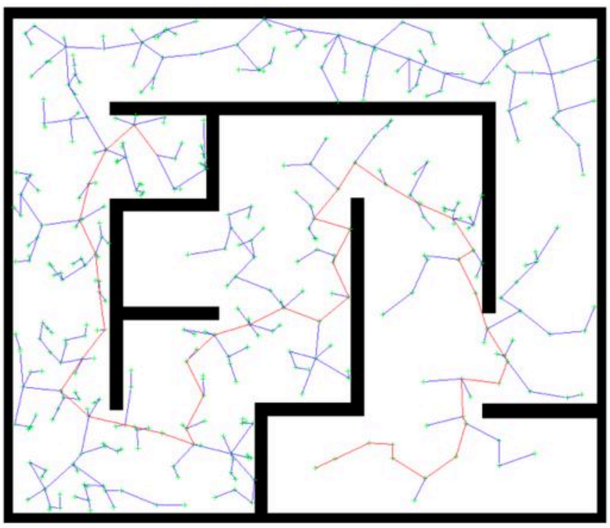
- **RRT algorithm** (Rapidly-Exploring Random Trees)



Autonomous
UdG Robots
Sampling-based algorithms

Single-Query Sampling-Based Planners

- **RRT algorithm** (Rapidly-Exploring Random Trees)



Sampling-based algorithms

Algorithm 10: Build RRT Algorithm

```
Input:
q0: the configuration where the tree is rooted
n: the number of attempts to expand the tree
Output:
A tree T = (V, E) that is rooted at q0 and has ≤ n configurations
V ← {q0}
E ← Ø
for i = 1 to n do
   qrand ← a randomly chosen free configuration
   extend RRT (T, qrand)
end for
return T
```

Algorithm 11: Extend RRT Algorithm

```
Input:
T = (V, E): an RRT
q:a configuration toward which the tree T is grown
Output:
A new configuration qnew toward q,or NIL in case of failure
qnear ← closest neighbor of q in T
qnew ← progress qnear by step_size along the straight line in Q between qnear and qrand
if qnew is collision-free then
   V ← V ∪ {qnew}
   E ← E ∪ {(qnear, qnew)}
   return qnew
end if
return NIL
```

Autonomous Robots UdG

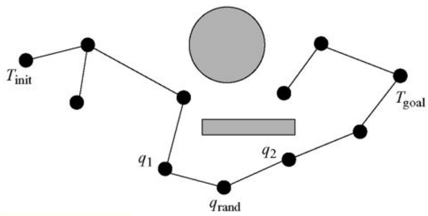
Sampling-based algorithms

RRT algorithm

- The sampling is usually guided towards q_{goal} (or q_{init}) to improve the efficiency:
  - with p probability: $q_{rand} = q_{goal}$
  - with (1-p) probability: qrand = random uniform distribution
- Merging of trees, T_{init} and T_{goal},

Algorithm 13: Merge RRT Algorithm

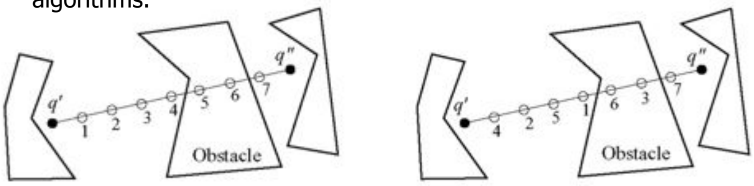
```
Input:
T1: first RRT
T2: second RRT
ℓ: number of attempts allowed to merge T1 and T2
Output:
merged if the two RRTs are connected to each other; failure otherwise
for i = 1 to ℓ do
   qrand ← a randomly chosen free configuration
   qnew,1 ← extend RRT (T1, qrand)
   if qnew,1 ≠ NIL then
      qnew,2 ← extend RRT (T2, qnew,1)
      if qnew,1 = qnew,2 then
         return merged
      end if
      SWAP(T1, T2)
   end if
end for
return failure
```



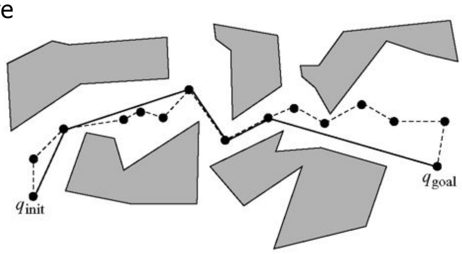
Sampling-based algorithms

Sampling-based algorithms implementation details

- **Straight-line local planner** implementation:
  - Discretization of the line according to a small step size.
  - Collision checking strategies: incremental (left) and subdivision (right) algorithms.



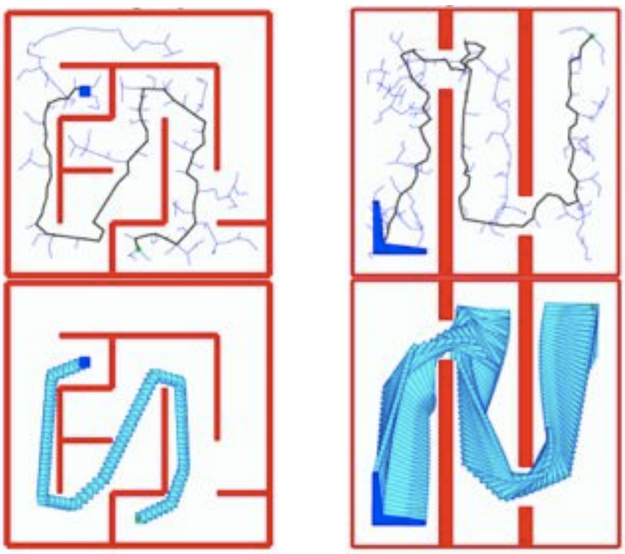
- **Postprocessing queries** to improve shortness and smoothness.
  - Greedy approach: connect q_{goal} from q_{init}, if it fails try from a closer position until it connects. Once q_{goal} connected start again with its directly connected position.



Autonomous
UdG Robots

Sampling-based algorithms

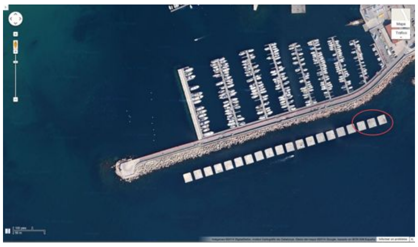
RRT algorithm, examples



UdG Autonomous Robots

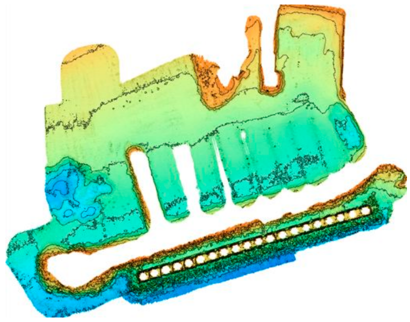
Research project: AUV motion planning

- Solving start-to-goal queries to move through a breakwater



Bathymetry

2.5D elevation map using a multibeam profiler sonar (Sant Feliu de Guixols).



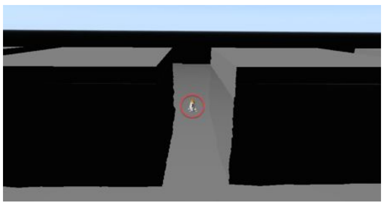
Breakwater

A series of concrete blocks (14.5mx12m), separated by four-meter gap. Average depth of 7m.

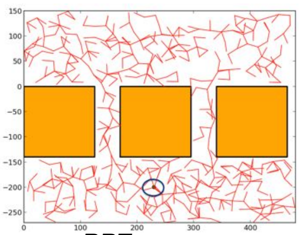
UdG Autonomous Robots

Research project: AUV motion planning

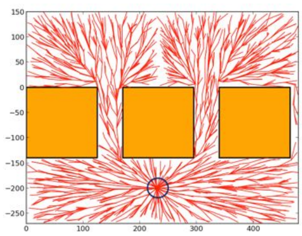
- Offline planning



UnderWater Simulator (UWSim) *[Prats-IROS12]*



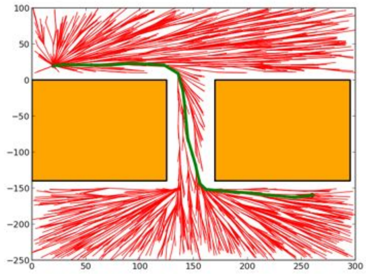
RRT *[LaValle96]*

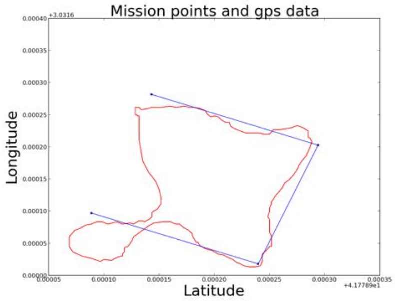


RRT* *[Karaman10]*

Autonomous Robots **Research project: AUV motion planning**

- Motion constraints? Non-holonomic vehicle.

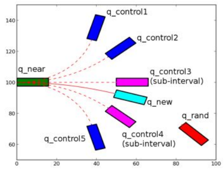


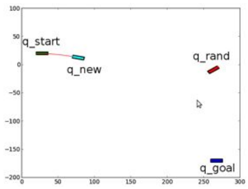


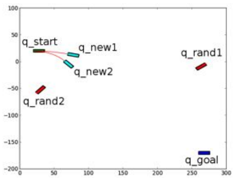
Autonomous Robots **Research project: AUV motion planning**

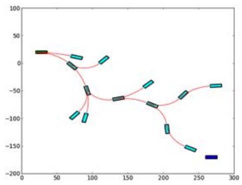
- Motion constraints: kinodynamic motion planning

$$\dot{x} = f(x,u) \qquad \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}u.cos(\psi)\\ u.sin(\psi)\\ r\end{bmatrix}$$





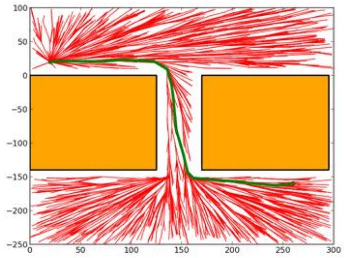


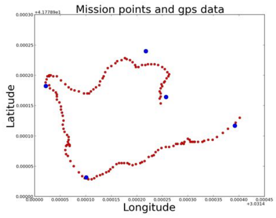


UdG **A**utonomous **R**obots

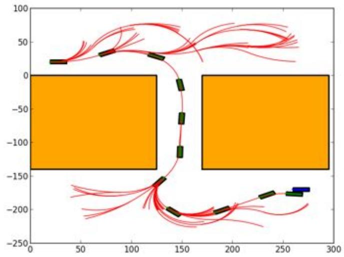
Research project: AUV motion planning

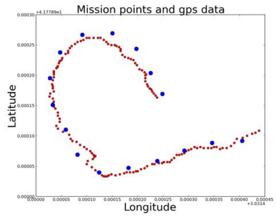
without constraints





with constraints





UdG **A**utonomous **R**obots

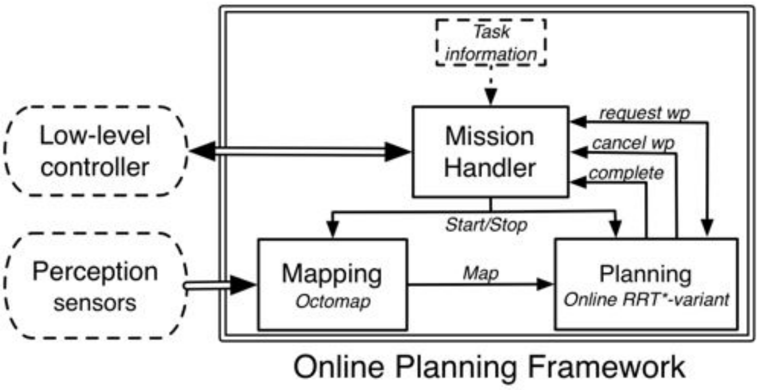
Research project: AUV motion planning

Research project: AUV motion planning

Online path planning?

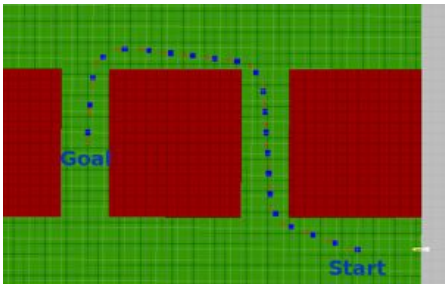
Autonomous Robots UdG

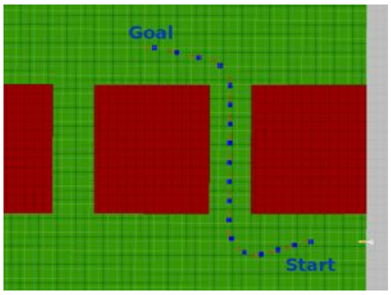
Research project: AUV motion planning



Autonomous UdG Robots

Research project: AUV motion planning





Motion or differential constraints.

- *Dubins curves (alternative)*

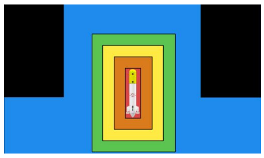
$$q = [x, y, \psi] \rightarrow q \in SE(2) \rightarrow q \in \Re^2 \times S$$

$$\dot{q} = f(q, u)$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v.\cos(\psi) \\ v.\sin(\psi) \\ r \end{bmatrix}$$

Collision risk + path length

- *Risk zones*
- *Integral of risk with respect to distance*



Autonomous UdG Robots

Research project: AUV motion planning

[Hernández-IROS16]

- *Motion Constraints.*
- *Optimization function: length and risk associated to a path*
- *Opportunistic collision and risk checking.*
- *Reuse of last best known solution.*

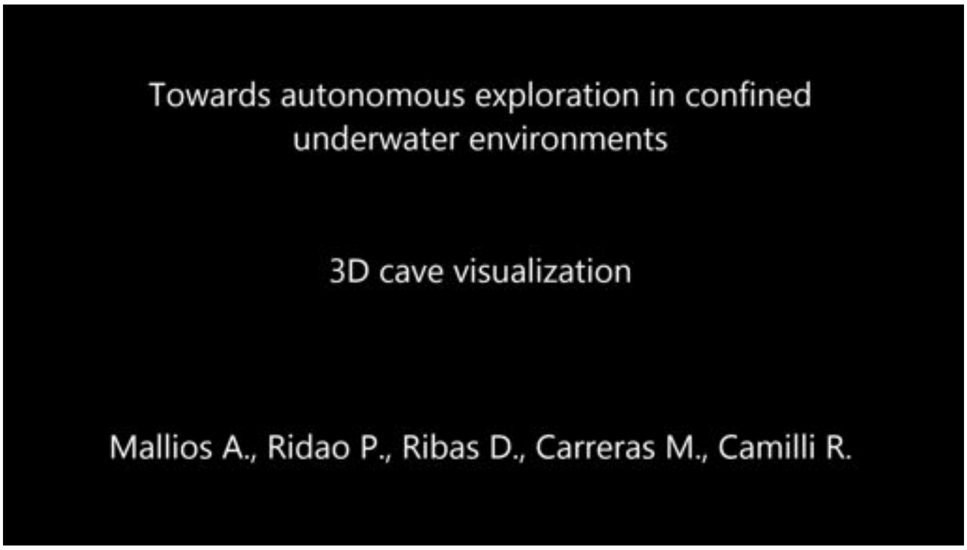
UdG **A**utonomous **R**obots

Research project: AUV motion planning

[Hernández, Istenič - Sensors16]

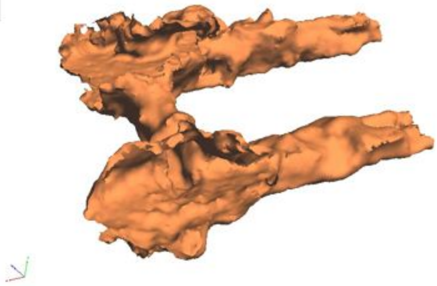
UdG **A**utonomous **R**obots

Research project: towards 3D motion planning



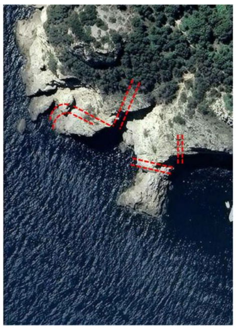
Autonomous Robots UdG

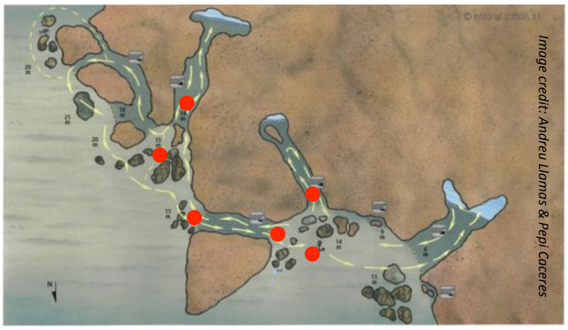
Research project: towards 3D motion planning



Autonomous Robots UdG

Research project: towards 3D motion planning





HIL Simulation: Autonomous Guidance In a Cave

- *No A Priori Map Used*
- *Navigation towards a Goal Waypoint out of the cave*
- *Rotating Forward Looking Multibeam*
- *Real Time Path Planning under Kinematic Constrains*
- *Real Time Octomap Mapping*
- *Autonomous Guidance*

Open Motion Planning Library (OMPL)

- *Consist of different sampling-based motion planning algorithms.*
- *Not collision checking or visualization tools included.*
- *Not designed for any specific scenario, collision checking done with user-defined routines.*
- *Support for kinodynamic motion planning.*
- *Support for commonly used state spaces (SE(2), SE(3), R^n, etc.).*
- *Extensible to user-defined state spaces.*

Taken from OMPL website: http://ompl.kavrakilab.org/

Bibliography

- H. Choset et al. "**Principles of Robot Motion**", MIT Press, 2005.
- Ronald C. Arkin, "**Behavior-Based Robotics**", MIT Press, 1998.
- Bekey, George A., "**Autonomous Robots: From Biological Inspiration to Implementation and Control**", MIT Press, 2005
- Murphy, Robin, "**Introduction to AI robotics**". Cambridge [etc.] : MIT Press, cop. 2000
- Dudek, Gregory, "**Computational principles of mobile robotics"**. Cambridge : Cambridge University Press, 2000
- R. Siegwart and I.R. Nourbakhsh, "Introduction to **Autonomous Mobile Robots**", MIT Press, 2004.