CSCE274 Robotic Applications and Design
Fall 2021
Control Architectures Overview

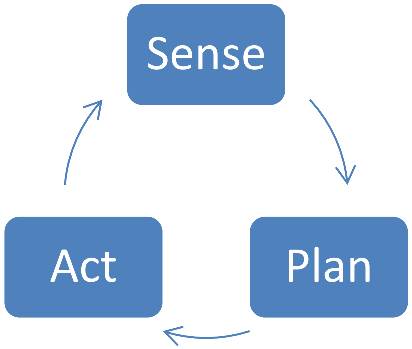
Ioannis REKLEITIS, Ibrahim SALMAN
Computer Science and Engineering
University of South Carolina
yiannisr@cse.sc.edu

Control architecture

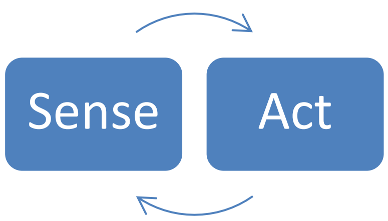
- A robot control architecture (or paradigm) is the set of principles, building blocks, and tools for designing robots
- It provides guiding principles and constraints for organizing robot's control system

Control architectures

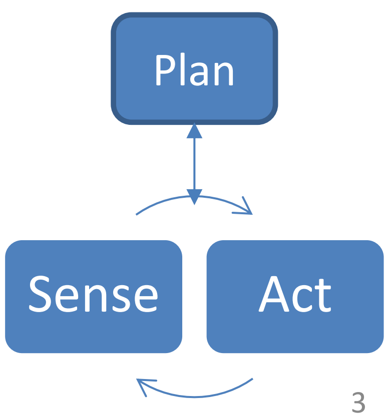
- Deliberative control
  - Top-down approach: sense-plan-act
  - Starts with high level goals that are decomposed in subtasks



- Reactive control
  - Bottom-up approach
  - Independent modules run concurrently monitoring sensor data and triggering actions accordingly



- Hybrid control
  - Deliberative at high level, reactive at low level



Control architectures

- Behavior-based control is usually considered in literature a type of reactive control architecture
  - Different behaviors to achieve a goal

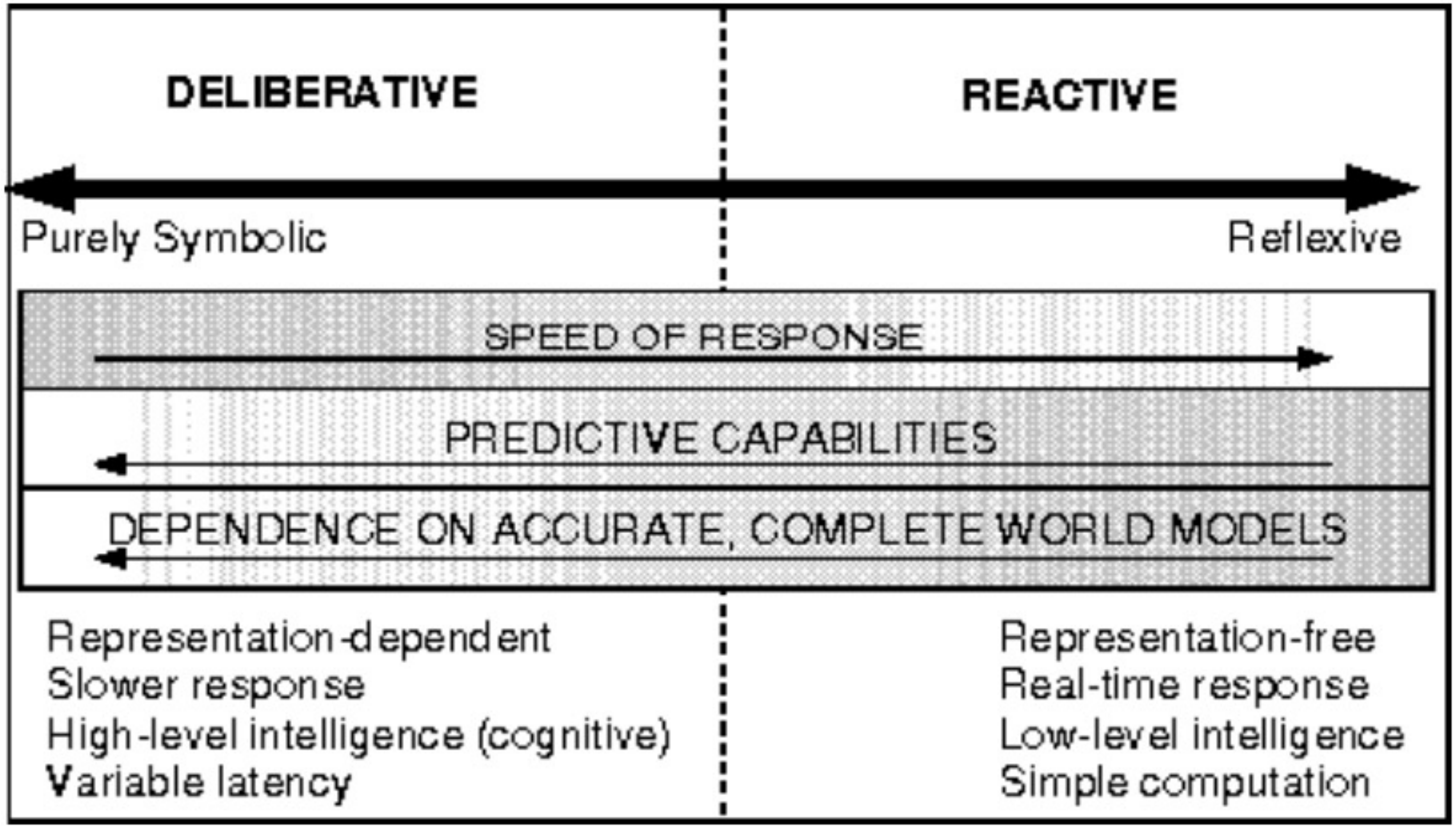
Dimensions

- Each architecture differs in how they consider different dimensions
  - Time-scale: long time-scale vs. real-time
  - Modularity: sequential vs. parallel
  - Representation of the world
    - Consider past or discard information
    - Discrete vs. continuous

Levels of control problem

- According to the different dimensions, each architecture solves control problems at different levels
  - High level: discrete problem, long time scale
    - E.g., pick bottle of water from the fridge
  - Intermediate level: continuous or discrete problem, time scale of few seconds
    - E.g., navigate to the fridge
  - Low level: continuous-valued problems, short time scale
    - E.g., where the robot should place the leg at the next step

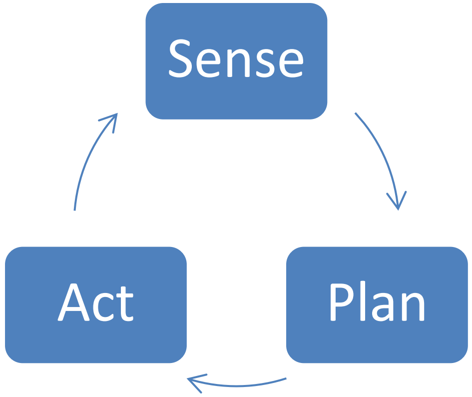
Spectrum of control



Source: [Arkin, 1998, MIT Press]

Deliberative architecture

- The robot in a deliberative control architecture (also called Sense-Plan-Act architecture)
  1. Plans a solution for the task by reasoning about the sensed world and the outcome of its actions
  2. Executes it



Planning

- "Planning is the process of looking ahead at the outcomes of possible actions, and searching for the sequence of actions that will reach the desired goal"

  Mataric, *"The Robotics Primer"*

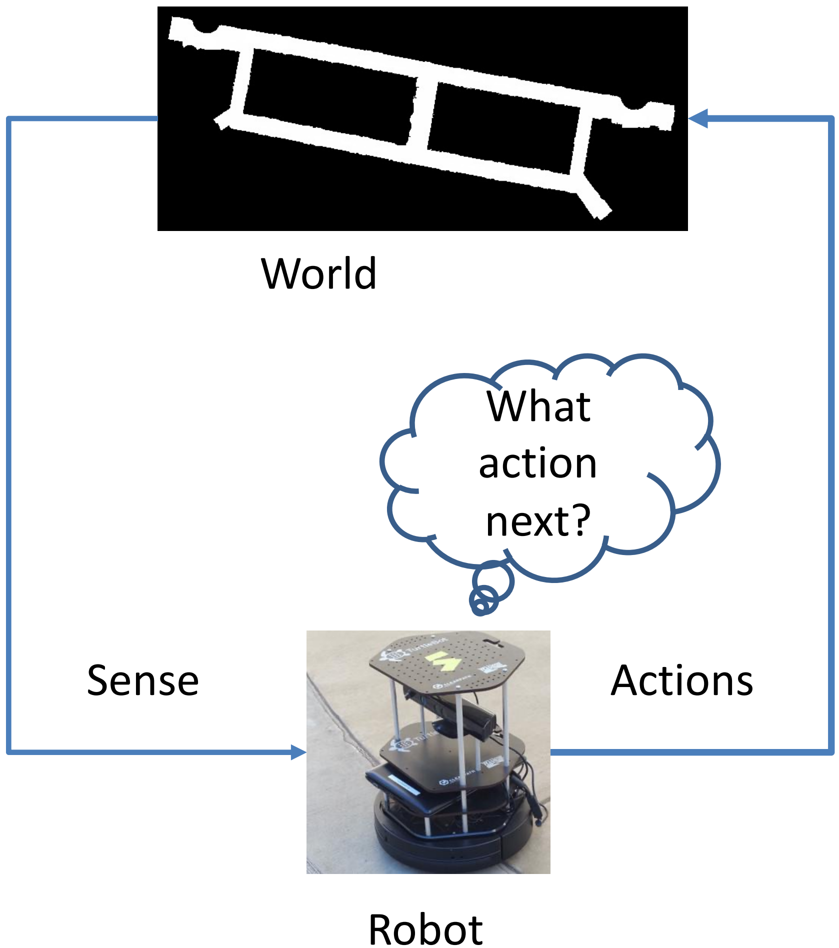
- "Planning can be interpreted as a kind of problem solving, where an agent uses its beliefs about available actions and their consequences, in order to identify a solution over an abstract set of possible plans"

  Russel and Norvig, *"Artificial Intelligence, a modern approach"*

Planning view

Static vs. Dynamic

Predictable vs. Unpredictable



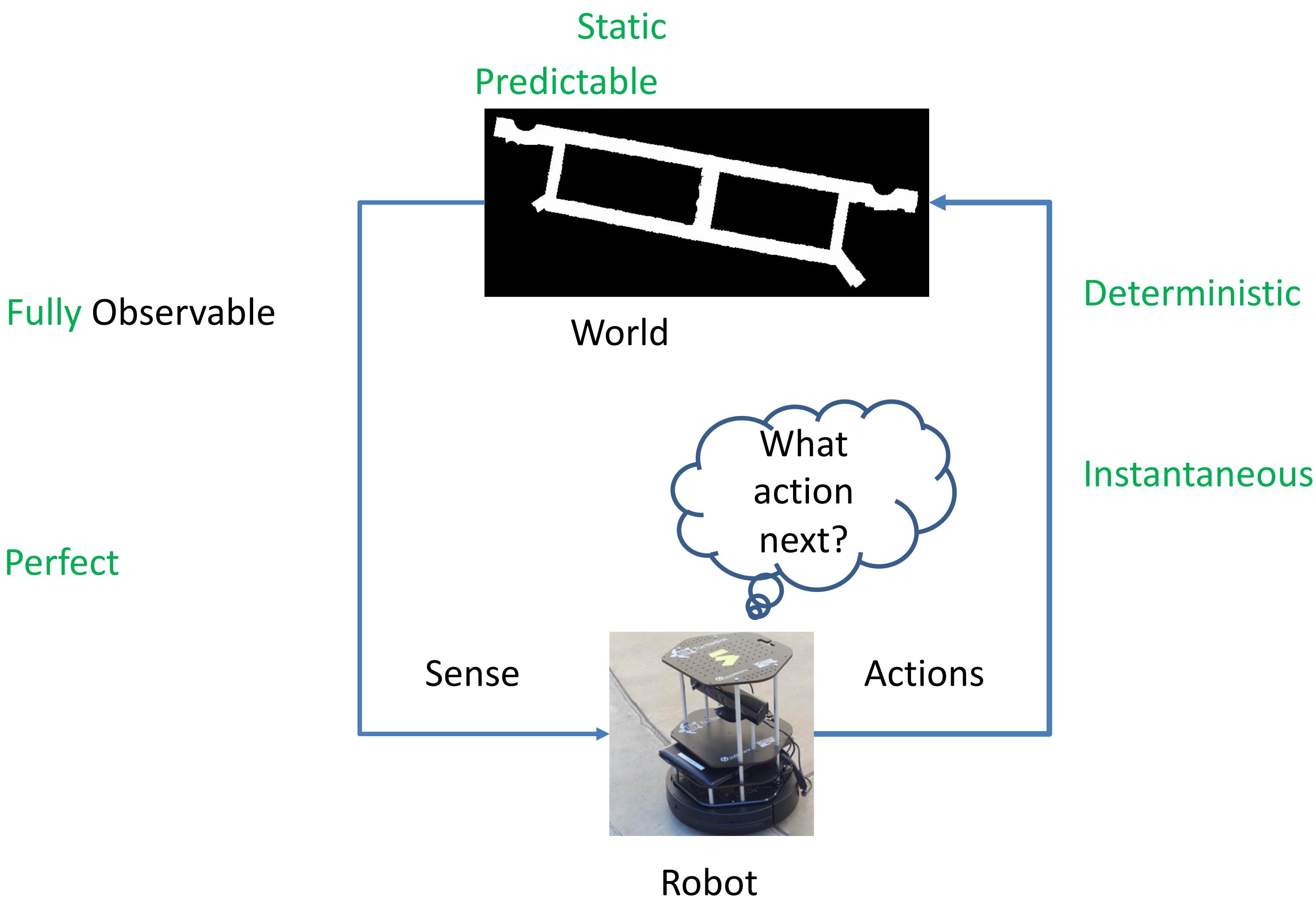
Fully vs. Partially Observable

Perfect vs. Noisy

Deterministic vs. Stochastic

Instantaneous vs. Durative

Classical Planning view

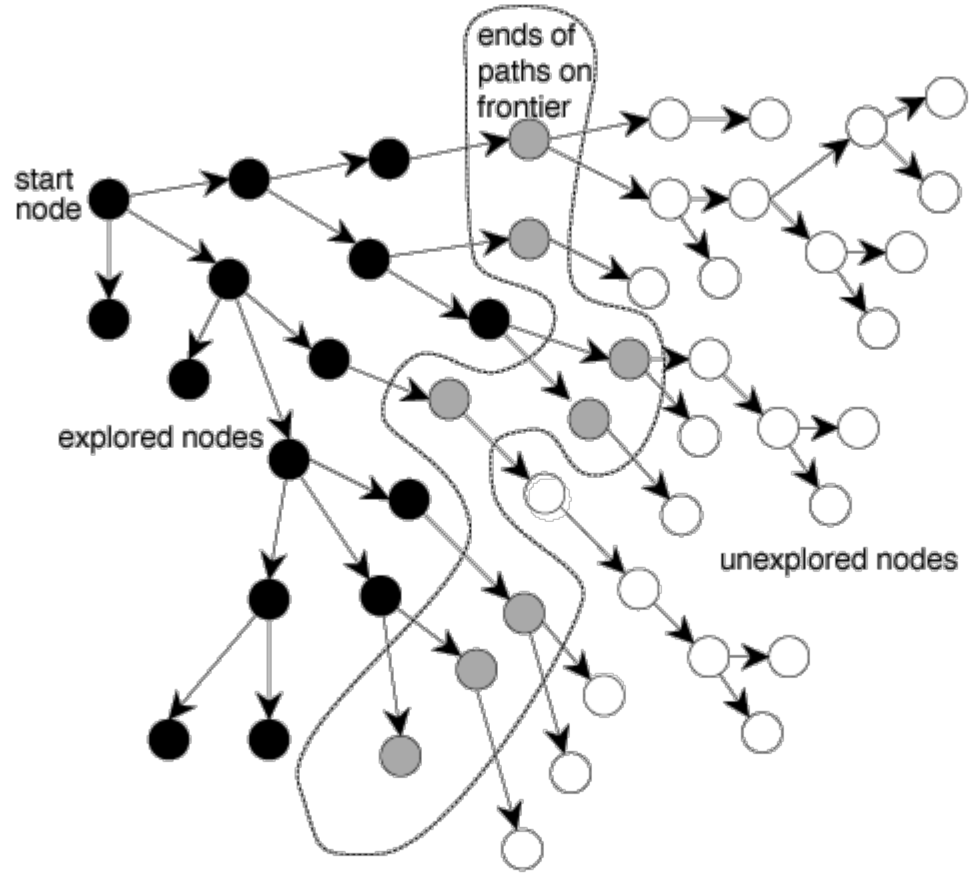


Solving planning problems by searching

- Search in discrete state spaces can be casted as a planning problem that can be defined by five components
  - *Initial state*, where the robot starts from
  - *Actions*, which can be performed by the robot
  - *Transition model*, given the current state and the action returns the new state
  - *Goal test*, to determine whether a state is a goal state
  - *Path cost*
- The solution is a plan/path, namely a *sequence of actions* from the initial state to the goal state

Solving planning problems by searching

- A planning problem can be casted as a graph search
  - Each state is a *node* in the graph
  - Each state-action pair is an *edge* in the graph



Source: artint.info

Problem-solving performance

- Classic planning algorithms search in the state space systematically

- A search algorithm can be evaluated according to:
  - Completeness: does the algorithm guarantee to find a solution if it exists?
  - Optimality: is the solution found optimal, according to optimality criterion/a?
  - Time complexity: computational time to find the solution
  - Space complexity: memory needed to perform the search

Basic tree-search algorithm

- Several search algorithms follows the following pattern

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

Source: [Russell and Norvig, 2016, Prentice Hall]

Uninformed search

- Breadth first search: expands nodes at the same depth from the initial state before going deeper
- Depth first search: expands the deepest unexpanded node
- …

Informed search

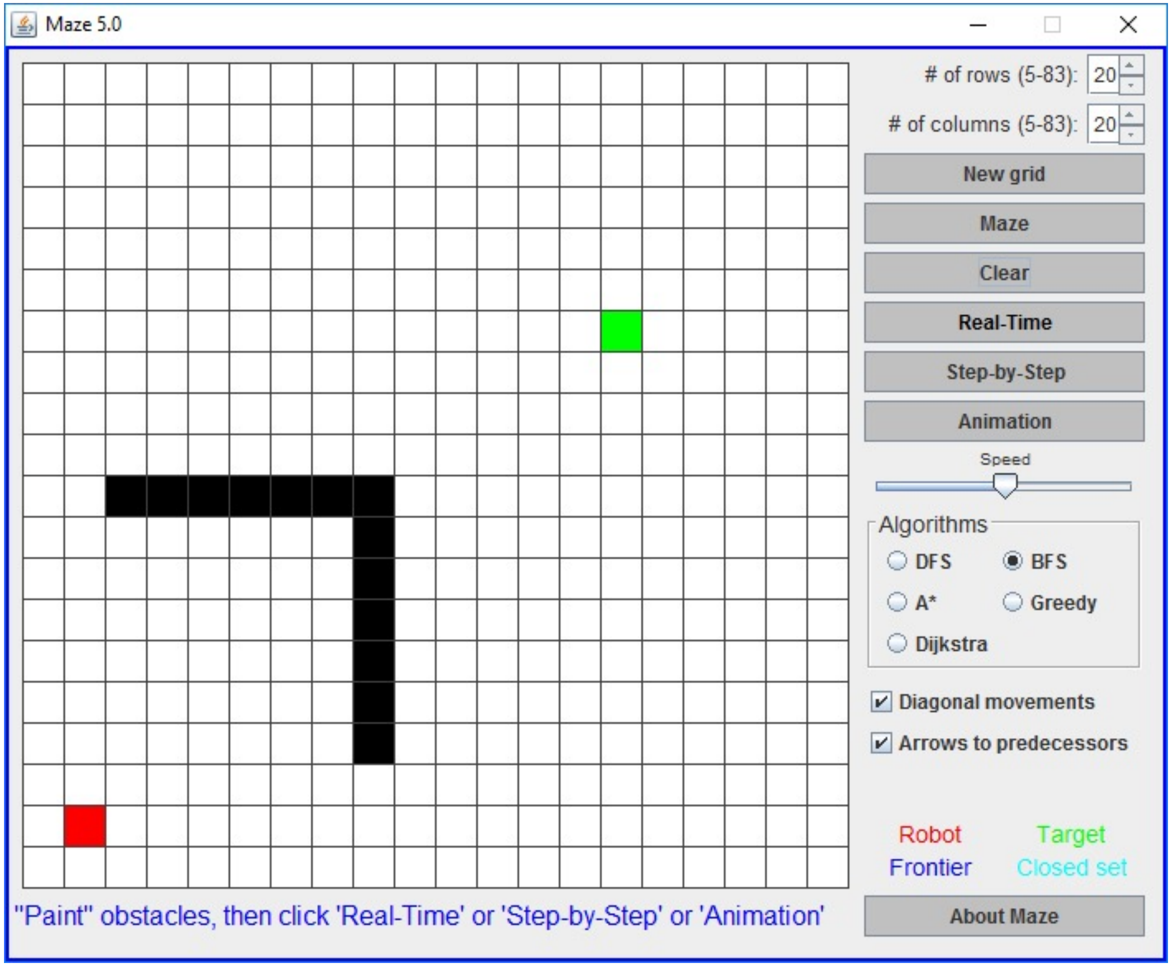
- Expansion of states can be performed by using
  - the cost $g(x)$ to get to a node x from the initial state
  - a heuristic function $h(x)$ that predicts the cost from a state x to the goal

Informed search

- Dijkstra's algorithm: the best node is selected according to the cost to get to the node
- Greedy best first search: the best node is selected according to a heuristic
- A*: expands node with minimal cost including a heuristic
- …

Example: path planning

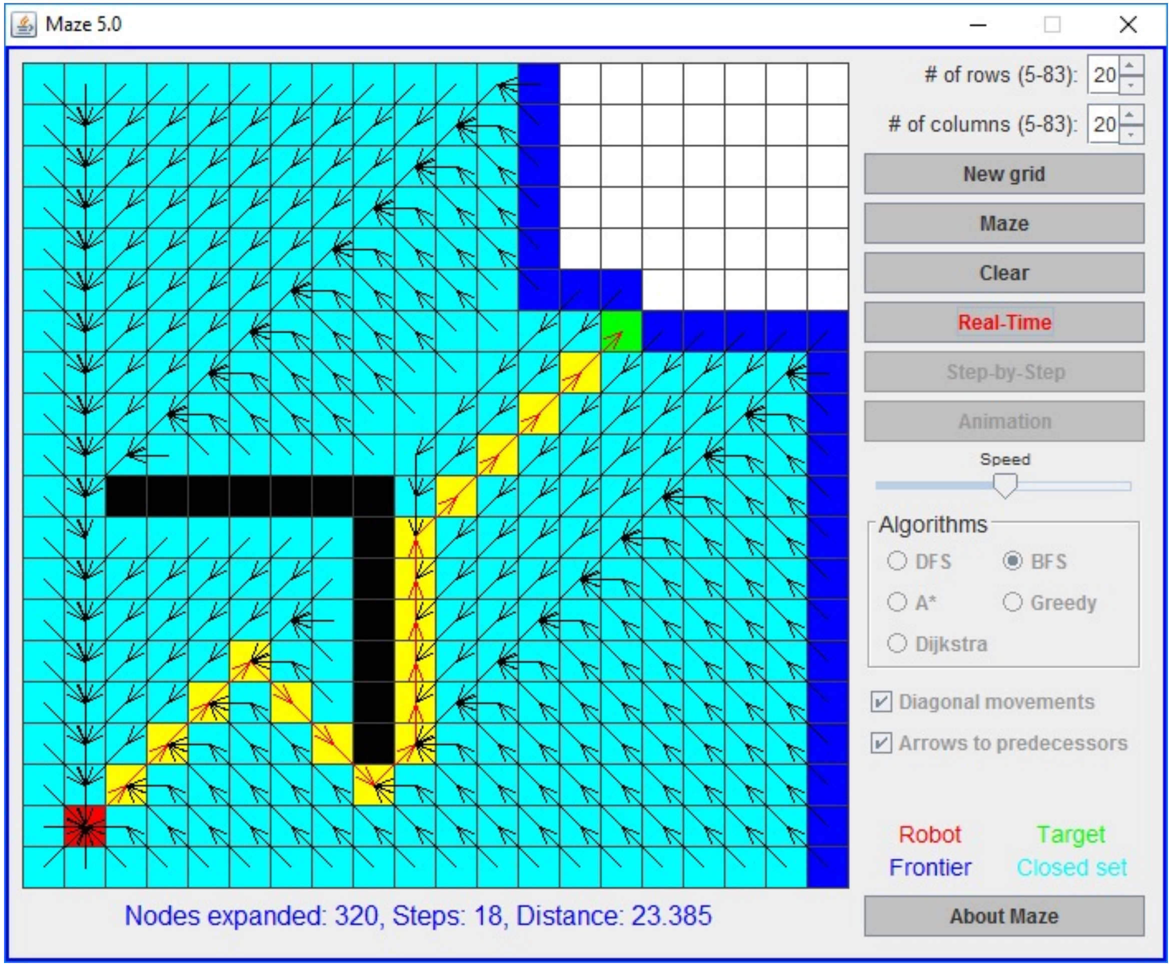
- Initial state: cell in red
- Action: up, down, left, right, diagonal left/right up/down
- Transition model: given a cell and an action, new neighbor cell (only if in free space)
- Goal test: is state in target (green)?
- Path cost: each step costs 1 or sqrt(2) depending on the action



Source: youtube.com/channel/UCmW8X0UX8U4VqO2MfjovY-A

Example: path planning

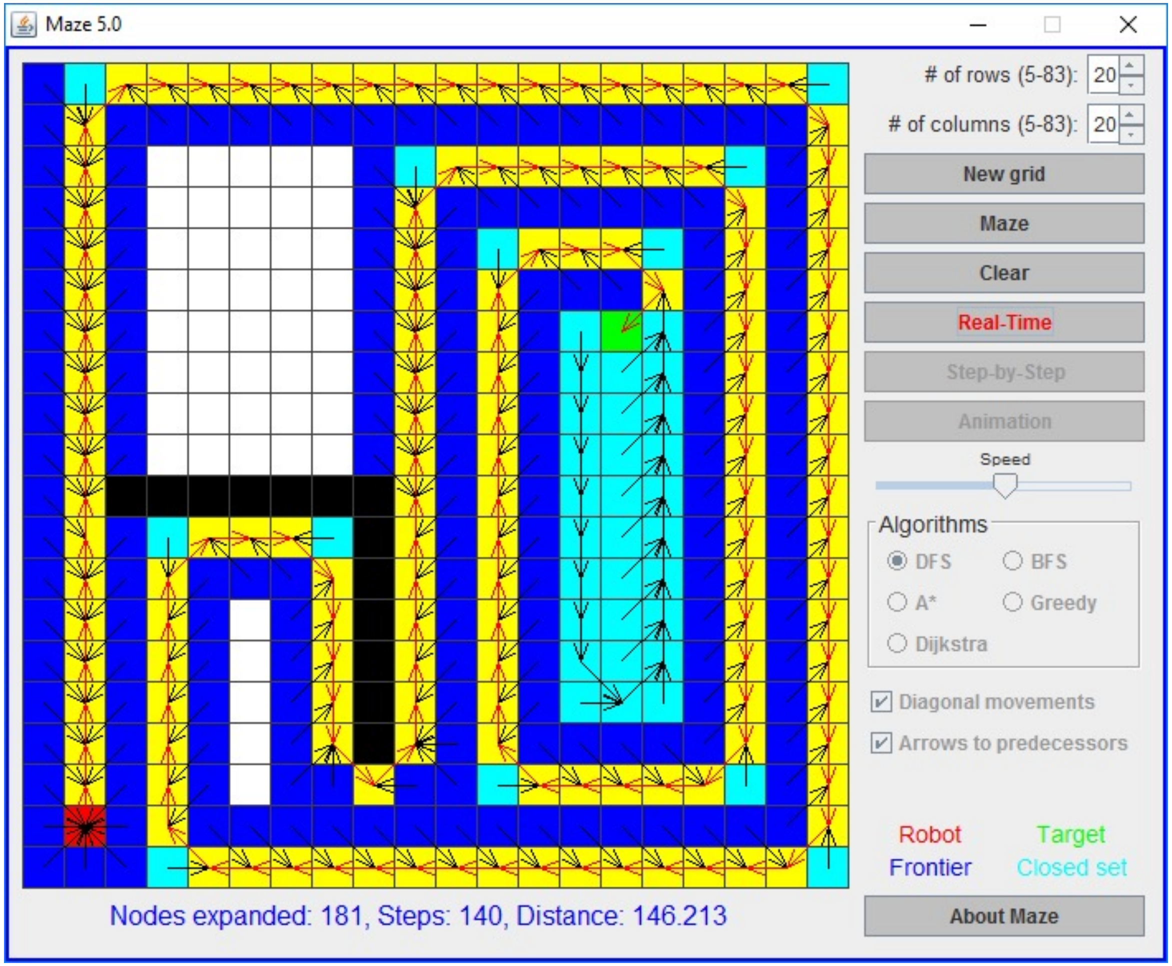
- BFS



Source: youtube.com/channel/UCmW8X0UX8U4VqO2MfjovY-A

Example: path planning

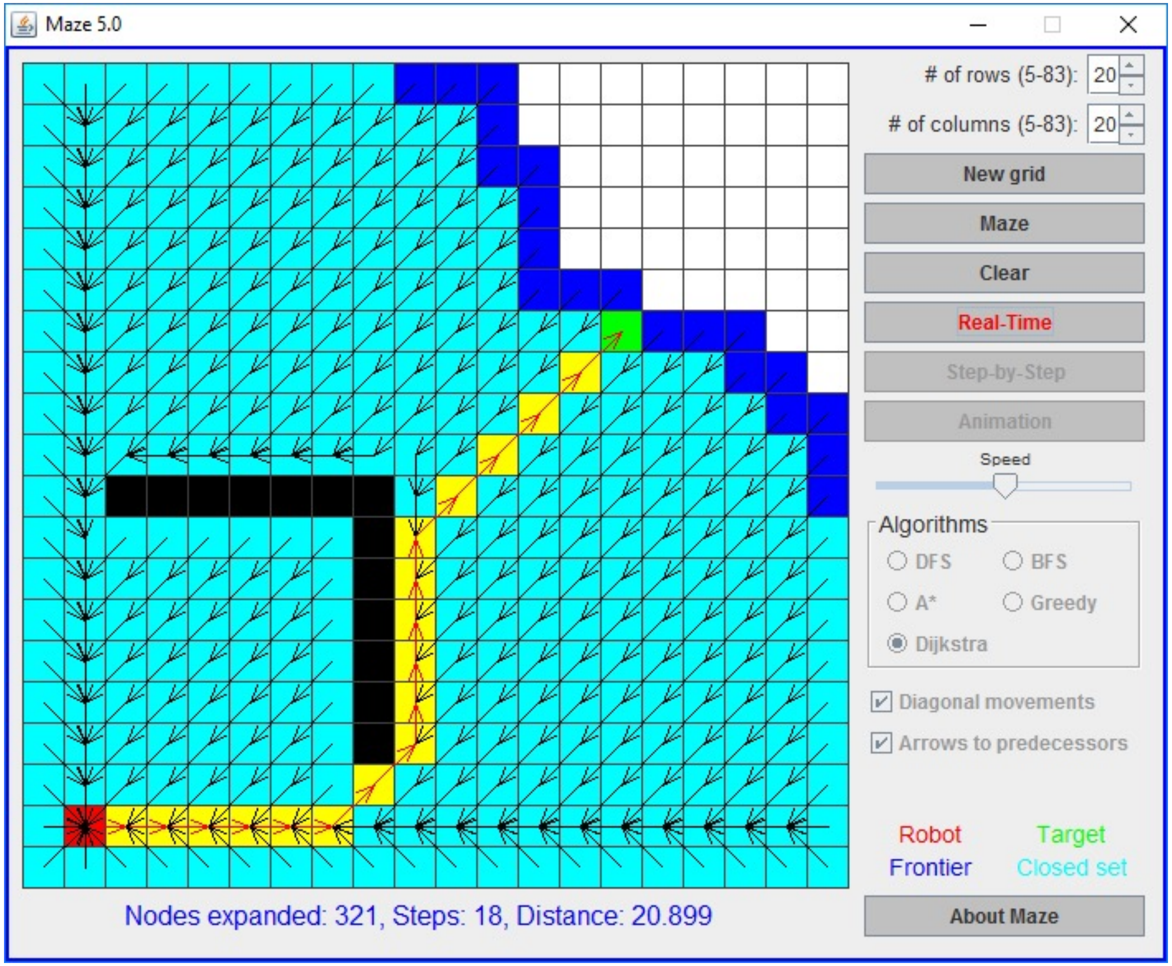
- DFS



Source: youtube.com/channel/UCmW8X0UX8U4VqO2MfjovY-A

Example: path planning

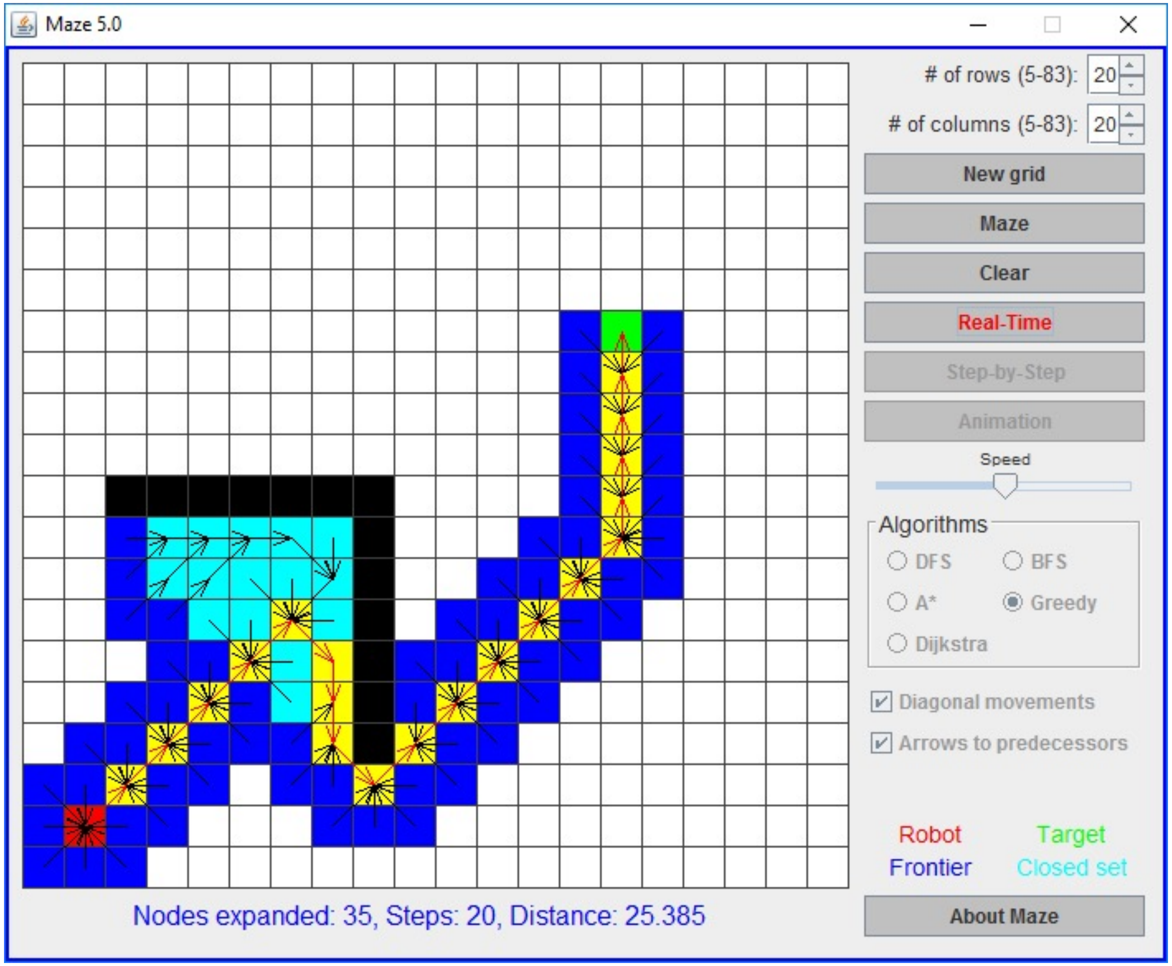
- Dijkstra



Source: youtube.com/channel/UCmW8X0UX8U4VqO2MfjovY-A

Example: path planning

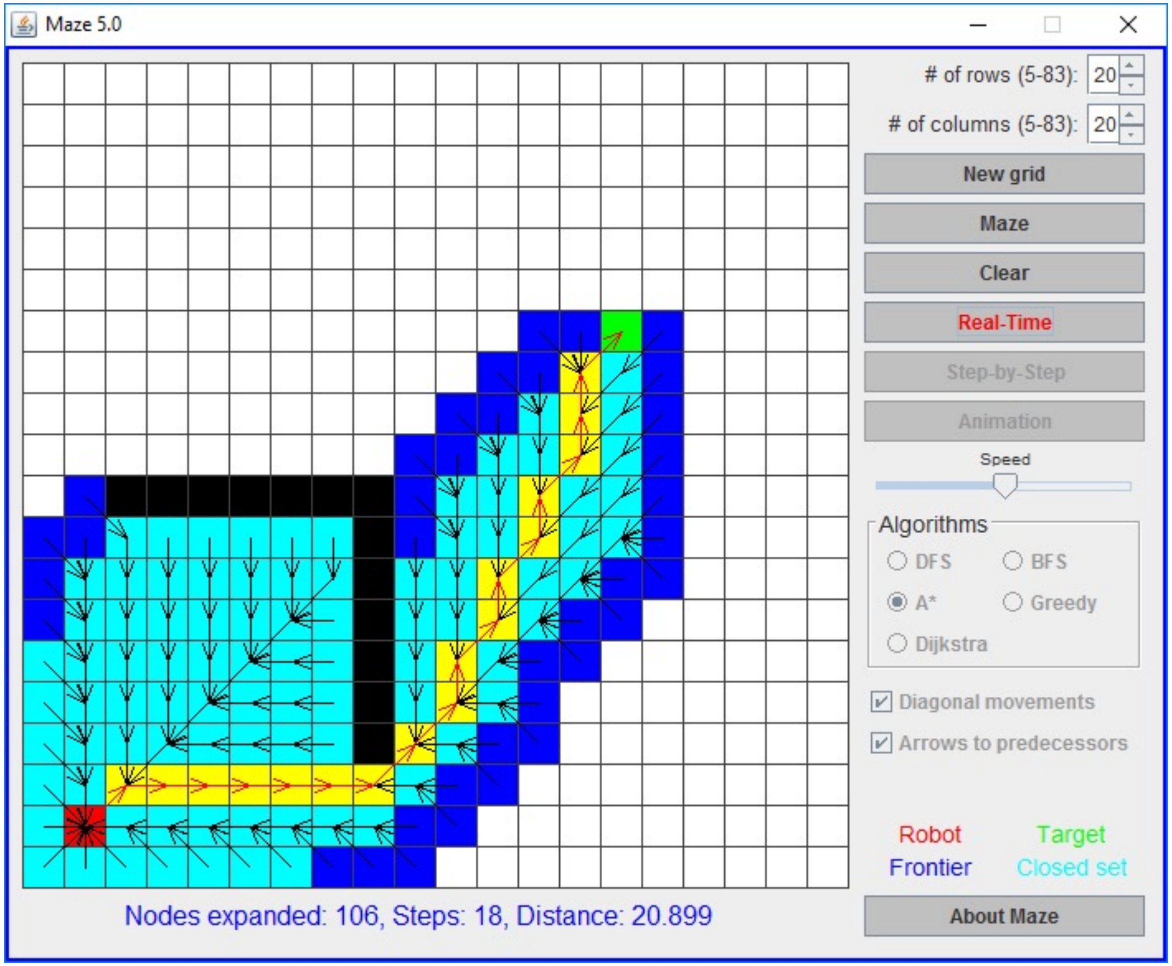
- Greedy



Source: youtube.com/channel/UCmW8X0UX8U4VqO2MfjovY-A

Example: path planning

- A*



Source: youtube.com/channel/UCmW8X0UX8U4VqO2MfjovY-A

Graph search

- If repeated states are not detected, a linear problem could become exponential
- The main idea is to keep track of expanded states

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set
```

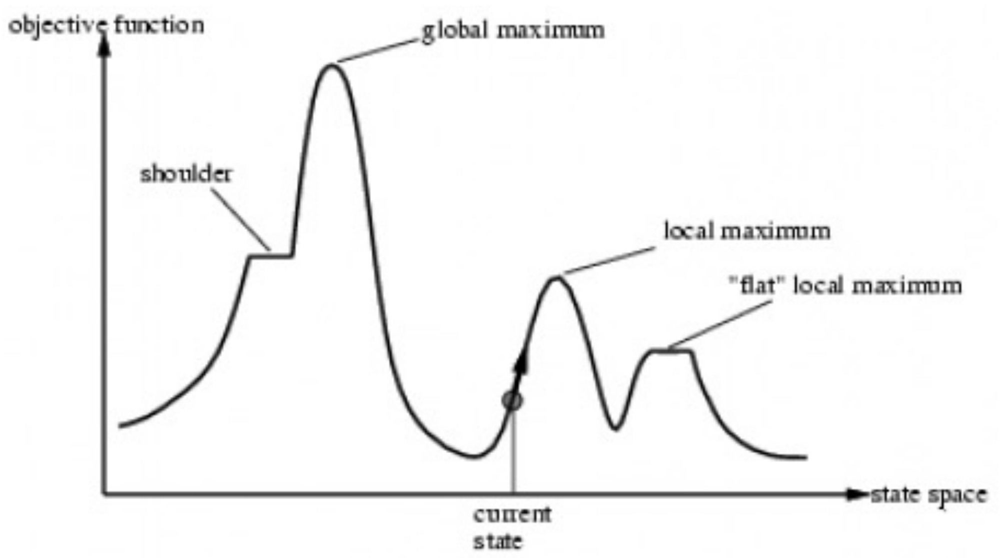
Source: [Russell and Norvig, 2016, Prentice Hall]

Sampling-based search

- Search space could be too big in some practical problems
- Sampling-based search algorithms select only some states
  - Randomly
  - Informed

Local search

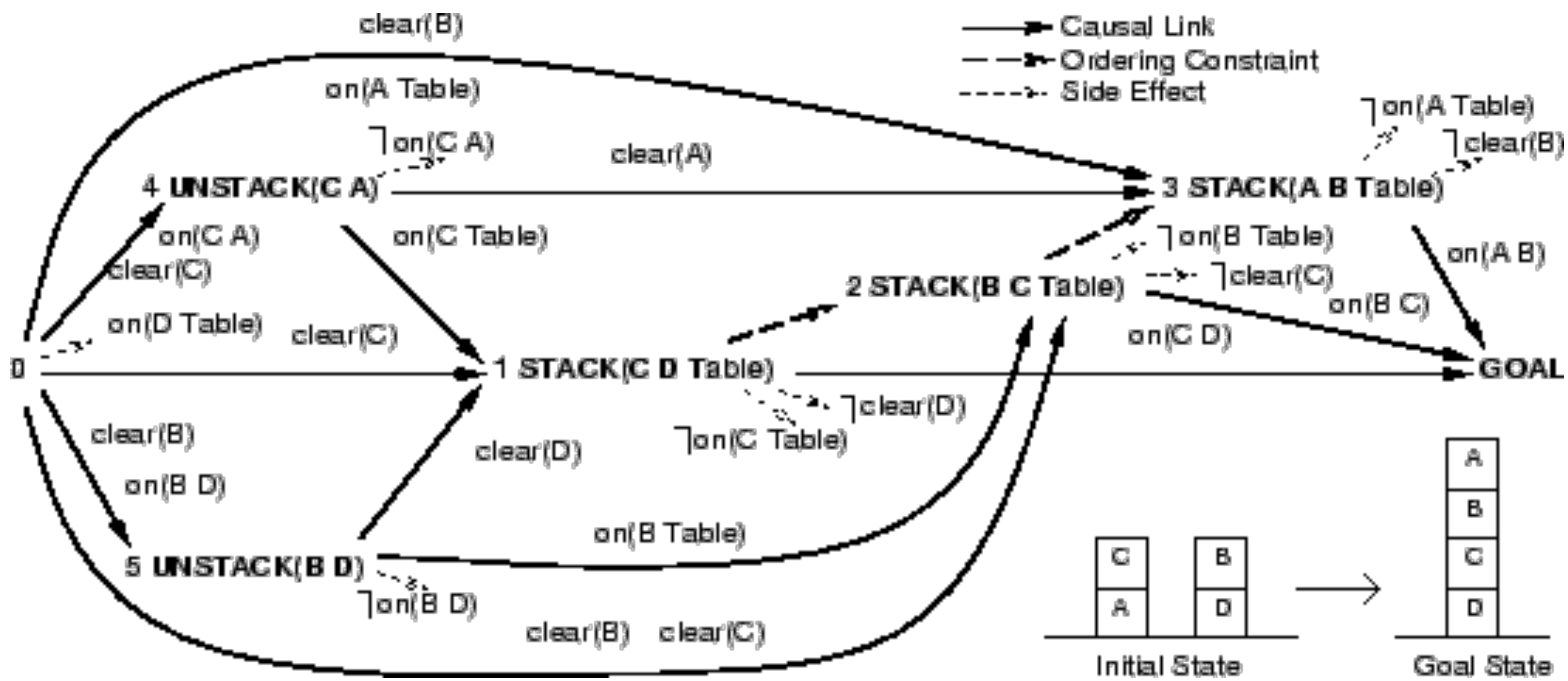
- Local search algorithms operate using a single current node and not storing paths
- Usually they are not guaranteed to be optimal and they suffer of the problem of local minima



Source: [Russell and Norvig, 2016, Prentice Hall]

Logic based planning

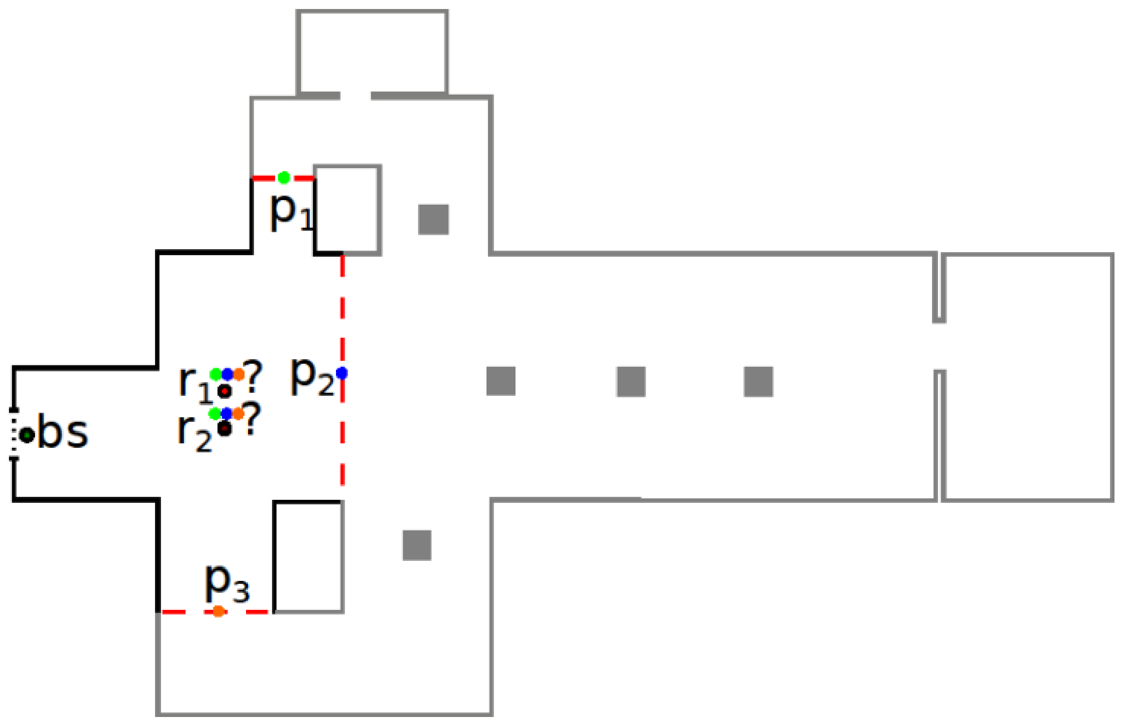
- AI Symbolic approaches used to solve plans



Source: cs.cmu.edu/afs/cs/project/jair/pub/volume15/ambite01a-html/node7.html

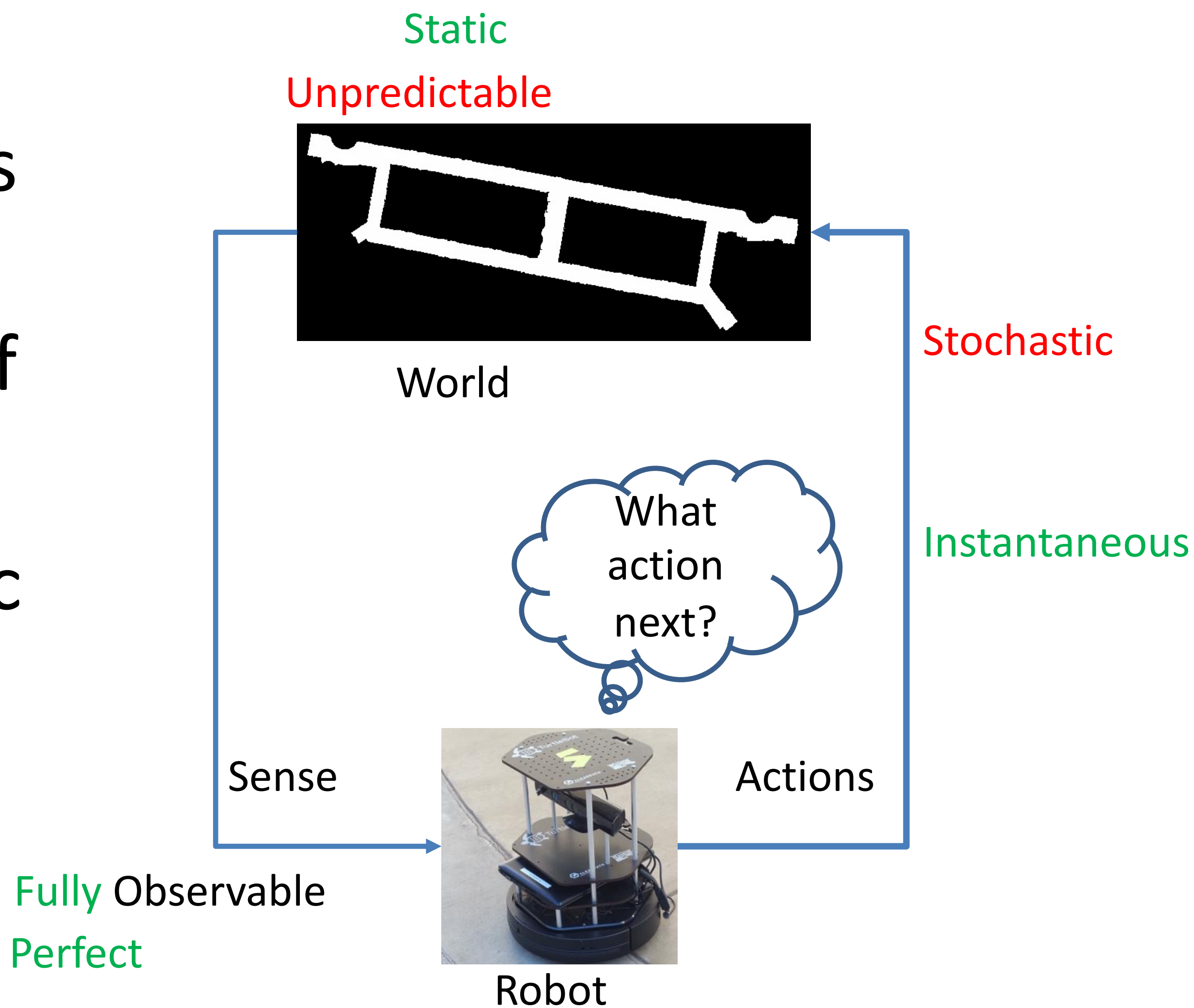
Online search

- An online search problem requires that a robot executes the action



Planning views

- Different planning views which involve different set of techniques
- E.g., Stochastic planning

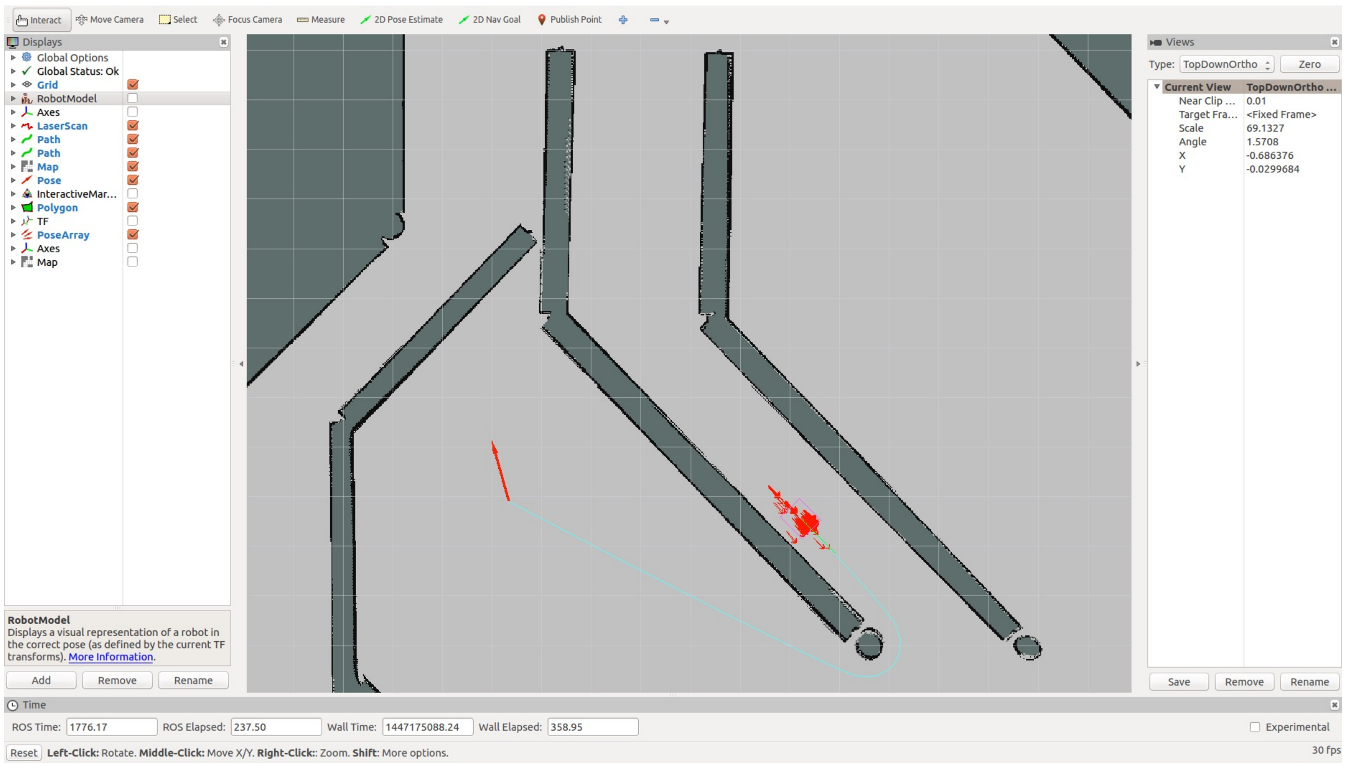


Deliberative architecture

- Drawbacks:
  – Time-scale: long time to search for a plan
  – Space: large memory can be occupied to calculate a plan
  – Information: world information should be updated

Examples – Path planning

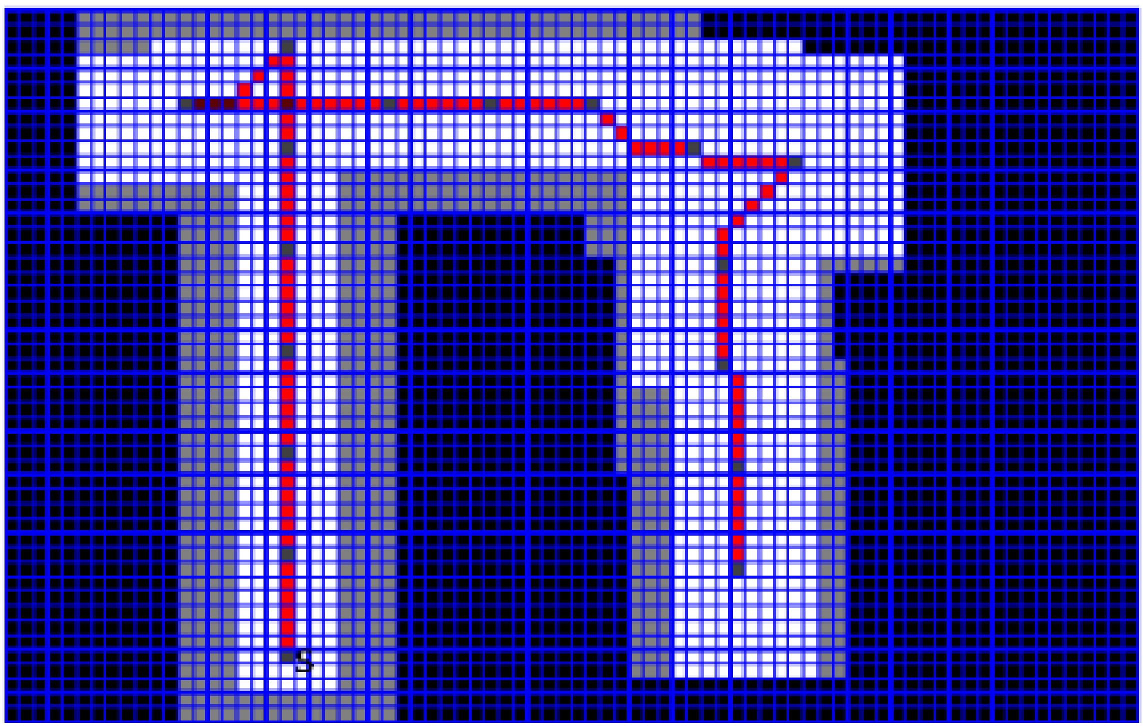
- Finding a path on an occupancy grid



Source: clearpathrobotics.com

Examples – Exploration

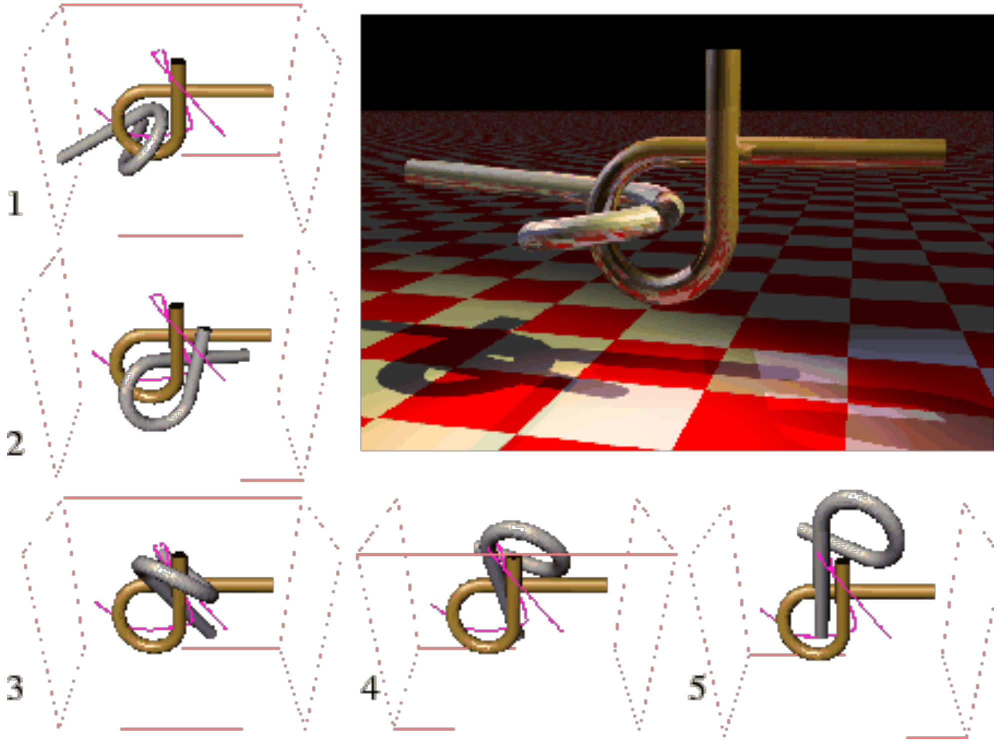
- Explore environment to build its map



Source: [Quattrini Li et al., 2012, AAAI]

Examples – Puzzle

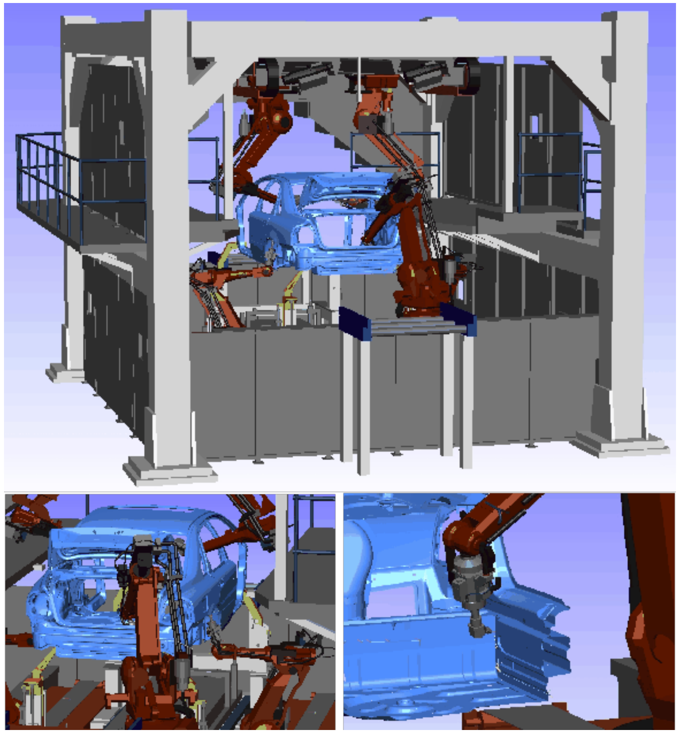
- Finding a way to pull this bars apart



Source: planning.cs.uiuc.edu

Examples – Assembly

- Sealing cracks in automotive assembly



Source: planning.cs.uiuc.edu

Reactive control architecture

- Reactive control architecture, differently from deliberative control architecture, is characterized by
  - A *lack of representation*
  - Not looking ahead at the possible outcomes
  - Responding only to sensors readings

- It should be multitasking to monitor different sensors

Table format

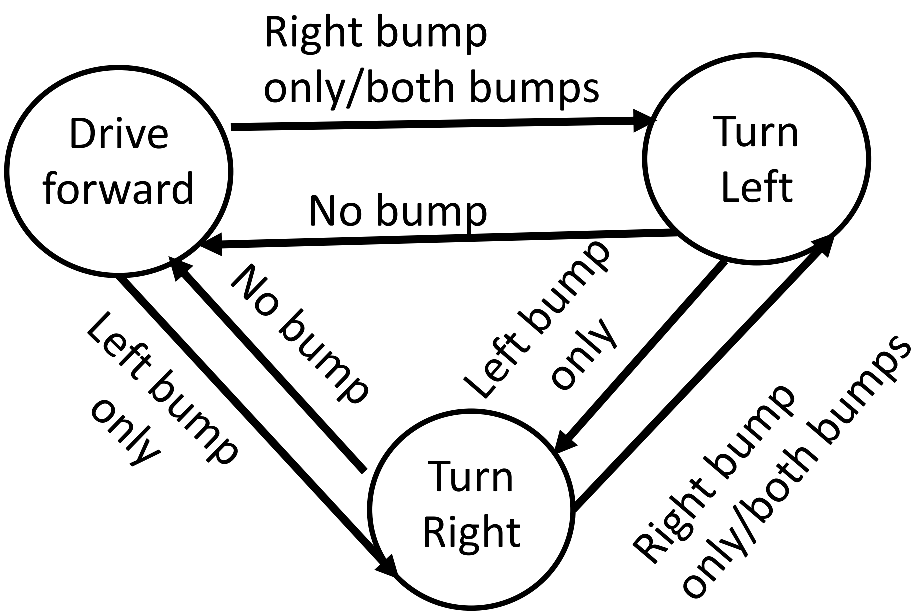
- A table that maps observation and actions can be used to describe reactive controllers

- E.g., a robot equipped with bumpers

Observation	Action
No bumps	Drive forward
Left bump only	Turn right
Right bump only	Turn left
Both bumps	Turn left

State machine

- Reactive controllers can be represented also with a state machine as directed graph
  - Each vertex is a state labeled with the behavior
  - Each edge shows the transition from one state to another



Note that there is an *Init* state that corresponds to when the robot is initialized.

How to define situations/states

- In case of sensors that return continuous values, it is unfeasible to represent every single value
- Some states should be defined taking into account intervals of values
- E.g., Robot with two sonar sensors, each of them at 45° wrt the motion direction of the robot

Observation	Action
Safe zone	Drive forward
Danger-zone left sonar only	Turn right
Danger-zone right sonar only	Turn left
Both bumps	Turn left

Subsumption architecture

- A way to organize a reactive controller is by following the subsumption architecture introduced by Prof. Rodney Brooks at MIT in 1985

- Subsumption consists of a collection of modules, each of which achieves a task
  - The design is bottom-up, from simpler to more complex

Limitations with reactive control architectures

- Some situations could lead robot to oscillate between two actions

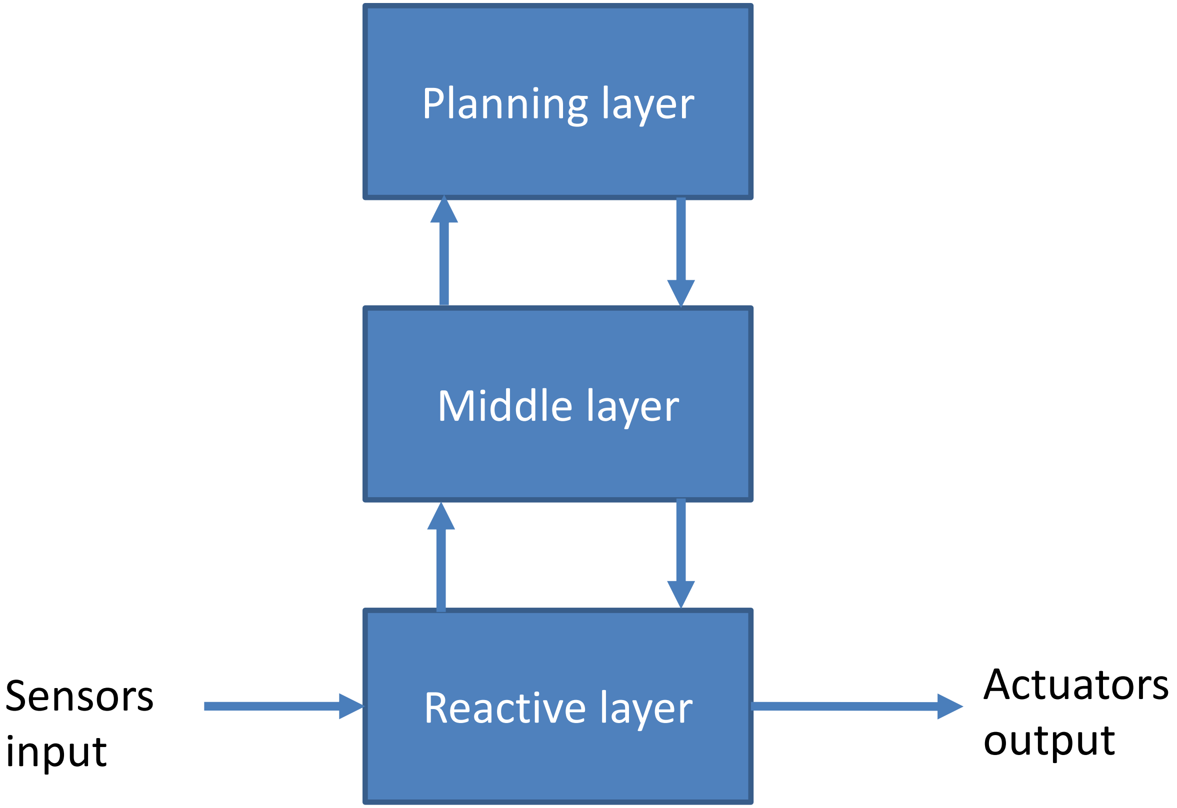
- To solve the problem
  - Include some randomness
  - Keep a bit of history

Hybrid control architecture

- Hybrid control architecture combines both deliberative and reactive control
- Hierarchical organization for the two control architectures
  - Deliberative control architecture in charge of planning some abstract actions
  - Reactive control architecture in charge of executing an abstract action

Three layer architectures

- A middle layer is necessary for linking the deliberative and reactive controls



Three layer architectures

- Replanning could happen
  - If deliberative layer finds a better plan
  - if reactive layer cannot proceed

- Plans could be generated online, as the reactive layer executes one abstract action

Hybrid control drawbacks

- Drawbacks include:
  - Middle layer hard to design and implement as it is usually special-purpose
  - Control can degenerate and the effectiveness of both could be minimal

Behavior-based control architecture

- Behavior-based control architectures are extension of reactive control architectures

- It uses "behaviors" as modules for control

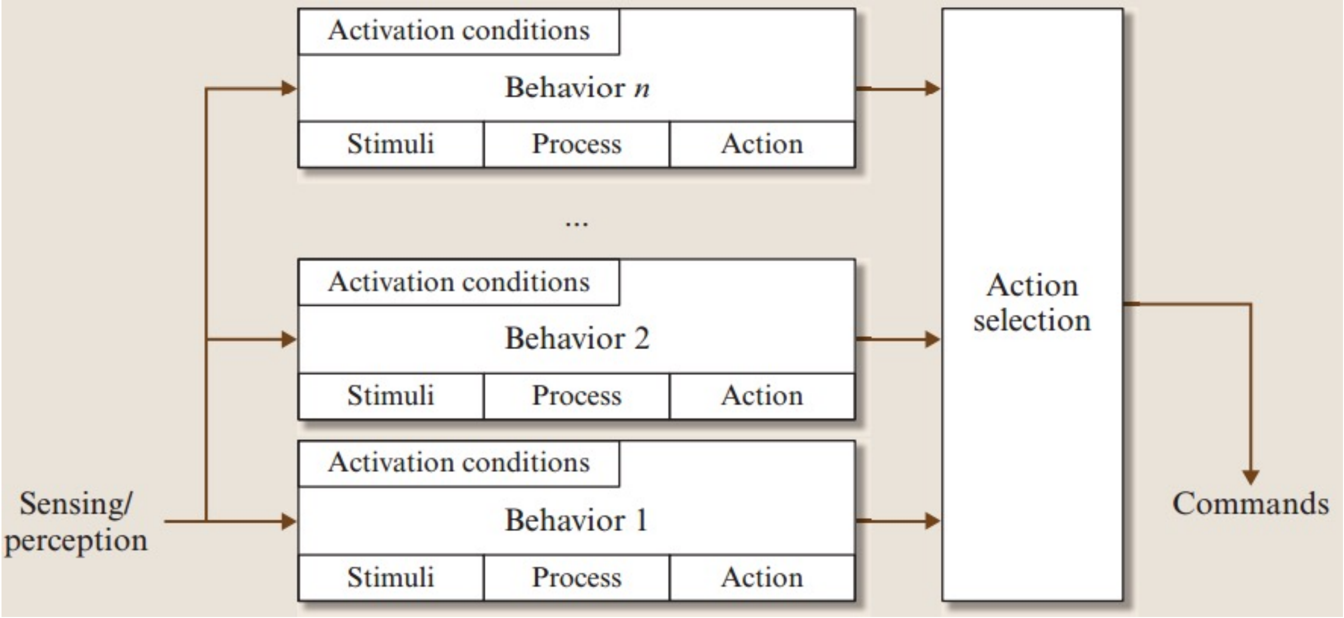
- A behavior
  - Achieves and/or maintain particular goals
  - Is time-extended, not instantaneous
  - Can talk to other behavior modules

Behavior-based control architecture

- Behaviors are typically executed in parallel
- Behaviors are operating on compatible time-scales
- Networks of behaviors are used to store state and to construct world models/representations

Behavior-based control architecture

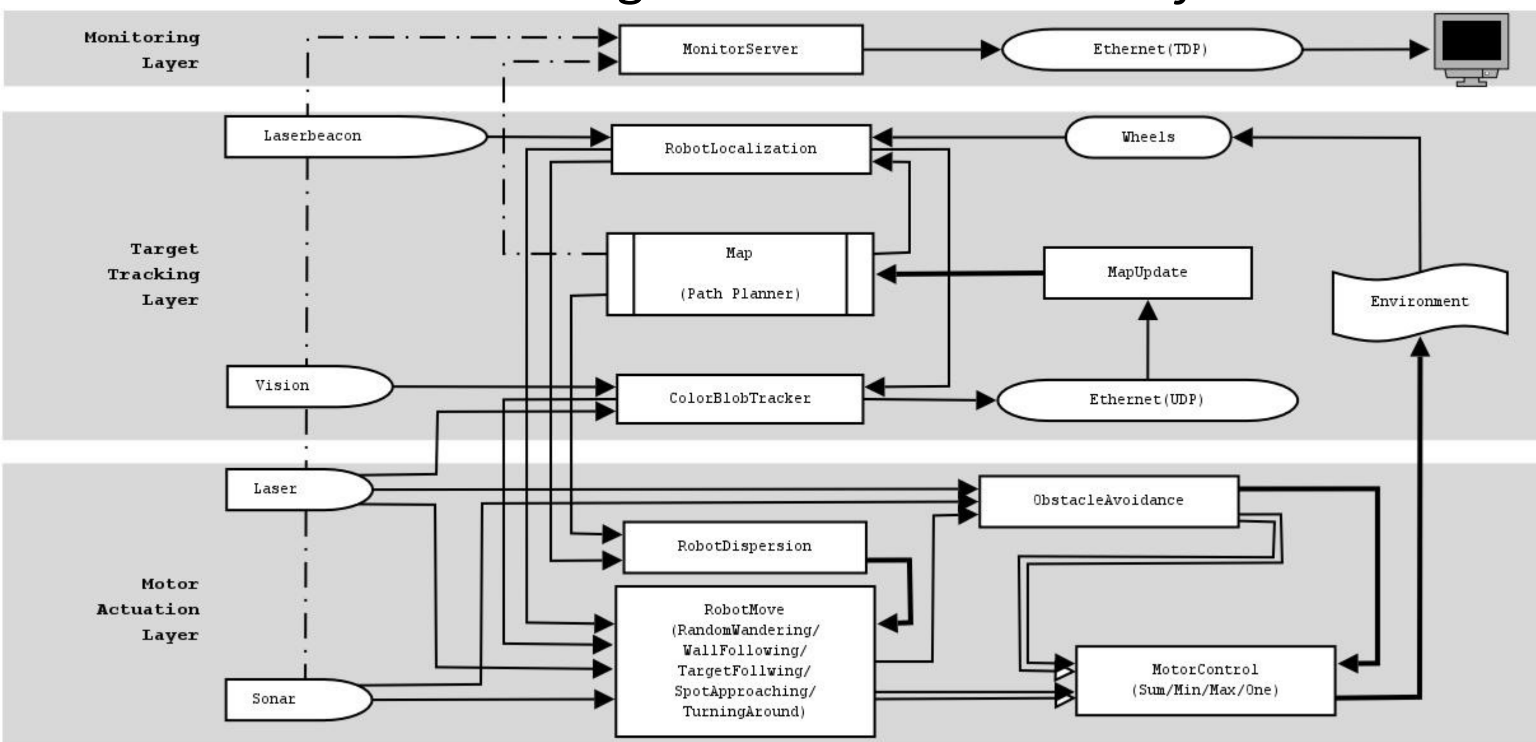
- Activation conditions allow behavior to generate actions
- Actions are generated from stimuli



Source: [Mataric and Michaud, 2008, Springer]

Behavior-based control architecture

- Behavior-based control can be viewed as a generalization of the subsumption architecture
- Each behavior can be designed at different *level of abstraction*



Source: robotics.usc.edu/~boyoon/rba.html

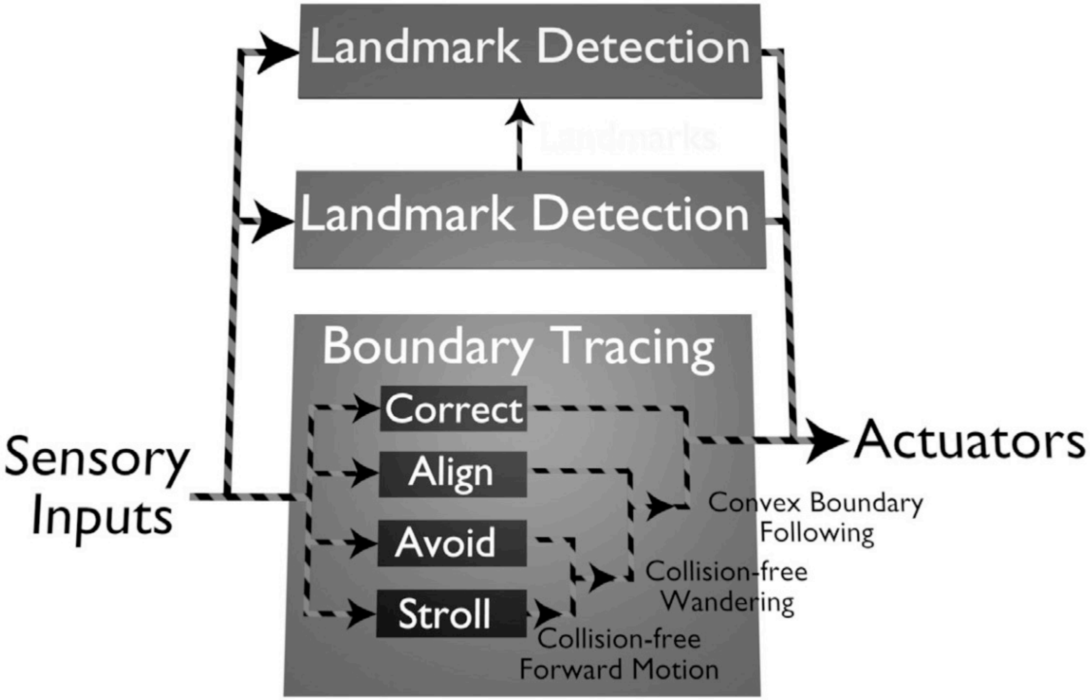
Example: distributed mapping

- Toto Robot (around 1990)
  - 12 sonars
  - compass

Source: [Mataric, 2007, MIT Press]

Example: distributed mapping

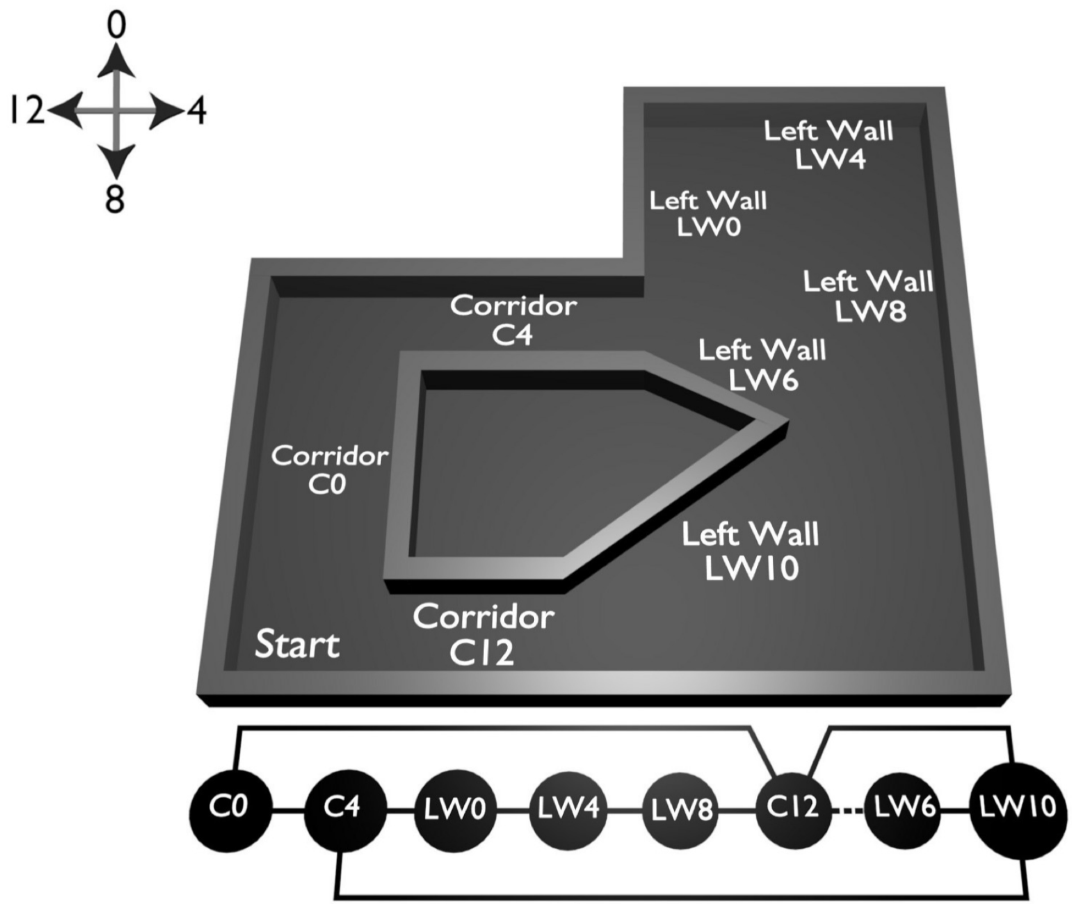
- Control diagram



Source: [Mataric, 2007, MIT Press]

Example: distributed mapping

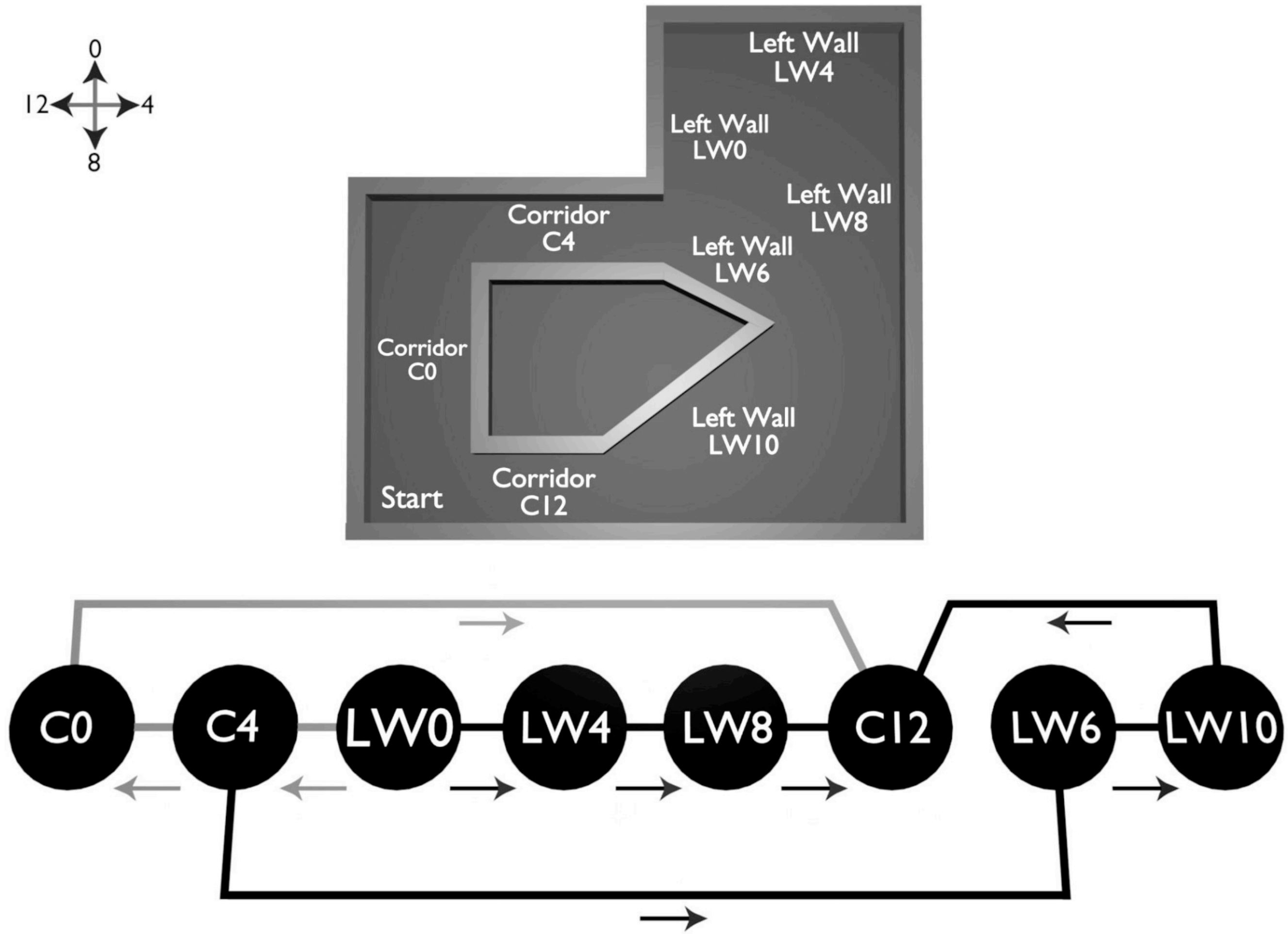
- Representation



Source: [Mataric, 2007, MIT Press]

Example: distributed mapping

- Path planning



Source: [Mataric, 2007, MIT Press]

Behavior coordination

- When more than one behavior is available, *behavior coordination* should be defined so that the robot knows what to do

Arbitration

- *Arbitration* process selects one action or behavior from multiple possible candidates
  - *Fixed priority hierarchy*
  - *Dynamic hierarchy*
- It is a competitive method
- It is used at higher level (e.g., high-level behaviors)

Fusion

- *Behavior fusion* is the process of combining multiple possible candidates actions or behaviors into a single output action/behavior

- It is a cooperative method

- Used at lower level (e.g., velocities)

Behavior-based vs reactive

- Behaviors can store a representation of the world by utilizing a distributed network of behaviors
- It has learning capabilities

- Reactive control architecture does not use any representation of the world
- It does not have learning capabilities

Behavior-based vs hybrid

- Usually multirobot
- Layers do not drastically differ in timescale

- Usually single robot
- Layers drastically differ in timescale

- Organized in layers
- Both look ahead

Emergent behavior

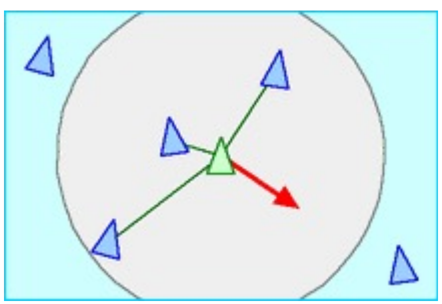
- Emergent behavior is structured, patterned, or meaningful behavior that is apparent from an observer’s viewpoint, but not from controller’s viewpoint

- Some emergent behaviors could be desirable and good, while some others could be bad

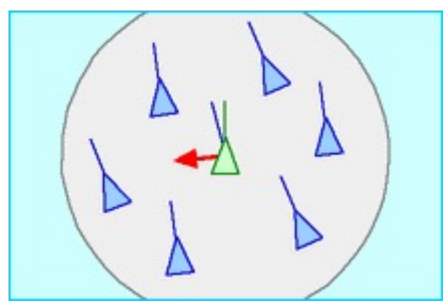
Flocking behavior

- Flocking motion, a collective motion of a large number of entities, is an example of emergent behavior
  - Robots move as a group using only local information

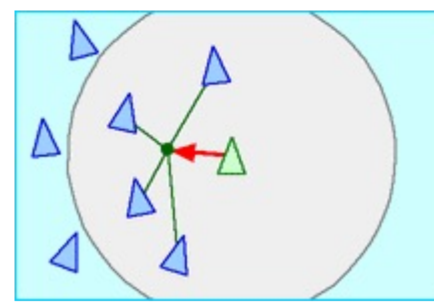
Source: red3d.com/cwr/boids/



Separation: steer to avoid local mates

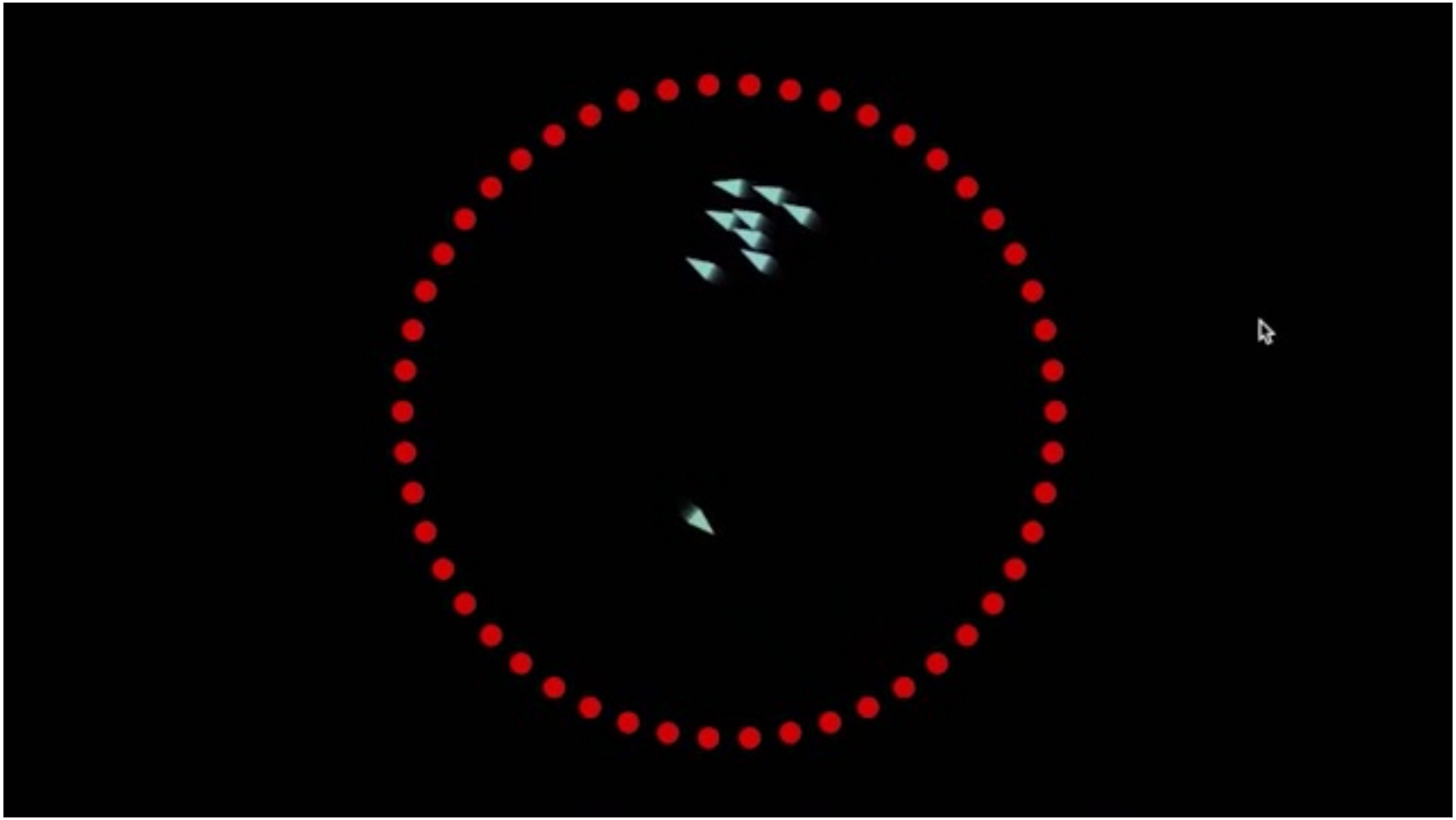


Alignment: steer towards average heading



Cohesion: steer to move toward the average position of local mates

Flocking behavior



Source: youtube.com/watch?v=QbUPfMXXQIY