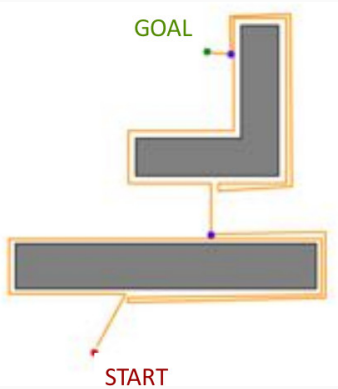
236609 - AI and Robotics - Fall 2022

Lesson 4: Beyond Reactive Control and SLAM

Sarah Keren

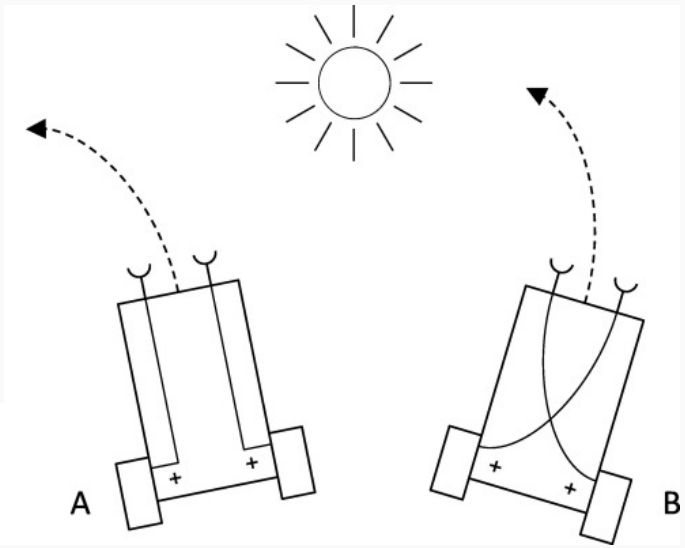
The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

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
```



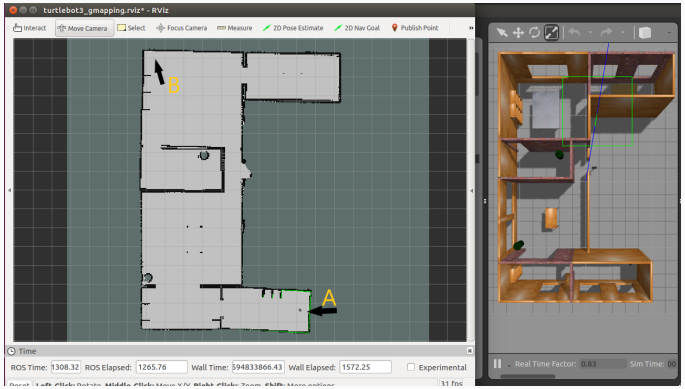
- Reactive control
- PD and PID control
- Bug-based path planning (mostly-local without a map)
  - Robots can see the goal (direction and distance)
  - But there are unknown obstacles in the way (No map)

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

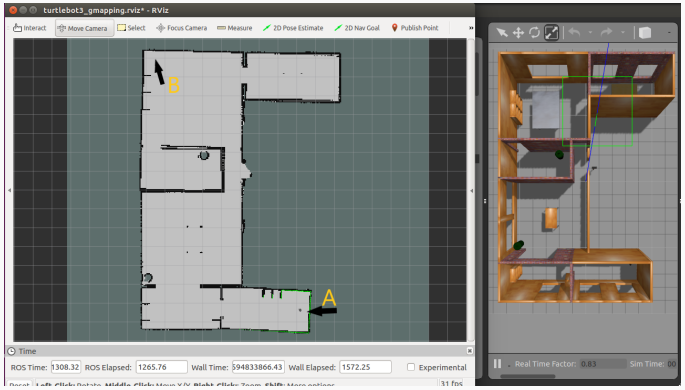
Table of contents

Our Challenge

Our Challenge



Our Challenge



What about more complex tasks?

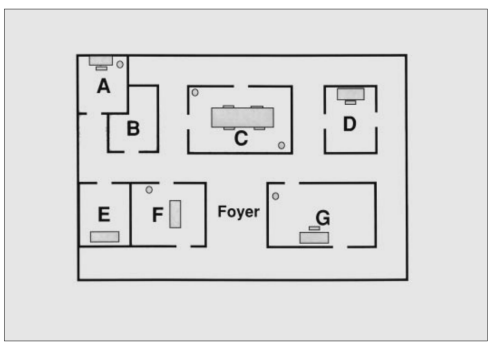
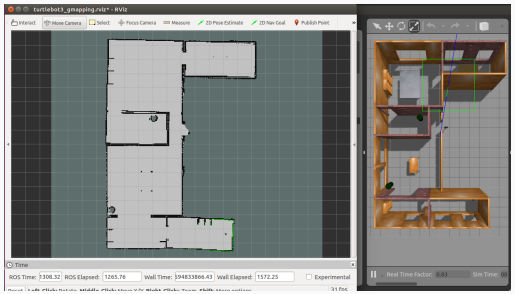


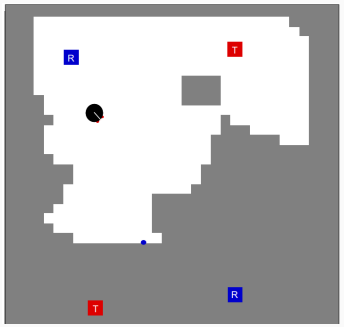
Figure 2. The Competition Arena.

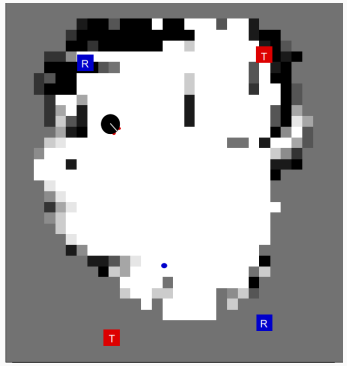


Running Case Study

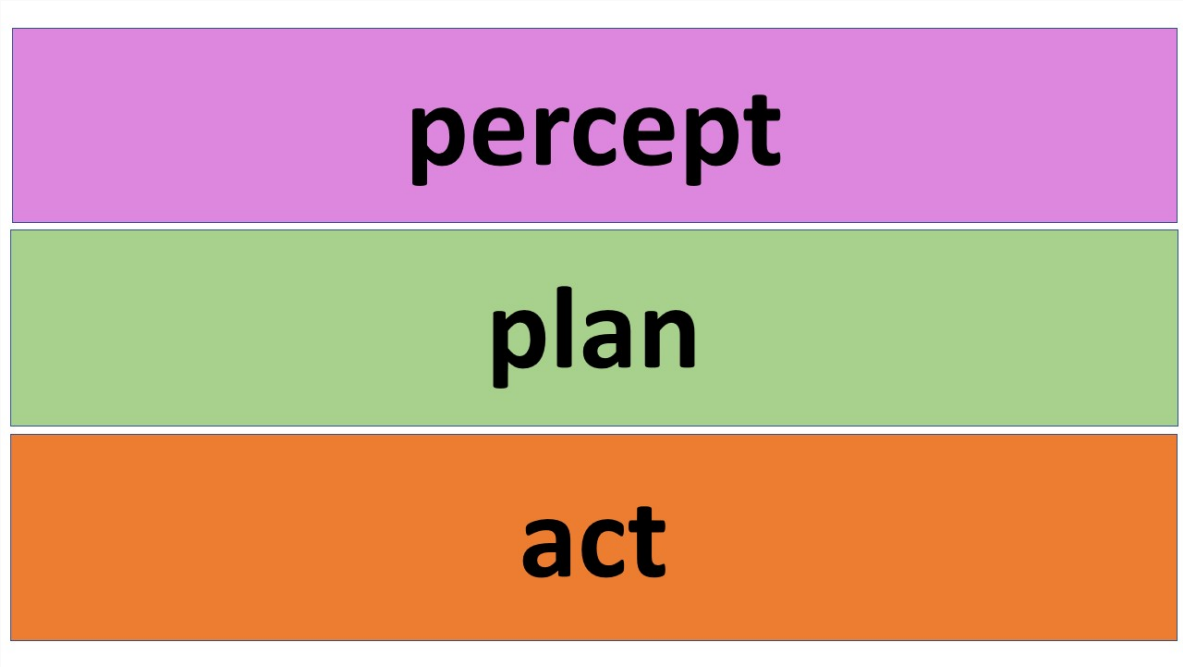
Pick Up the Trash (AAAI Competition, 1994-1995): collect red soda cans and put them in blue rubbish bins.

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





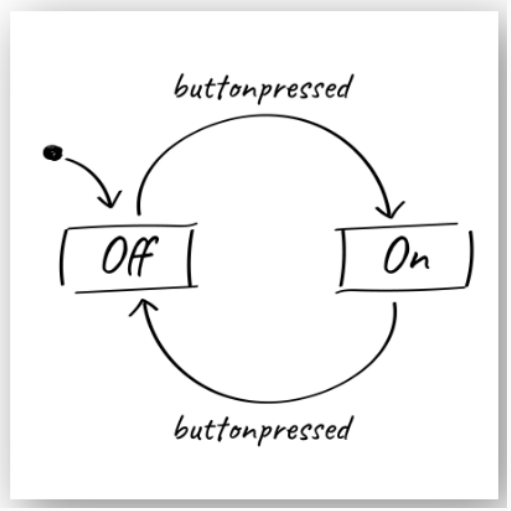
Control Architectures

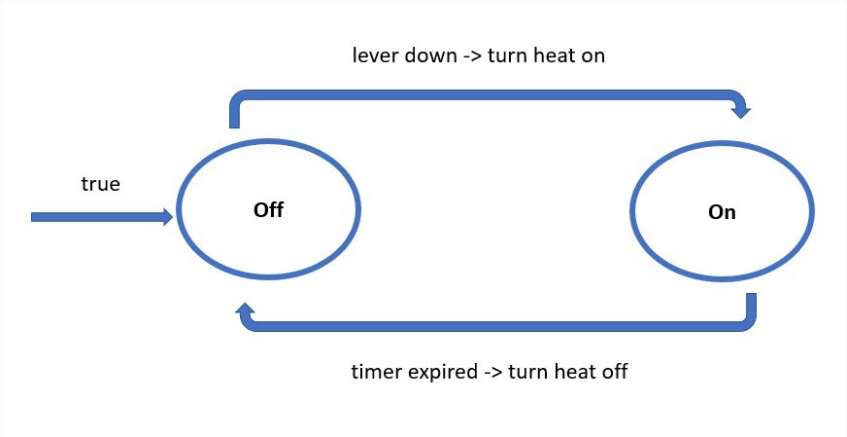


How to combine the three modules ?

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 Machine for Robots

A **finite state machine (FSM)** for robots is defined as a 6-tuple:

$$M = (Q, \Sigma, \delta, q_0, F, \Lambda)$$

where:

- Q: A finite set of states representing the robot's modes or behaviors (e.g., *Idle*, *Navigate*, *Pickup*, *Dropoff*).
- Σ: A finite set of input symbols representing sensory inputs or external events (e.g., *ObstacleDetected*, *TargetReached*).
- $\delta : Q \times \Sigma \to Q$: The state transition function, mapping a current state and input symbol to a new state.
- $q_0 \in Q$: The initial state, where the robot starts (e.g., *Idle*).
- $F \subseteq Q$: A set of final states representing termination conditions or completed tasks.
- $\Lambda : Q \to \mathcal{A}$: An output function mapping each state to a set of actions $\mathcal{A}$ the robot performs in that state (e.g., *MoveForward*, *PickUpObject*).

Finite State Machine for Robots

The robot operates as follows:

- Begin in the initial state q_0.
- Continuously monitor inputs Σ from sensors or the environment.
- Transition between states using the transition function δ, based on the current state and observed inputs.
- Execute actions $\Lambda(q)$ associated with the current state q.
- The robot stops or completes its task upon reaching a state $q \in F$.

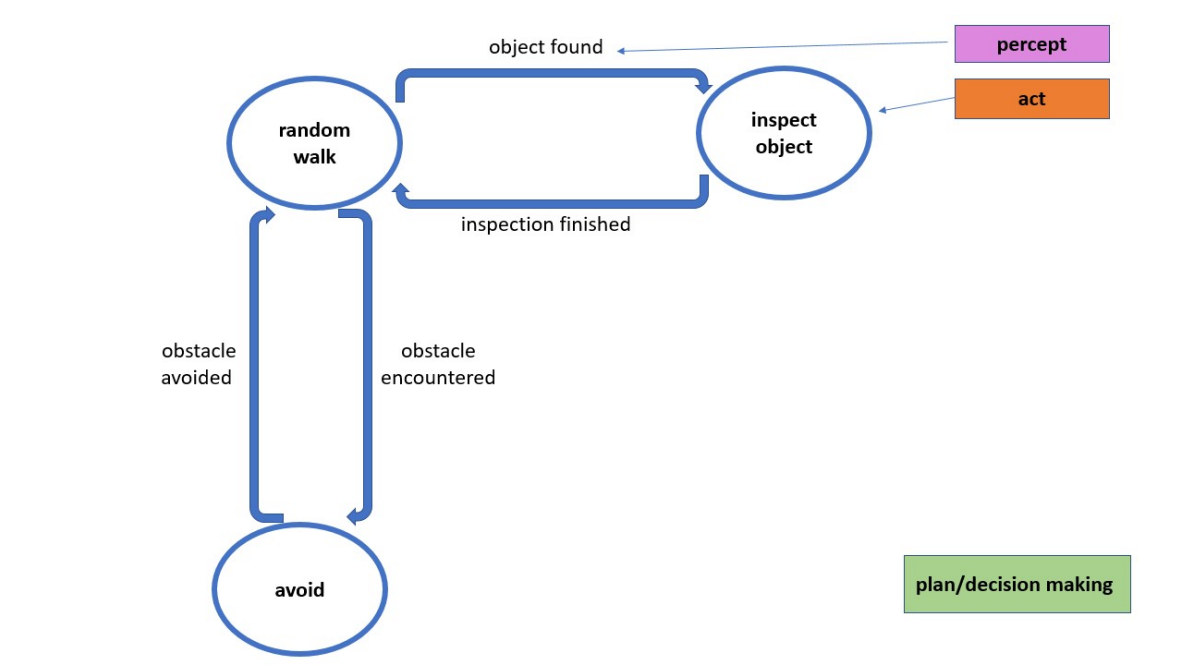
This FSM is particularly useful for modeling robotic behaviors in autonomous systems, where decisions are driven by sensor inputs and task requirements.

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 ?

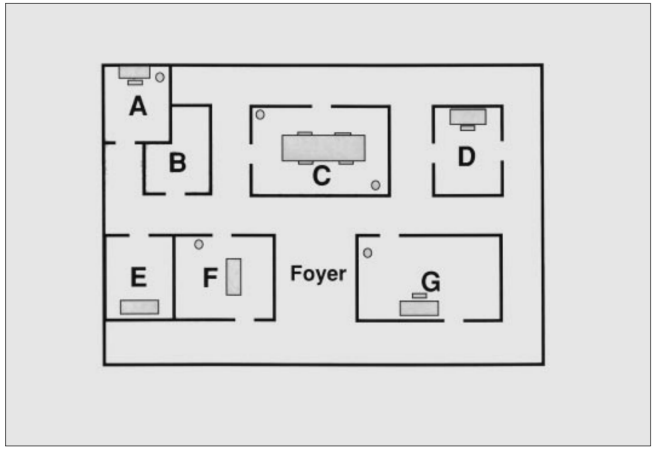
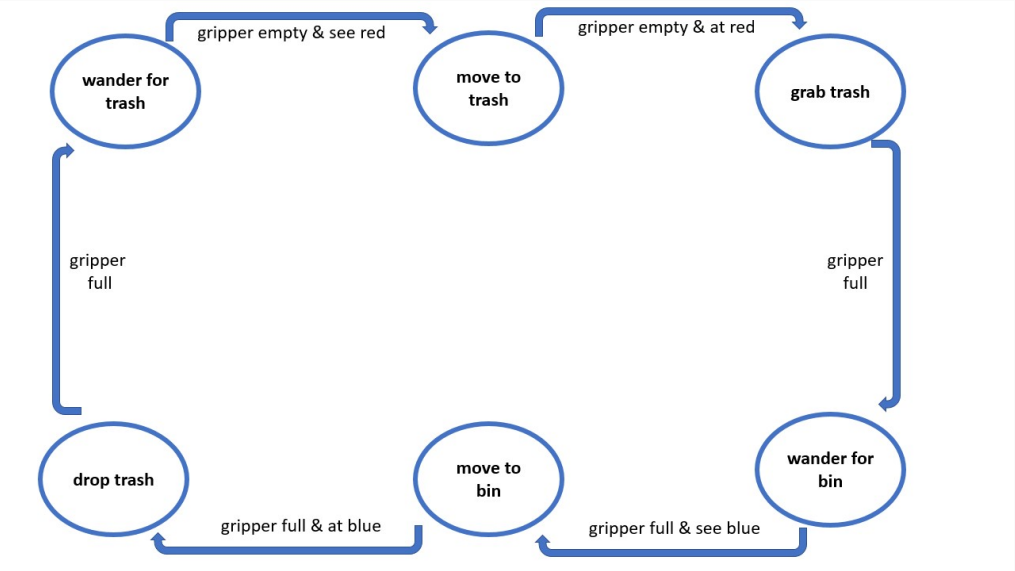


Figure 2. The Competition Arena.

Finite State Machines

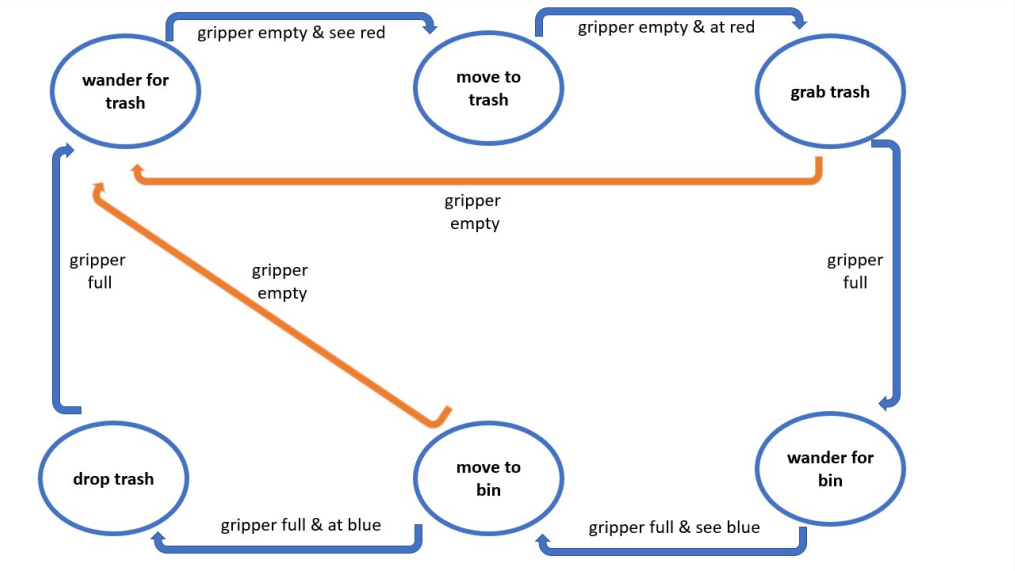
FSM for pick up the trash



What's missing?

Finite State Machines

FSM for pick up the trash - take 2



Are we done ?

Reminder: Propositional STRIPS

Propositional STRIPS planning task defined by a tuple $P = \langle \mathcal{F}, I, \mathcal{A}, G, \mathcal{C} \rangle$, where

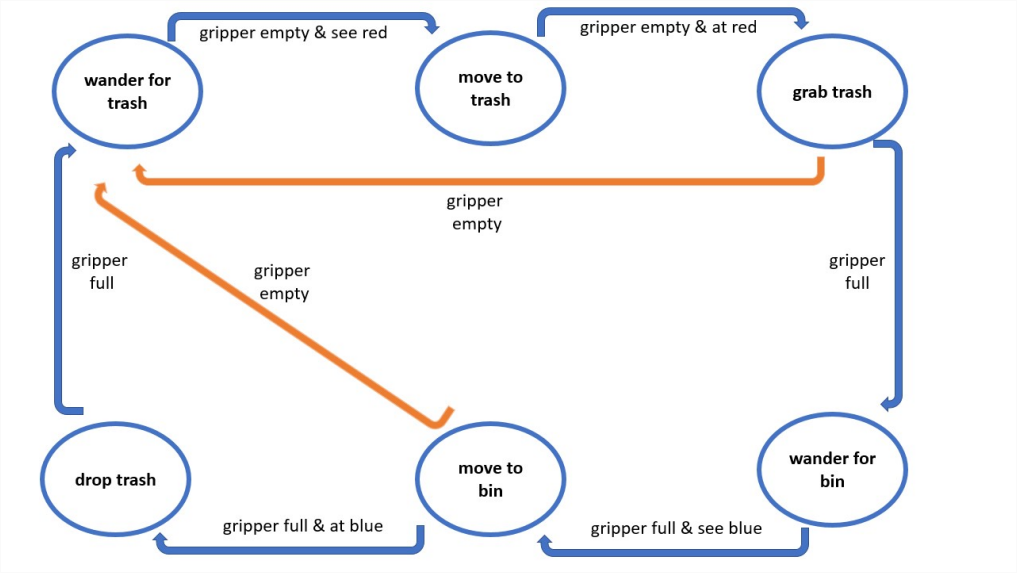
- $\mathcal{F}$ is a set of fluents and a state s is represented by the fluents that are true in s
- $I \subseteq F$ is the initial state
- $G \subseteq F$ represents the set of goal states, and
- $\mathcal{A}$ is a set of actions.
  - Each action is a triple $a = \langle pre(a), add(a), del(a) \rangle$, which represents the precondition, add, and delete lists respectively, all subsets of $\mathcal{F}$.
  - An action a is applicable in state s if $pre(a) \subseteq s$.
  - If action a is applied in state s, it results in a new state $s' = (s \setminus del(a)) \cup add(a)$.
- $\mathcal{C} : \mathcal{A} \to \mathbb{R}_0^+$ is a cost function that assigns each action a non-negative cost.

Remidner: Propositional STRIPS

- The objective is to find a plan $\pi = \langle a_1, \ldots, a_n \rangle$: a sequence of actions that brings an agent from the initial state to a goal state.
- The cost $c(\pi)$ of a plan π is $\Sigma_{i=1}^{n}(C(a_i))$.
- Often, the objective is to find an optimal solution for P: a plan π^* that minimizes the associated cost.
- The literature is rich with different approaches developed to solve the planning problem (Bonet and Geffner 2013): more on this later on in the course.

From *https://ai.dmi.unibas.ch/misc/tutorial_aaai2015/planning02.pdf*

Finite State Machines vs. STRIPS



ROSPlan

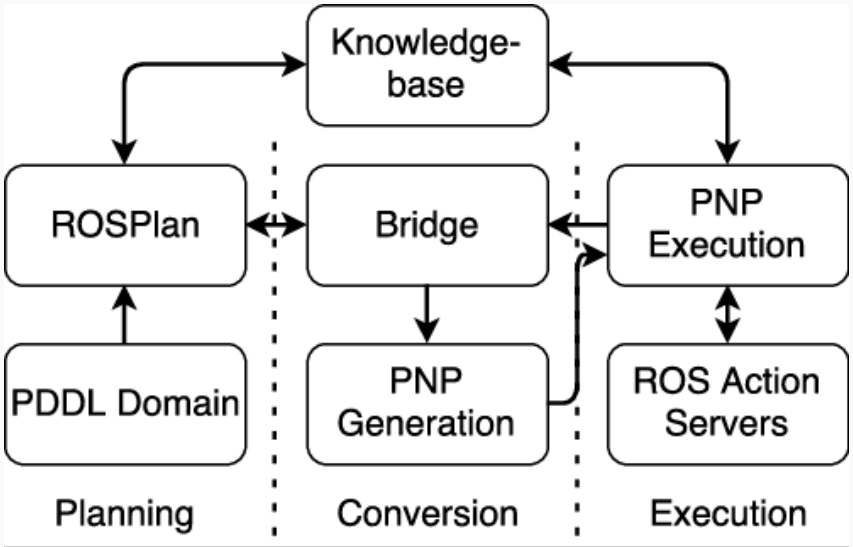


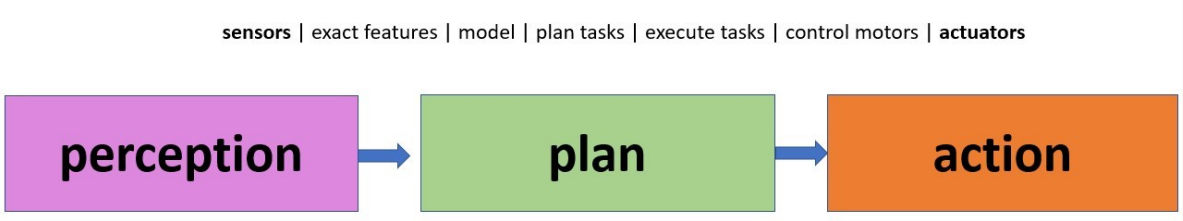
Figure 1: State machines and PDDL

https://kcl-planning.github.io/ROSPlan/

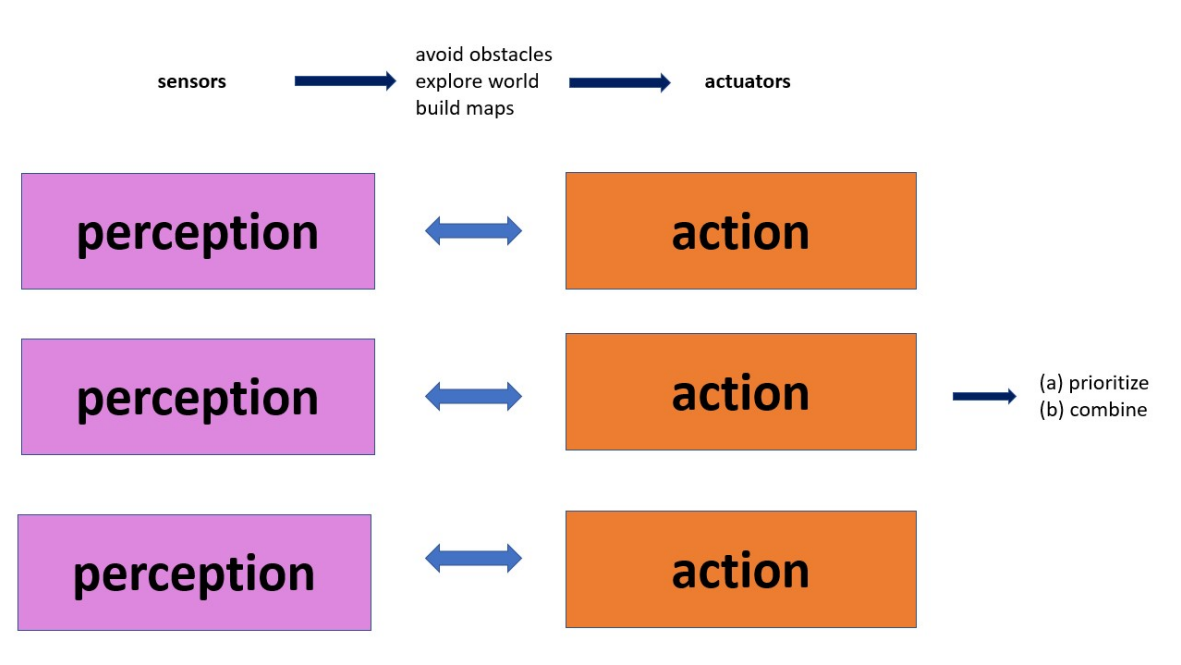
Architecture

Two Paradigms

Serial architecture:



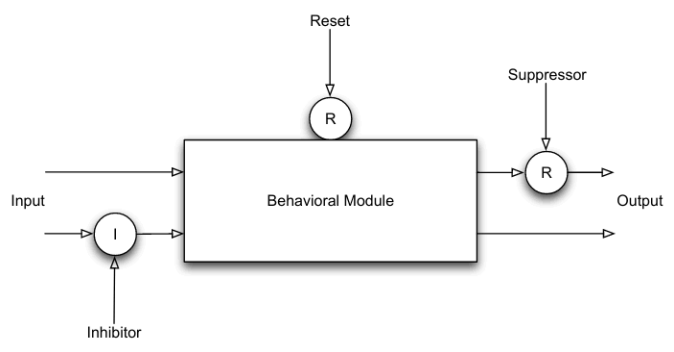
Concurrent behaviors:

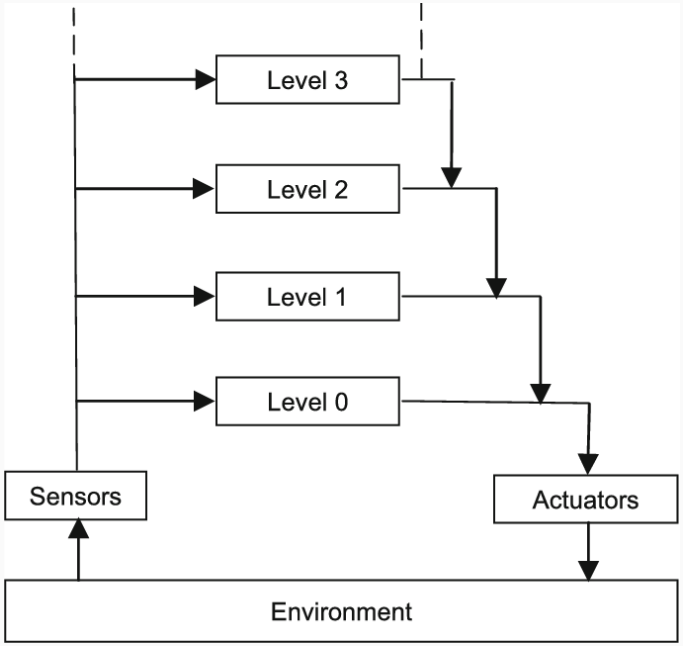


Subsumption Architecture

- 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

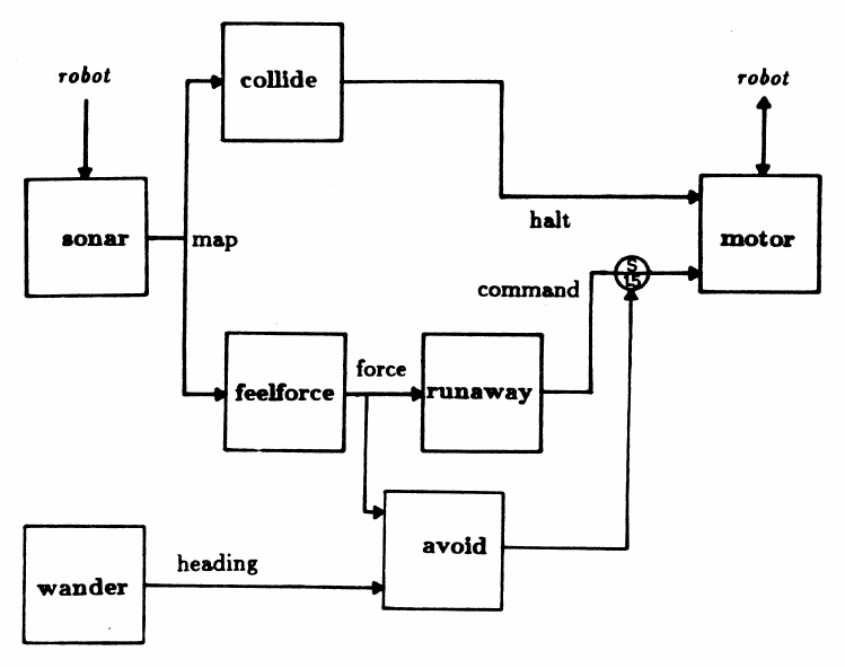




- **Inhibitor:** inhibits input signal
- **Suppressor:** replaces signal with suppressing signal
- **Hierarchical structure:**
  - Higher levels **subsume** the role of lower levels when they wish to take control
  - Each behavioral layer runs independently, concurrently and asynchronously.

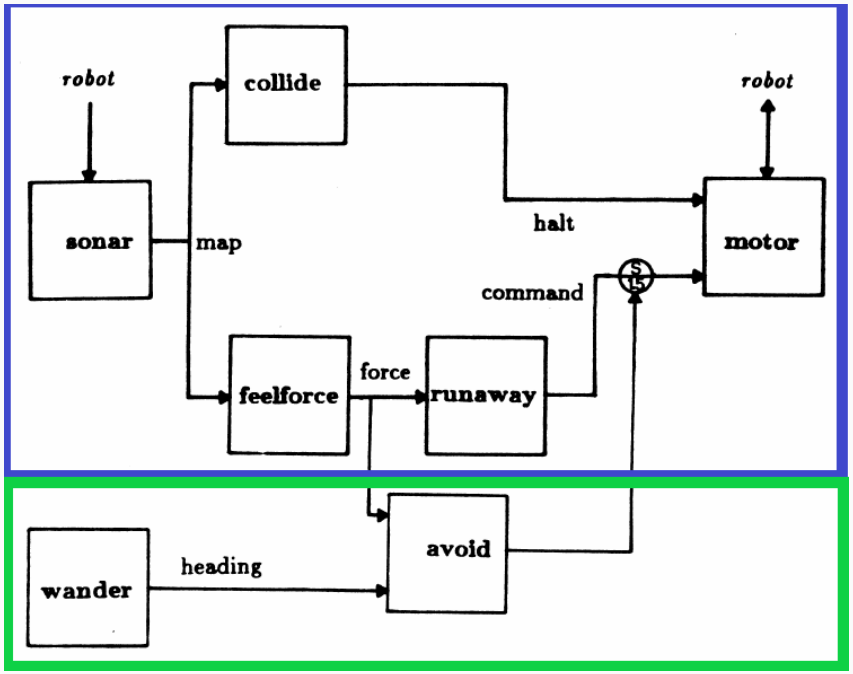
Subsumption Architecture

From Boork's paper.



Layers ?

Subsumption Architecture



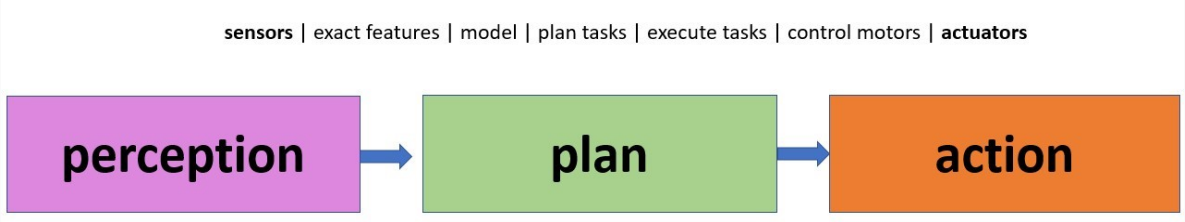
- Layer 0: makes sure robot does not come into contact with other objects
- Layer 1: wander
- Layer 0+1 : robot can wander without colliding

What about pick up trash ?

https://youtu.be/K6GNw6QUpR4?t=2946

Deliberative Sense-Plan-Act

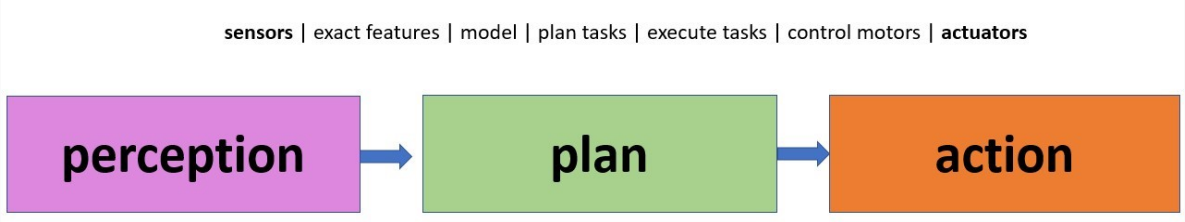
Serial architecture:



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

Planning in Robotics

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

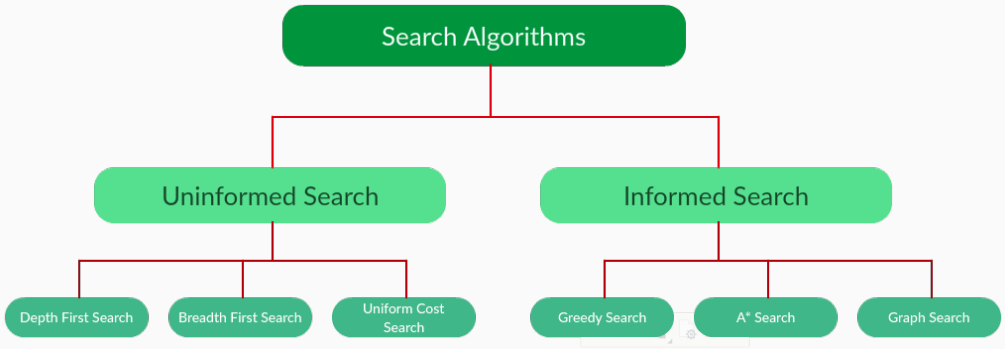
- Using graph-search algorithms to identify optimal plans.
- Heuristic estimations, extracted automatically from the problem description, are used to guide the search
- Admissible heuristics are guaranteed to underestimate the cost to goal (or over-estimate value).
- Using admissible heuristics to guide 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

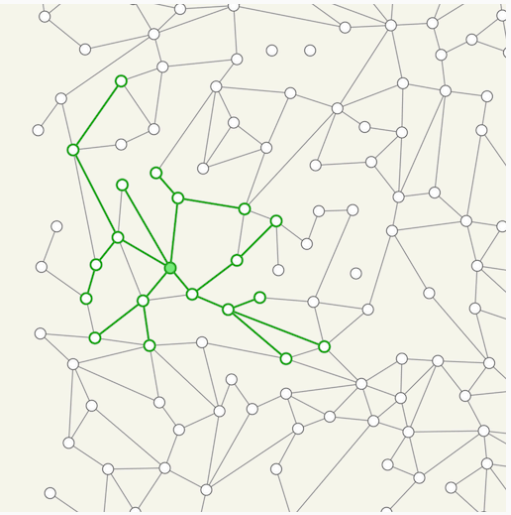
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the 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.

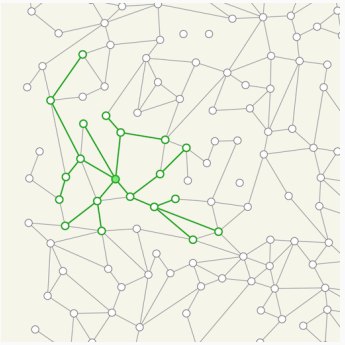


The objective is to find a (minimal-cost) path from v_i to v_g.

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

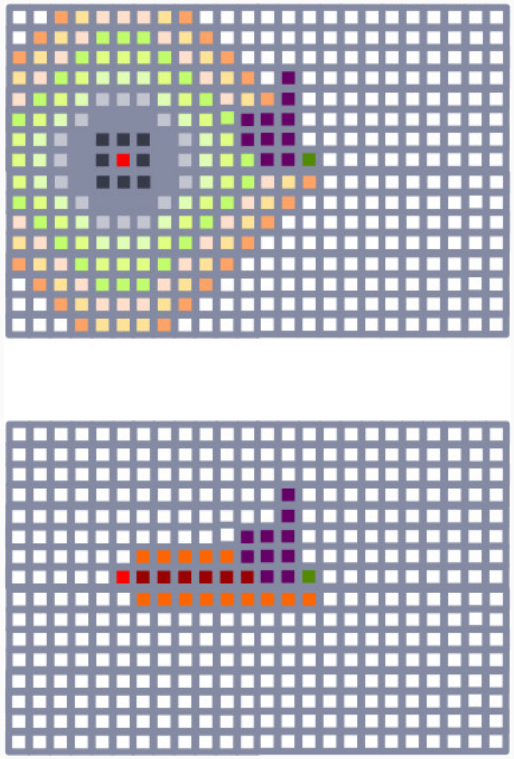
Algorithm 1 Best First Search (BFS)

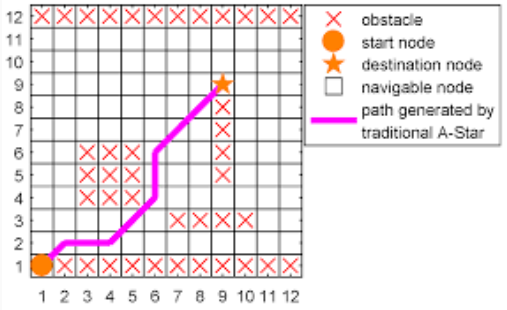
BFS(P, h)

```
create OPEN list for unexpanded nodes
put ⟨P.root, h(P.root)⟩ 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 ∈ GetSuccessors(n_cur, P) do
        put ⟨n_suc, g(n_suc) + h(n_suc)⟩ in OPEN
    end for
    n_cur = ExtractMax(OPEN)
end while
return no solution
```

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

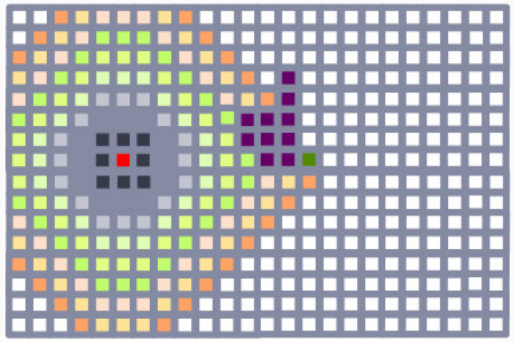
A* in Robotics

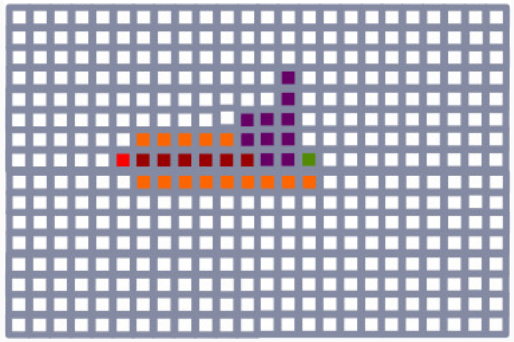


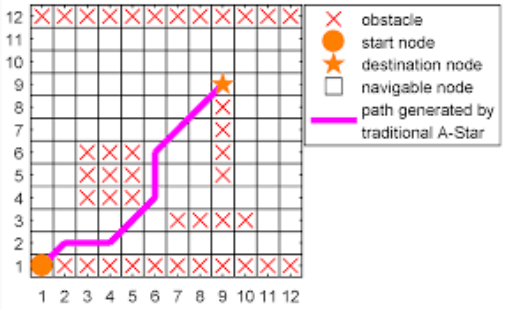


Are we done ?

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

Mapping

Problem Formulation: Navigation

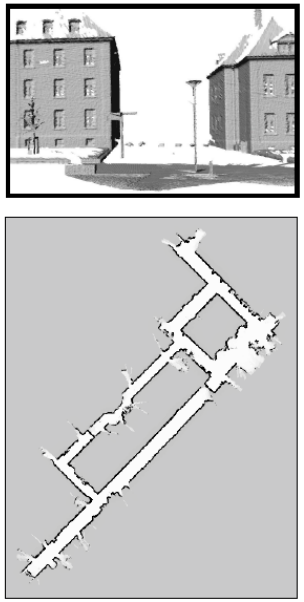
We will focus on robots, but it's a general problem (think Google maps)

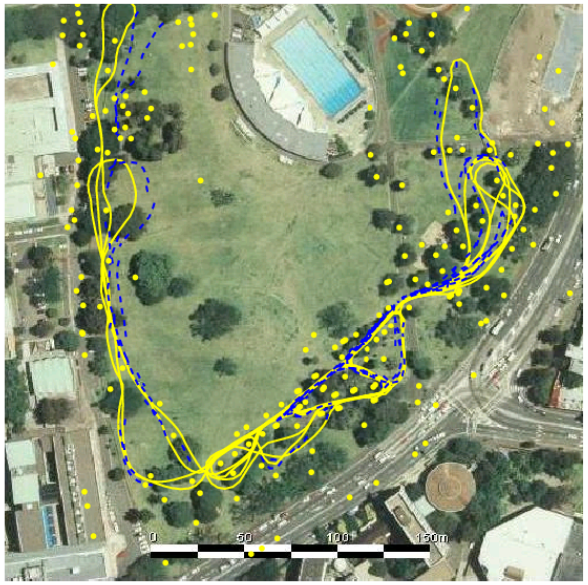
Two components given as input:

- **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.





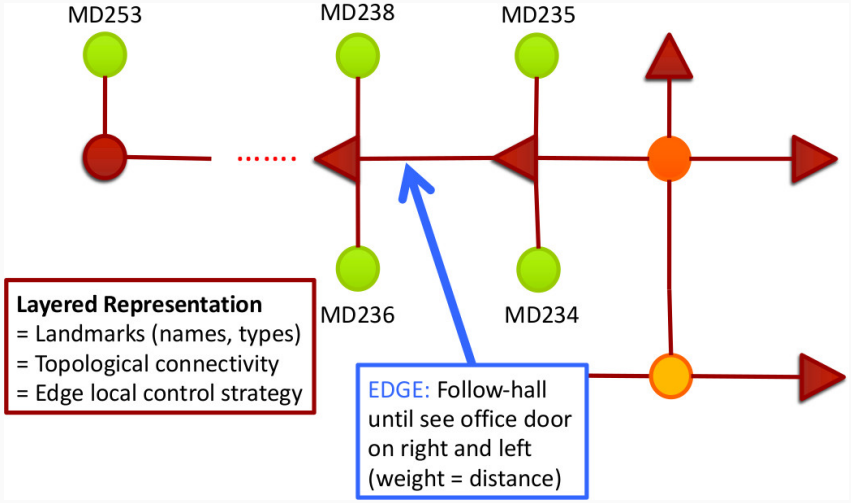
Courtesy by E. Nebot

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!

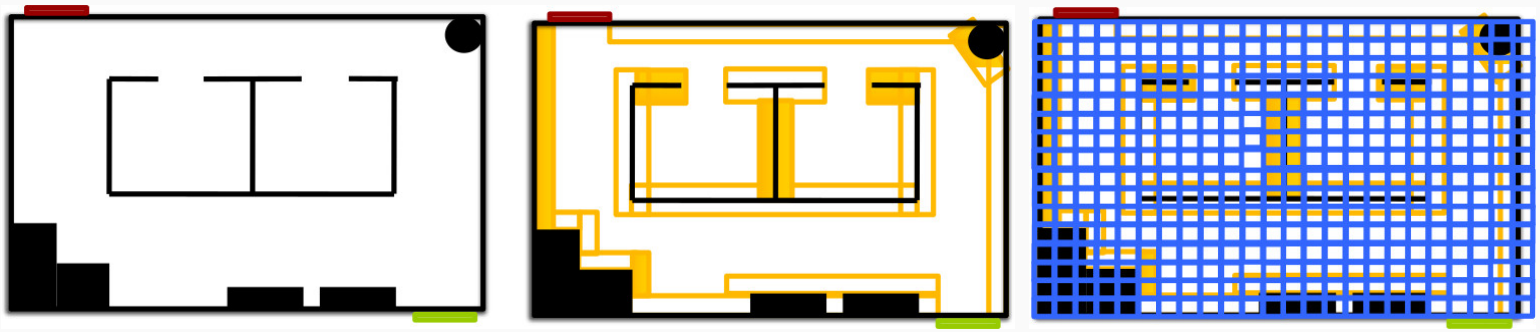


Map Representation: grid based

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?

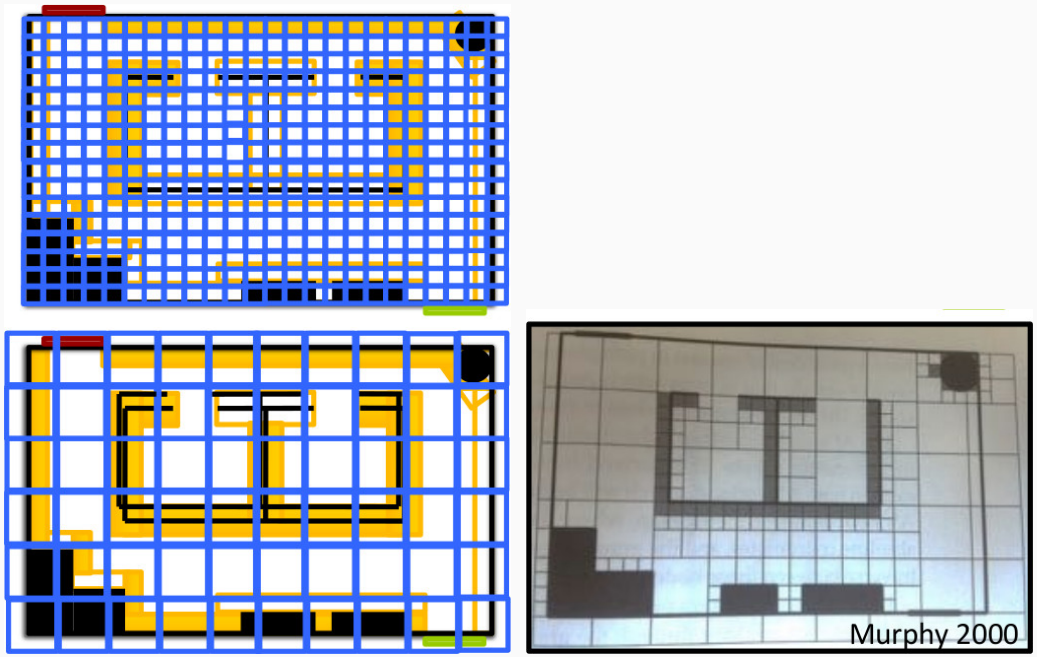
- Too small – computationally expensive, jagged paths
- Too big – might miss paths

Ideas?

Map Representation: grid based

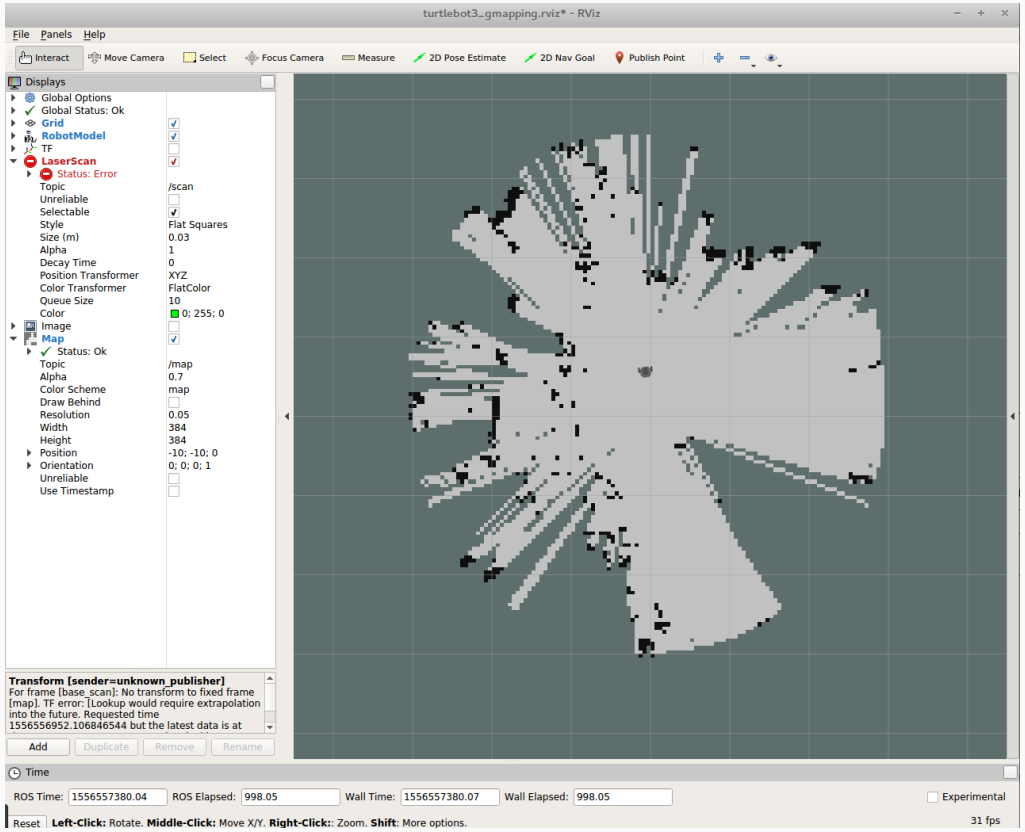
Quadtree

- 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