ROS-I Basic Training “Mobile Manipulation”

Navigation

M. Stuettgen, N. Limpert
Mobile Autonomous Systems and Cognitive Robotics Institute (MASCOR)

August 9, 2017

Learning Objectives

You will

- get to know an overview of what navigation is
- see how to find global path
- learn how to make use of ROS move_base

Outline

- Foundations
- Path Planning

Configuration Space

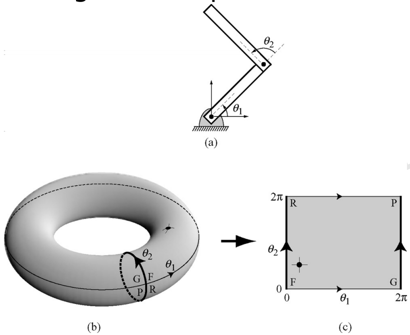
Configuration / Configuration Space

A configuration is a full specification
of the position of every point of the system.

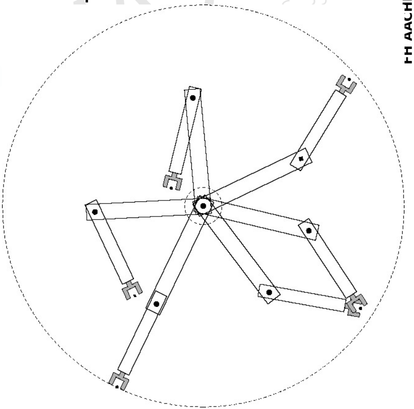
The configuration space (C-space) then is
the space of all possible configurations.

Configuration Space and Workspace of a two-link Robot

Configuration Space:

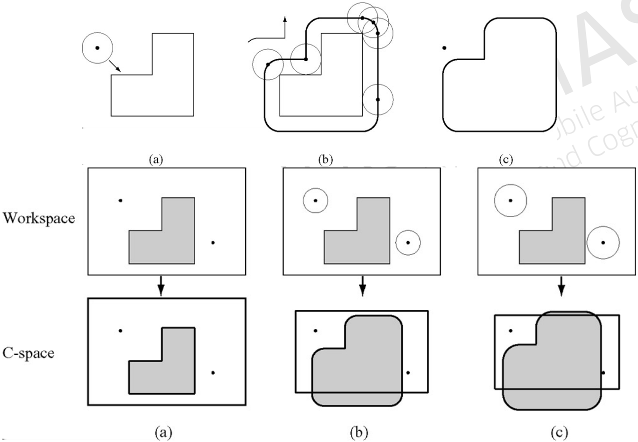


Workspace:



Source: (Choset et al., 2005) ©MIT Press

Example of a Round Robot

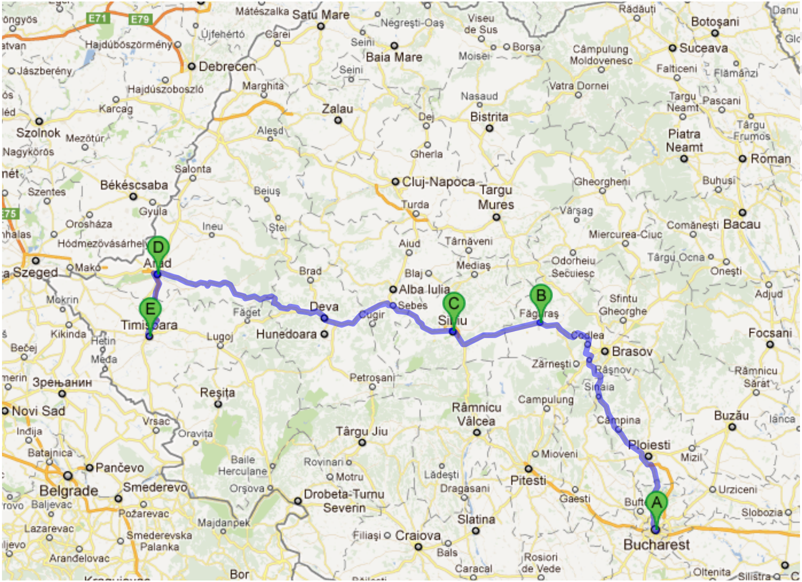


Source: (Choset et al., 2005) ©MIT Press

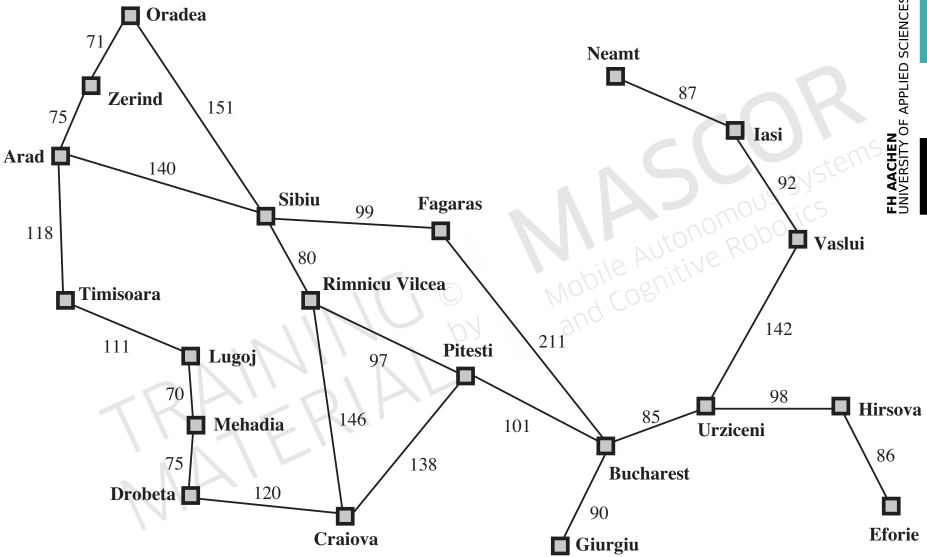
Outline

- Foundations
- Path Planning

Directions to Timisoara, Romania
635 km – about **9 hours 3 mins**



FH AACHEN
UNIVERSITY OF APPLIED SCIENCES


Oradea
71
Zerind
151
75
Arad
140
Neamt
87
Iasi
92
Sibiu
99
Fagaras
Vaslui
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
Craiova
90
Giurgiu
Eforie

What is the problem about? (from an agent's perspective)

in general

- start / goal
- map of the environment
- path costs
- search strategy
- sensory information

What does an agent need to do to get from Bucharest to Timisoara?

- map, actions, sensing

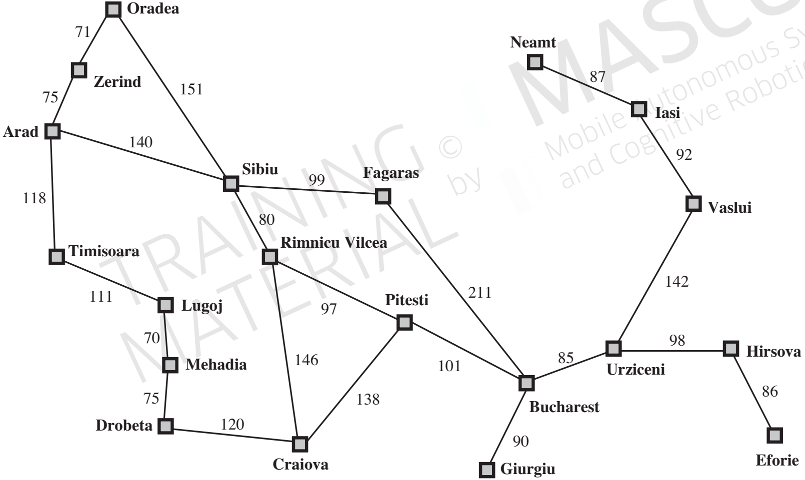
Solving a (search) problem

- Search
- Solution
- Execute solution (open-loop)

Optimality of the solutions

The **solution** of a problem is a sequence of actions that leads from the initial state to the goal state.
A solution is an **optimal solution**, if it has the lowest path costs among all possible solutions.

From Bucharest to Timisoara

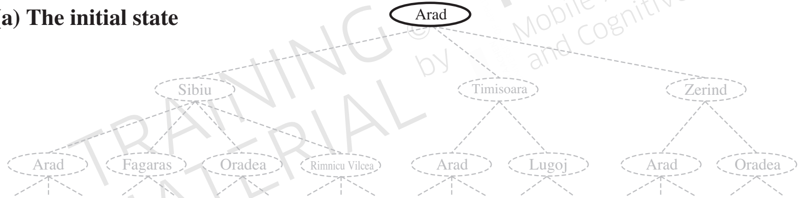


Summary: Problem formulation

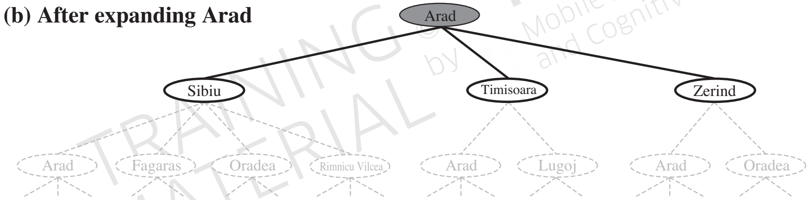
- Initial state
- Actions
- Executability of actions
- Description of actions (transition model/successor)
  State space = initial state + actions + transition model
- Path = sequence of states, which are reachable by actions
- Goal test
- Costs: path costs, step costs, optimality of a solution

Route Planning: Finding a Path in the Search Tree

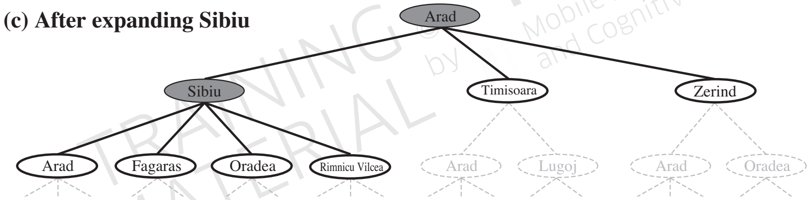
(a) The initial state



Route Planning: Finding a Path in the Search Tree



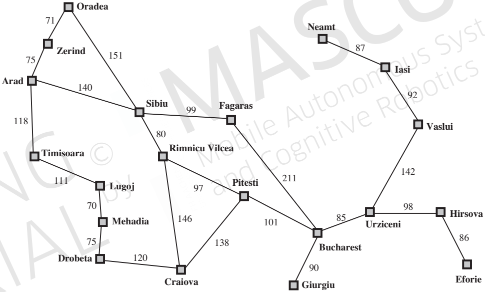
Route Planning:
Finding a Path in the Search Tree



Search Trees

Root:	Initial state
Node:	State in the state space of the problem
Expanding nodes	Actions of the problem define successor nodes, e.g. parent node $In(Arad)$ expands child nodes $\{In(Sibiu), In(Timisoara), In(Zerind)\}$
Frontier/Open list:	unexpanded leaves of the search tree
Search strategy:	Order in which nodes in *Frontier* are expanded Search methods differ in their search strategy in general
Loops in the tree:	can lead to infinite paths

A Heuristic for the Romania Route



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Heuristik h_{SLD}
straight line distance
to the goal

A*-Search

- Best-first search method
- Selecting the next node:

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

with

$$\begin{aligned} g(n) &= \text{actual cost to get to } n \\ h(n) &= \text{estimated cost of cheapest path} \\ &\ \text{via } n \text{ to the goal} \end{aligned}$$

- A* is optimal and complete if h is admissible or consistent resp.
- A* is identical to UCS but uses $g + h$ instead of only g.

Optimality Requirements for A*

Admissibility

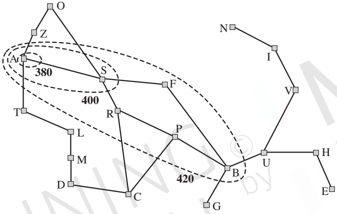
$h(n)$ is admissible, if for every n holds:

$h(n)$ never overestimates the costs to the goal

(Then f never overestimates the real costs,
because $f(n) = h(n) + g(n)$ with $g(n)$ being the actual cost to n.

Example: straight line distance is admissible!

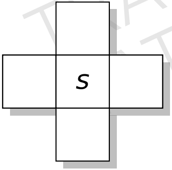
Expansion of nodes in A*



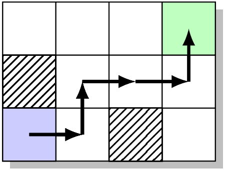
- Because the f-cost do not decrease, nodes are expanded along the f-cost-contour.
- The more precise the heuristic, the more the contour yields towards the goal. (UCS: concentric circles)
- All nodes with $f(n) < C^*$ are expanded.
- Some nodes on the goal contour are expanded ($f(n) = C^*$)
- complete, if finitely many nodes exist with $f(n) \leq C^*$
- A* is optimally efficient, i.e. no other search algorithm expands fewer nodes with the same heuristic.

A* for Path Planning on Grids

- Discretise state space (occupancy grid)
- $h(n, m) = \sqrt{(x_n - x_m)^2 + (y_n - y_m)^2}$
- $g(n) =$ path length $\equiv$ #cells in path
- neighbors of a cell s:



- paths:

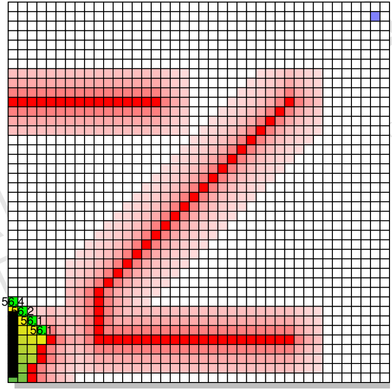


A* Algorithm

```
AStar ()
begin
    Initialize()
    while open ≠ ∅ do
        s ← open.Pop()
        if s = s_goal then
            return path
        closed ← closed ∪ {s}
        foreach s' ∈ nghbrs(s) do
            if s' ∉ closed then
                if s' ∉ open then
                    g(s') ← ∞
                    parent(s')=nil
                end
            end
            UpdateVertex(s, s')
        end
        return "no path found"
    end
end
```

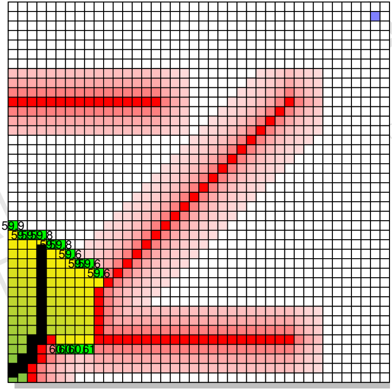
```
Initialize
begin
    g(s_start) ← 0
    parent(s_start) ← s_start
    open.Insert(s_start, f(s_start))
    closed ← ∅
end

UpdateVertex (s,s')
begin
    if g(s) + c(s, s') < g(s') then
        g(s') ← g(s) + c(s, s')
        parent(s') ← s
        if s' ∈ open then
            open.Remove(s')
        open.Insert(s',f(s'))
    end
end
```

Example I: A* I



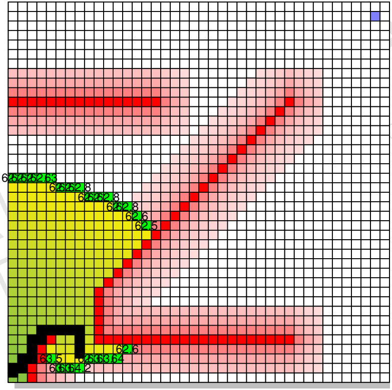
After 20 expanded cells

Example I: A* II



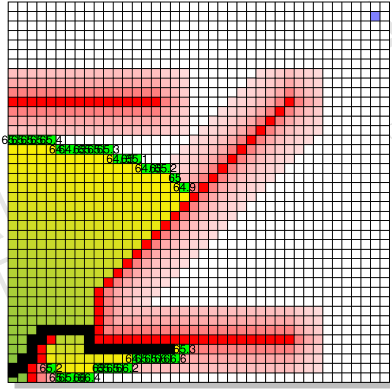
After 100 expanded cells

Example I: A* III



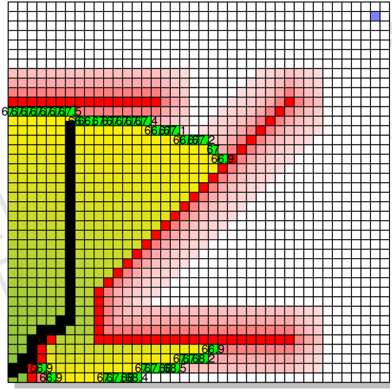
After 200 expanded cells

Example I: A* IV



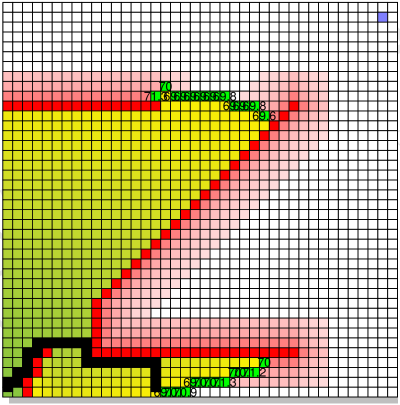
After 300 expanded cells

Example I: A* V



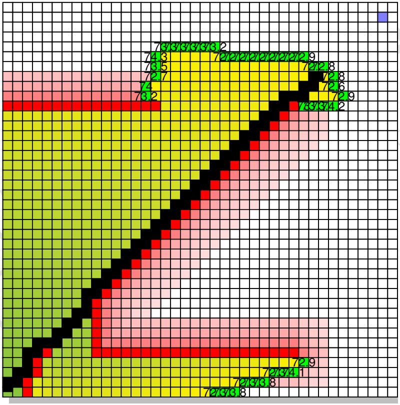
After 400 expanded cells

Example I: A* VI



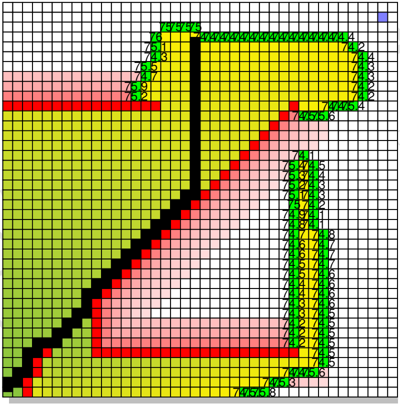
After 500 expanded cells

Example I: A* VII



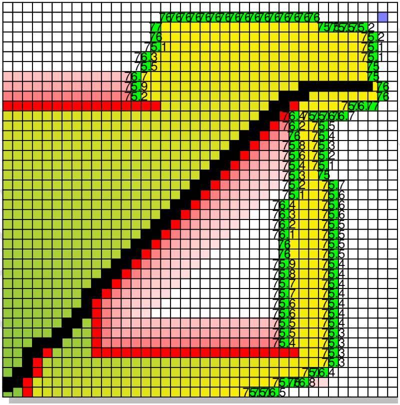
After 600 expanded cells

Example I: A* VIII



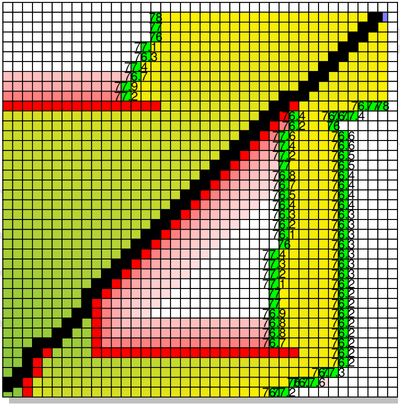
After 700 expanded cells

Example I: A* IX



After 800 expanded cells

Example I: A* X



After 899 expanded cells

Memory-bounded heuristic search methods

Problem with A*: Space Complexity!

- Iterative-deepening A* (IDA*)
  Use *f*-cost as a limit (not the depth)
- Memory-bounded A* (MA*)
- Simple MA* (SMA*)
- Recursive Best-First Search

Learning Objectives

You will learn

- how to set up and use the move base

ROS Move Base - I

- provides an implementation of an action
- attempts to reach a given goal in the world with a mobile base
- uses a global and local planner to accomplish its global navigation task
- manages communication between the components of the navigation stack

http://wiki.ros.org/move_base

ROS Move Base - II

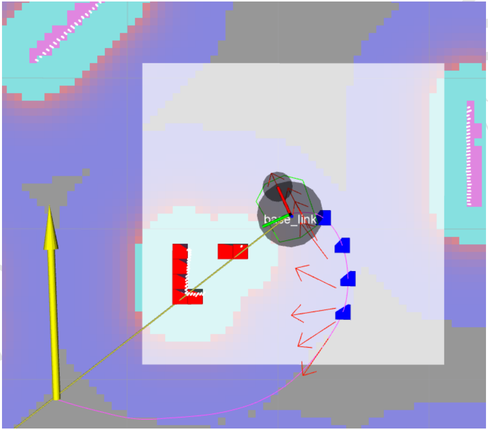


Figure: move base visualization in RVIZ

ROS Move Base - III

Properties

- Movement in 3D (x, y, θ)
- Implementation of an `ActionServer`
  - Send navigational goals as an action
  - Monitor execution feedback
    - robot pose w.r.t. global frame
  - Cancel execution
- Accepts messages of type `geometry_msgs/PoseStamped`

Basic dependencies

- Movement execution as accurate as possible
- Map source
- Correctly setup tf-tree
  - local frame on rotation point!
- Footprint definition
- Correct timing setups (expected update rates)

http://wiki.ros.org/move_base

ROS Move Base - IV

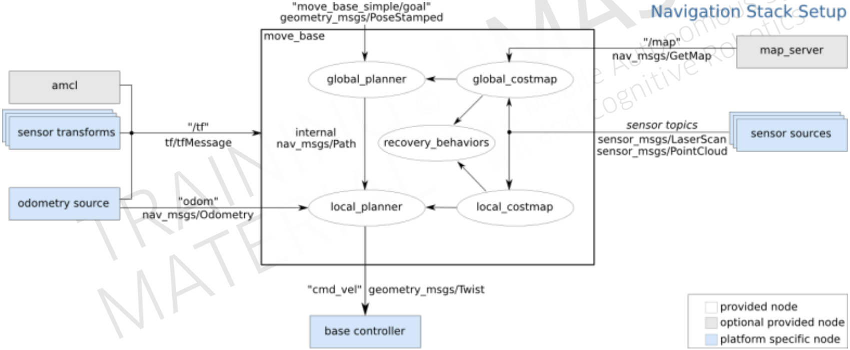
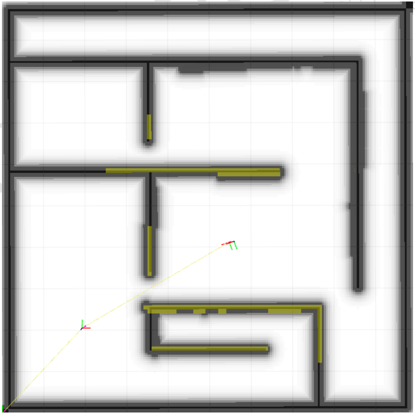


Figure: ROS navigation stack

ROS Move Base - Costmaps

Global and local costmaps

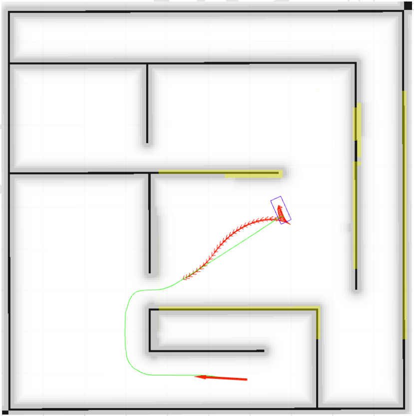
- Representation of occupancy grid
- Layer-based approach (static / obstacle / inflation)
- Occupancy grid representation for planners



ROS Move Base - Planning

Global and local planner

- Global: Plan for long distances
  - Supposed to be fast
- Local: Execute motions
  - Provide collision avoidance

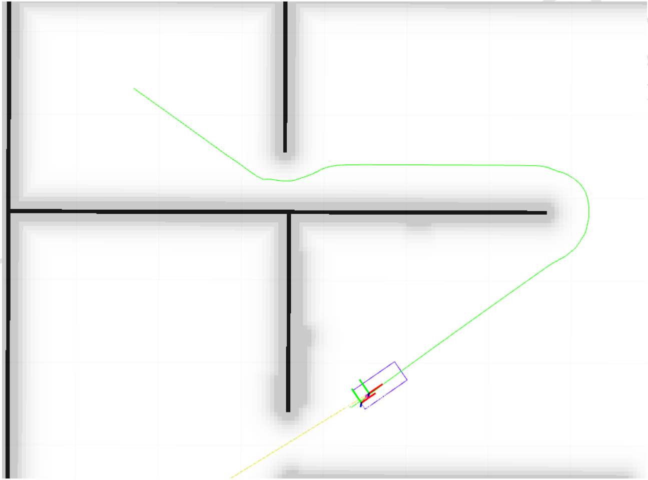


ROS Move Base - Global Planner I

- Global planner providing an implementation of Dijkstra's algorithm
- Graph-search based approach (global costmap)
- Does not take kinematic constraints into account
- Works out of the box with no further configuration

http://wiki.ros.org/global_planner

ROS Move Base - Global Planner II



ROS Move Base - Actionlib API

Task

Have an abstraction layer for navigation goals.

ROS Move Base - Actionlib API

Task

Have an abstraction layer for navigation goals.

- ActionServer implementation available for higher level applications
- Send a goal provided as `move_base_msgs/MoveBaseActionGoal`
- Cancel goal
- Get feedback (robot position)

ROS Move Base - Actionlib API

Task

Have an abstraction layer for navigation goals.

- ActionServer implementation available for higher level applications
- Send a goal provided as `move_base_msgs/MoveBaseActionGoal`
- Cancel goal
- Get feedback (robot position)

ROS Move Base - Configuration

Configuration is done in five yaml-files:

- costmap_common_params.yaml
- global_costmap_params.yaml
- local_costmap_params.yaml
- move_base_params.yaml
- teb_local_planner_params.yaml

http://wiki.ros.org/navigation/Tutorials/RobotSetup#Running_the_Navigation_Stack

ROS Move Base - Configuration

example costmap_common_params.yaml:

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan

laser_scan: {sensor_frame: frame_name, data_type: LaserScan,
  topic: topic_name, marking: true, clearing: true}
```

http://wiki.ros.org/navigation/Tutorials/RobotSetup#Running_the_Navigation_Stack

ROS Move Base - Configuration

example global_costmap_params.yaml:

```
global_costmap:
  global_frame: /map
  robot_base_frame: base_link
  update_frequency: 5.0
  static_map: true
```

http://wiki.ros.org/navigation/Tutorials/RobotSetup#Running_the_Navigation_Stack

ROS Move Base - Configuration

example local_costmap_params.yaml:

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

http://wiki.ros.org/navigation/Tutorials/RobotSetup#Running_the_Navigation_Stack

ROS Move Base - Configuration

example teb_local_planner_params.yaml:

```
TebLocalPlannerROS:
  max_vel_x: 0.6
  max_vel_y: 0.6
  max_vel_x_backwards: 0.6
  max_vel_theta: 1.0
  acc_lim_x: 1.0
  acc_lim_y: 1.0
  acc_lim_theta: 1.0

 footprint_model:
   type: "line"
   line_start: [-0.19, 0.0]
   line_end: [0.19, 0.0]

....
```

ROS Move Base - Configuration

example move_base launch file

```
<launch>

    <node pkg="move_base" type="move_base" respawn="false" name
        <!-- Load common configuration files -->
        <rosparam file="$(find teb_local_planner_youbot)
/config/move_base_params.yaml" command="load" />
        <rosparam file="$(find teb_local_planner_youbot)
/config/costmap_common_params.yaml" command="load" ns="global_co
        <rosparam file="$(find teb_local_planner_youbot)
/config/costmap_common_params.yaml" command="load" ns="local_cos

  ...
    </node>
</launch>
```

http://wiki.ros.org/navigation/Tutorials/RobotSetup#Running_the_Navigation_Stack

Tutorial

Task

Set up the navigation stack for the robot