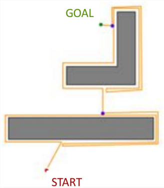
236609 - AI and Robotics - Fall 2022

Lesson 4: Beyond Reactive Control and SLAM

Sarah Keren

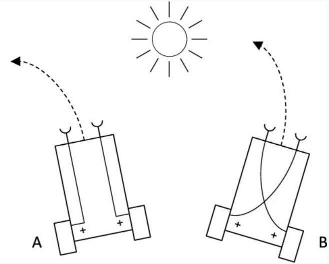
The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Recap



Proportional Control Program Loop

```
Move 0.5 body-length forward
If distance-to-wall > desired,
   let error = |desired – distance-to-wall|
   Then turn Kp * error towards the wall
   Else turn  Kp * error away from the wall
```



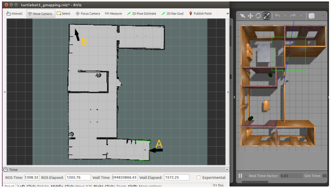
- Reactive control
- PD and PID control
- Bug-based path planning (mostly-local without a map)
    - Robots can see the goal (direction and distance)
    - But there are unknown obstacles in the way (No map)

What if you can't see the goal ? What about complex tasks ?

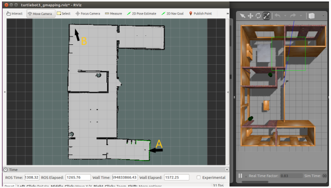
Table of contents

Our Challenge

Our Challenge



Our Challenge



What about more complex tasks?

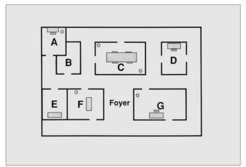
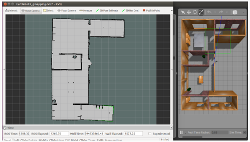


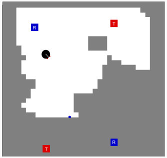
Figure 2. The Competition Arena.

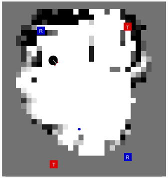


Running Case Study

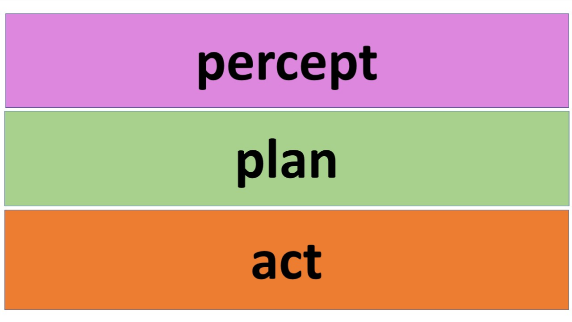
Pick Up the Trash (AAAI Competition, 1994-1995): collect red soda cans and put them in blue rubbish bins.

https://ojs.aaai.org/index.php/aimagazine/article/view/1213



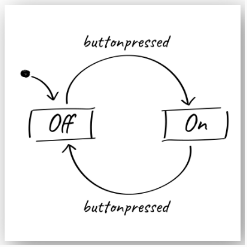
Control Architectures

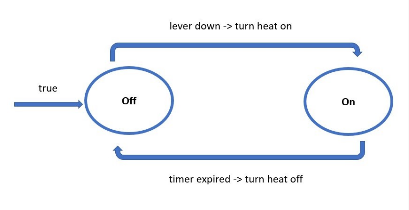


How to combine the three modules ?

Finite State Machines

- Previously: Braitenberg vehicles operate on current input values only
- Next step: add **state**: remember what state the robot is in
- **Finite State Machines/ Finite Automata**: a finite set of states and transitions between these states.





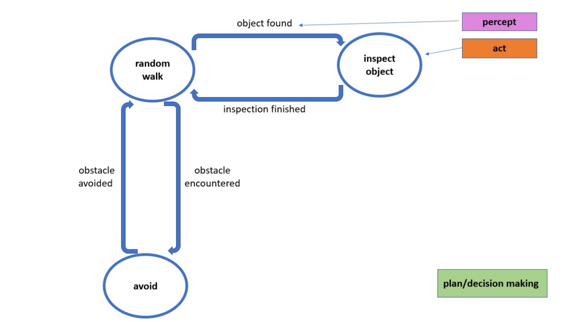
FSM for exploring robot ?

Finite State Machines

FSM for exploring robot

Finite State Machines

FSM for exploring robot



https://youtu.be/K6GNw6QUpR4?t=1964

Finite State Machines

FSM for pick up the trash ?

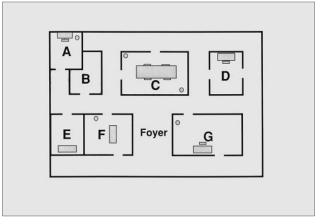
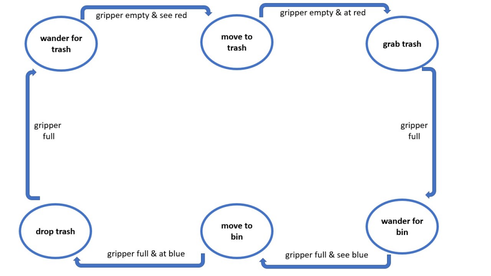


Figure 2. The Competition Arena.

Finite State Machines

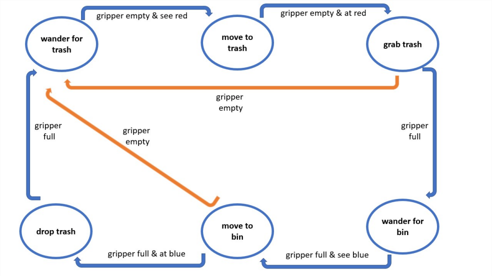
FSM for pick up the trash



What’s missing?

Finite State Machines

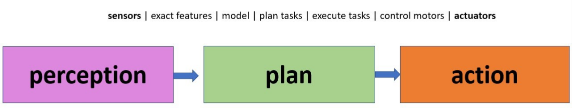
FSM for pick up the trash - take 2



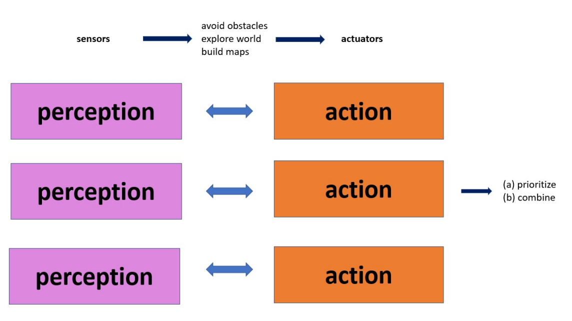
Are we done ?

Two Paradigms

Serial architecture:



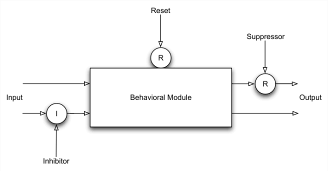
Concurrent behaviors:

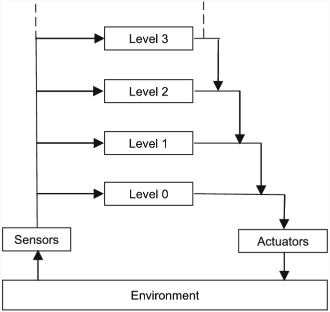


Subsumption Architecture

- 1984- Braintenberg
- 1986- Rodney Brooks (MIT): ”A robust layered control system for mobile robot”
- Radical new idea at the time - today has over 11,500 citations
- Robot has several **behavioral modules** (basic behaviors), each is represented by an augmented finite state machine.
- Response to sensor input: predominantly rule based (discrete)
- Coordination of behaviors: priority-based, via **inhibition** and **suppression**
- Hierarchical structure of behaviors: most important ones on the bottom and least important on top - cascade of rules can be triggered.

Subsumption Architecture

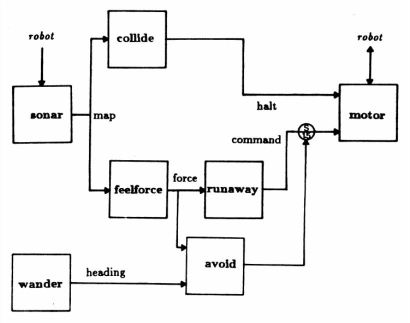




- **Inhibitor:** inhibits input signal
- **Suppressor:** replaces signal with suppressing signal
- **Hierarchical structure:**
  - Higher levels **subsume** the role of lower levels when they wish to take control
  - Each behavioral layer runs independently, concurrently and asynchronously.

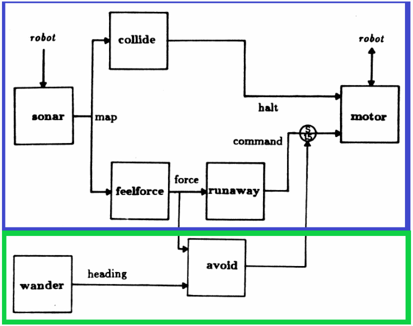
Subsumption Architecture

From Boork's paper.



Layers ?

Subsumption Architecture

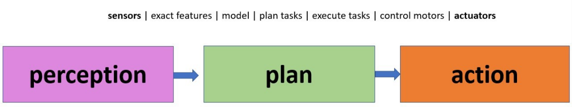


- Layer 0: makes sure robot does not come into contact with other objects
- Layer 1: wander
- Layer 0+1 : robot can wander without colliding

What about pick up trash ?
https://youtu.be/K6GNw6QUpR4?t=2946

Deliberative Sense-Plan-Act

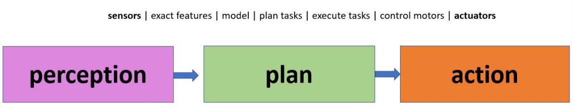
Serial architecture:



Pick up the trash:

Deliberative Sense-Plan-Act

Serial architecture:



Pick up the trash:

- **Sense:**
  - Sense and construct model of the world: e.g., positions of cans, positions of obstacles.
  - Assumptions?
- **Plan:** plan a sequence of actions - should be optimal
- **Execute:** actuate the plan

Problem Formulation: Navigation

Given:

- A **start pose** of the robot
- A desired **goal pose**
- A geometric description of the **robot** and its possible actions
- A geometric description of the **world**

Find: a path that moves the robot from start to goal

1. as quickly as possible (or any other optimization criteria)
2. without touching any obstacle (collision).

Is this model relevant for complex Task planning?

Problem Formulation: Task Planning

Given:

- A **start pose** of the robot
- ~~A desired goal pose~~ A reward function
- A geometric description of the **robot** and its possible actions.
- A geometric description of the **world**

Find: a path ~~that moves the robot from start to goal~~ that maximizes the expected accumulated reward

1. as quickly as possible (or any other optimization criteria)
2. without touching any obstacle (collision).

State Space / Graph Search

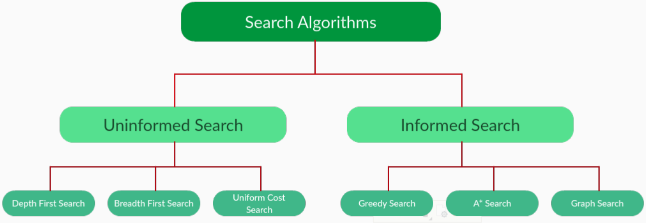
- Using graph-search algorithms to identify optimal plans.
- Heuristic estimations, extracted automatically from the problem description, are used to guide the search
- Admissible heuristics are guaranteed to underestimate the cost to goal (or over-estimate value).
- Using admissible heuristics to guide search algorithms that first explore paths with the lowest estimated cost (e.g., $A*$, is guaranteed to produce optimal solutions.

State Space / Graph Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
   end
```

State Space / Graph Search

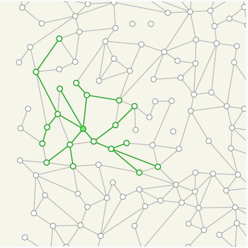
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```



Shortest-Path Graph Algorithms

The input is a **graph** (V, E), a **source node** $v_i \in V$, and a **goal node** $v_g \in V$ where:

- **nodes (vertices)** V
- **Edges** E are paths between the nodes
- Edges may be associated with uniform / non-uniform cost.

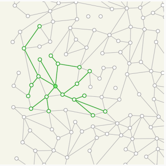


The objective is to find a (minimal-cost) path from v_i to v_g.

Shortest-Path Graph Algorithms

Methods (typically) use three functions to choose which nodes to expand next:

- $g(v)$ accumulated cost to from the source to v
- $h(v)$ estimated cost from v to the goal (heuristic)
- $f(v) = g(v) + h(v)$



- Blind (exhaustive) approaches use $g(v)$
- Greedy approaches use $h(v)$
- Informed search approaches use $f(x)$ (e.g., A*)

A*

Chooses next node to expand based on $f(n) = g(n) + h(n)$

1. $g(n)$ Distance from start
2. $h(n)$ Heuristic function that estimated the expected distance from goal

A heuristic is *admissible* if it 'optimisitic': it underestimates the cost to goal.

Key points:

- As long as the heuristic is admissible then A* an optimal solution.
- Trade-of between estimation quality and computation cost.
- h = straight-line / Manhattan distance is a good heuristic for motion planning.

Best First Search

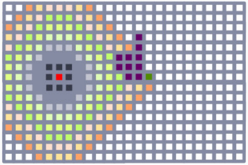
Algorithm 1 Best First Search (BFS)

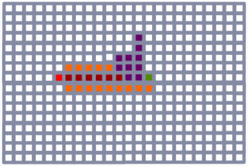
BFS(P, h)

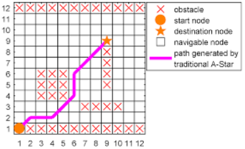
create OPEN list for unexpanded nodes
put $\langle P.root, h(P.root) \rangle$ in OPEN
$n_{cur} = ExtractMax(OPEN)$ (initial model)
while n_{cur} **do**
    if $IsTerminal(n_{cur})$ **then**
        return $ExtractPath(n_{cur})$ (best solution found - exit)
    end if
    for all $n_{suc} \in GetSuccessors(n_{cur}, P)$ **do**
        put $\langle n_{suc}, h(n_{suc}) \rangle$ in OPEN
    end for
    $n_{cur} = ExtractMax(\text{OPEN})$
end while
return no solution

How is A* a special case of Best First Search ?

A* in Robotics

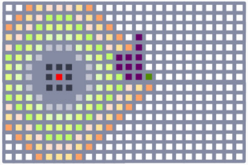


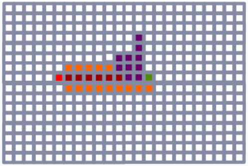


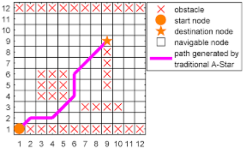


Are we done ?

A* in Robotics







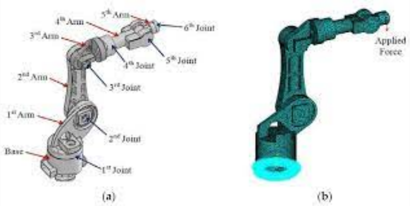
Are we done ?

- Robot capabilities (possible actions)
- Dealing with uncertainty and stochastic outcomes
- Dealing with failures
- Representing the environment
- Understanding the location of the robot

Workspace, Configuration Space and Task Space

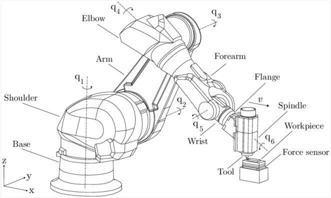
Configuration

- The **configuration** of a robot is a complete specification of the position of every point of the robot.
- A configuration q is **collision-free**, or free, if the robot placed at q does not intersect any obstacles.



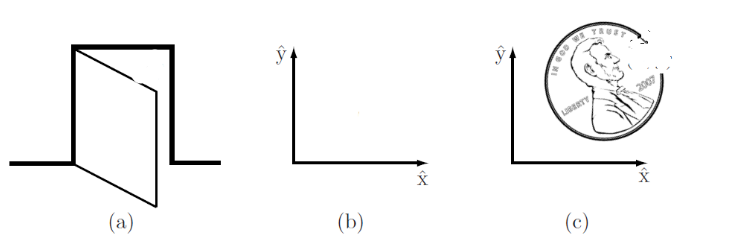
Degrees of Freedom (dof)

- The **degrees of freedom (dof)** of a robot are the minimum number n of real-valued coordinates needed to represent the configuration of a robot.
- A mechanism is typically constructed by connecting rigid bodies, called **links**, together by means of **joints**, so that relative motion between adjacent links becomes possible.
- **Actuation** of the joints, typically by electric motors, then causes the robot to move and exert forces in desired ways.
- dof typically refers to the number of movable joints of a robot.

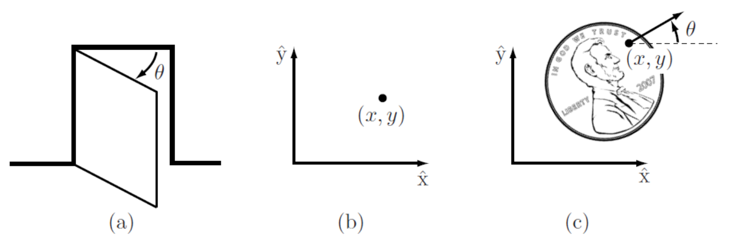


Degrees of Freedom (dof)

What's the dof ?



Degrees of Freedom (dof)



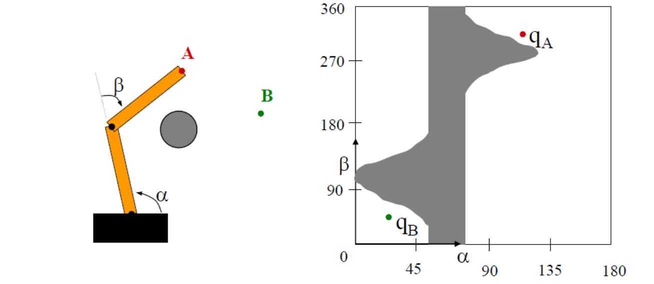
The configuration of a door is described by the angle θ (b) The configuration of a point in a plane is described by coordinates (x, y). (c) The configuration of a coin on a table is described by (x, y, θ), where θ defines the direction in which Abraham Lincoln is looking.

Configuration space, work space and task space

We are going to consider three space representations:

- The **workspace** is a specification of the robot's possible configurations in the environment.
- The **configuration space (C-space)** is a specification of the robot's attainable positions in the environment. It is the *n*-dimensional space containing all possible configurations of the robot.
- The **task space** is the space in which the robot's task can be naturally expressed.

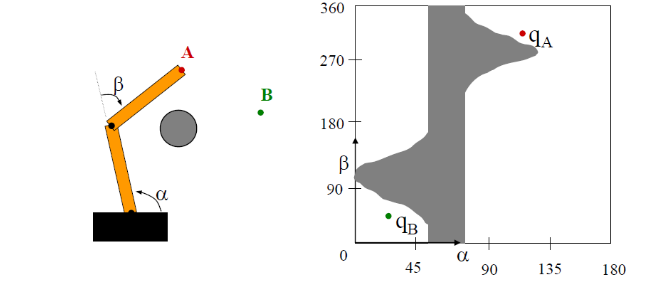
Configuration space, work space and task space: example



- This robot arm has two joints that move independently.
- Moving these joints alters the coordinates of the elbow and the gripper.

What's the workspace, configuration space and task space here ?

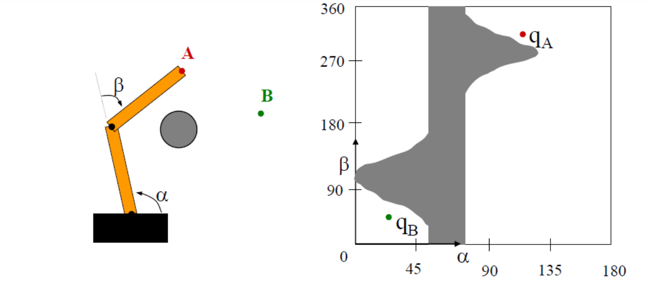
Configuration space, work space and task space



The **workspace** is a specification of the configurations that the end-effector of the robot can reach.

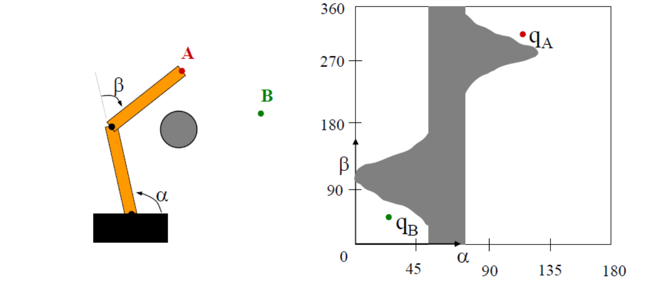
Here, the **workspace** is defined by the (x,y) coordinates of the robot that are specified in the same coordinate system of the world it's manipulating (left image).

Configuration space, work space and task space



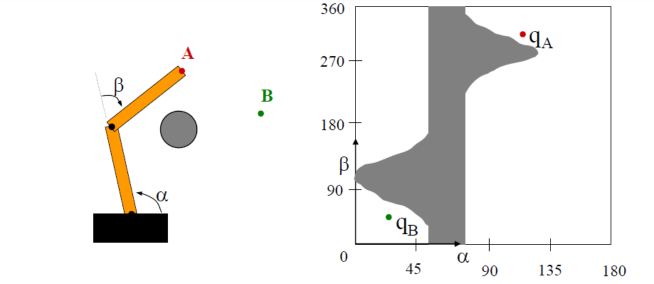
- Although the motion planning problem is defined in the actual world, it lives in another space: the **configuration space**.
- The configuration space (C-space) $\mathcal{C}$ is the *n*-dimensional space containing all possible configurations of the robot.
- The configuration of a robot is represented by a point in its C-space.
- Here, the C-space is defined by α and β and is represented in the right figure.

Configuration space, work space and task space



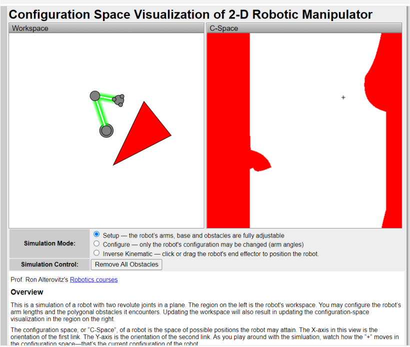
- The definition of the workspace is primarily driven by the robot's structure, independently of the task.
- The **task space** is a space in which the robot's task can be naturally expressed.
- For example, if the robot's task is to plot with a pen on a piece of paper, the task space would be R^2.
- The decision of how to define the task space is driven by the task, independently of the robot.

Configuration space, work space and task space



- Both the task space and the workspace involve a choice by the user.
- For example: the user may decide that some freedoms of the end-effector (e.g., its orientation) do not need to be represented.
- A point in the task space or the workspace may be achievable by more than one robot configuration.
- How do we map the workspace coordinates into configuration space? This is a problem of **inverse kinematics**.

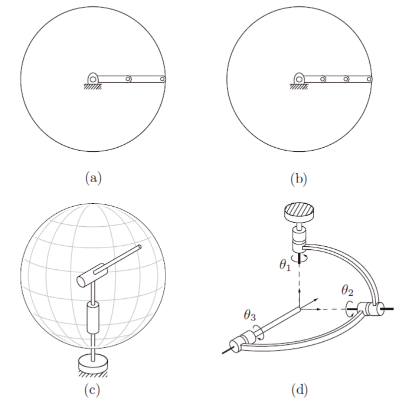
Configuration space, work space and task space



https://www.cs.unc.edu/~jeffi/c-space/robot.xhtml

https://www.youtube.com/watch?v=SBFwgR4K1Gk

Configuration space, work space and task space



- Two mechanisms with different C-spaces may have the same workspace (e.g. a & b)
- Two mechanisms with the same C-space may also have different workspaces (e.g. a & c).

Mapping

Problem Formulation: Navigation

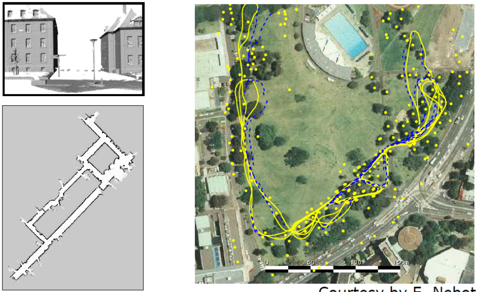
We will focus on robots, but it's a general problem (think Google maps)

Two components given as input:

- Map representation (graph):
  - Feature based maps (office numbers, landmarks)
  - Grid based maps (cartesian, quadtrees)
  - Polygonal maps (geometric decompositions)
- Path Finding Algorithms: find shortest path in graph

Mapping

- What is Robot Mapping?
  - a robot (a device) moves through the environment
  - Mapping – modeling the environment
- e.g., estimate landmark positions given the robot's poses.



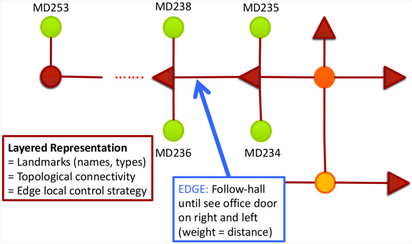
Courtesy by E. Nebot

Map Representation: Feature based

- Also known as a topological or landmark-based map
- Features your robot can recognize: includes both natural landmarks (corner, doorway, hallway) and artificial ones (office door numbers; or robot-friendly tags).
- World is a graph that connects landmarks
  - Edges represent actual motion: how to get from landmark A to landmark B
  - Edges can also keep extra attributes: distance, time it takes, etc.

Google Maps are topological maps for humans (e.g. turn at intersection)

Caveat: Much harder to construct topological maps for robots!

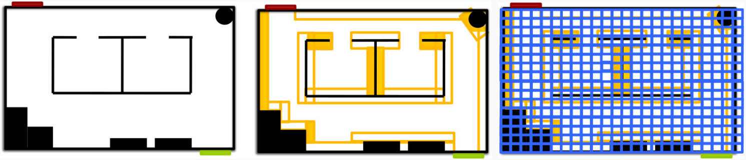


Map Representation: grid based

Ignore any notion of Features

Instead, Convert the map into a grid-graph:

- Step 1: Grow the boundaries (by robot size)
- Step 2: Overlay a grid and create an occupancy matrix.



How to choose the right resolution of the grid?

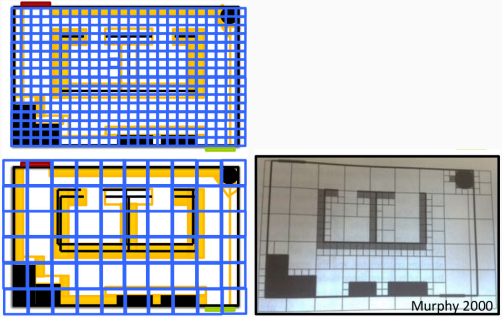
- Too small – computationally expensive, jagged paths
- Too big – might miss paths

Ideas?

Map Representation: grid based

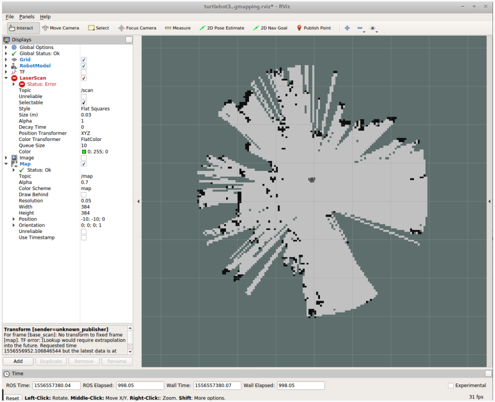
Quadtree

- Create a grid recursively!
  - Start with very coarse grid;
  - Then for each grid section, if there is an obstacles, refine.



Adapted from Murphy 2000

Occupancy Grid



- Maps the environment as an array of cells.
- Each cell holds a probability value – that the cell is occupied.
- Useful for combining different sensor scans, and even different sensor modalities. – sonar, laser, IR, bump, etc.

How do we use these maps to find paths for robots to follow?

Can’t we just apply A* ?

Summary

Summary:

- We looked at the structure of a robot
- We examined some approaches for reactive control for motion planning
- We examined the basics of SLAM

What next ?

- Take a closer look at navigation and path planning techniques