Lecture 5: Intelligent Control, Part 2

Elizabeth Sklar

Department of Informatics
King's College London

23 October 2018
(version 1.0)

Today

Navigation

Cognition

Behaviour Based Systems

Multi-Robot Systems

Today: Intelligent Control, Part 2

- Navigation
- Cognition

- Behaviours
- Multi-Robot Systems

Today

Navigation

Cognition

Behaviour-Based Systems

Multi-Robot Systems

What counts as navigation?

- Navigation is concerned with how a robot gets around the world.

- Navigation is more than just blundering about using dead reckoning.

- We assume that the robot:
  - Knows where it is.
  - Knows where it is going.

- We are concerned with getting the robot from one place to another.

- We distinguish between two kinds of navigation:

  - Global navigation
  - Local navigation

Navigation, p2

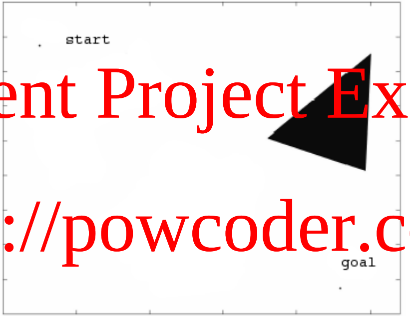
- Global navigation is about deciding how to get from some start point to a goal point.
- The robot plans in some sense.

- We will look at methods for path planning.
- In short, the robot comes up with something like a "plan" — a series of steps — to get it from its current location to its goal.
  - The "plan" is typically a sequence of waypoints.

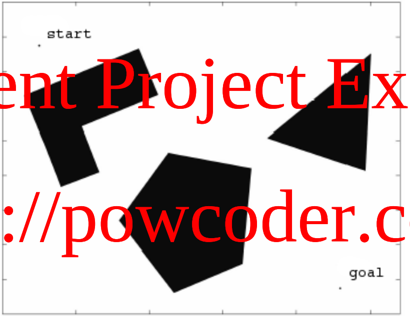
- We will look at some different methods that are appropriate for different map representations.
  - Remember them?

- Local navigation is about obstacle avoidance.
  - If there are objects in the way, make sure the robot does not hit them.

- There are a range of different approaches depending on what kind of information the robot has about the world.
  - Depends on sensors

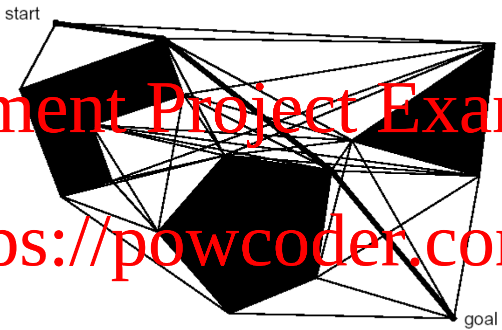


- One way to think about the difference between the two is in terms of the relationship between the robot's start point and the goal point.
- If there is a clear line of sight between the start point and the goal, then we only need to worry about obstacle avoidance.
  - Just avoiding some debris that isn't on the map



- However, if there is no line of sight from start to goal, then we have to find a path.
- Typically path segments will be between two points between which there is a line of sight.
  - Path segments connect waypoints

Visibility graph

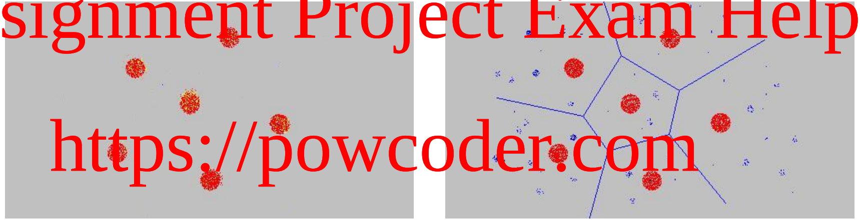


- Direct implementation of line-of-sight.
- Connect up all the vertices in the map.
- Given the line segments, look for the shortest path from start to goal.
- Then translate the path into a series of waypoints (i.e., the end points of the line segments).

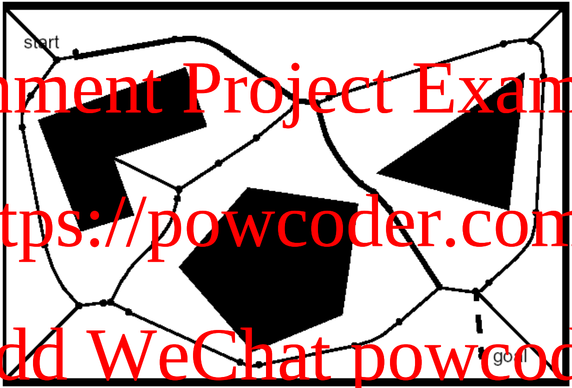
Visibility graph, p2

- Given the visibility graph on the previous slide, there is an obvious problem with using the lines as a guide for where the robot should go.
- What is it?
- Routes at the moment run arbitrarily close to the vertices of objects.
  - Problems with collisions.
- Fix this by expanding objects by enough that the robot will still clear them.
  - More than half the diameter of the robot.

Voronoi diagram



- A Voronoi diagram is a way to divide up a plane (a map)
- Given a set of points P, a Voronoi diagram is a set of polygons such that the points inside each polygon are closer to one member of P than any other.



- Can extend this to cases where P is a set of objects.
- Treat the line segments exactly like the edges in the visibility graph.

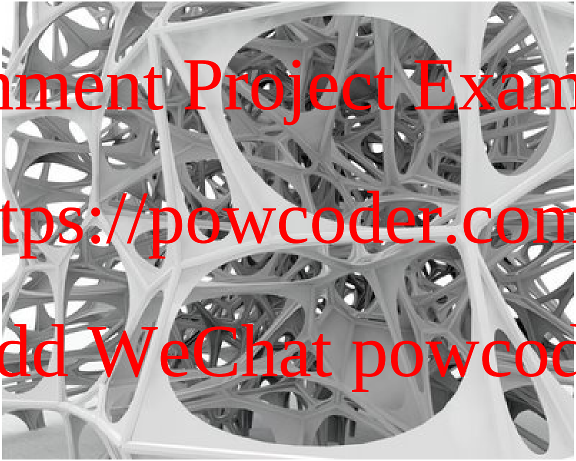
- The lines are not necessarily lines of sight
  - As above they may bend.

- However, they are object free, and so can be followed just like lines of sight can.

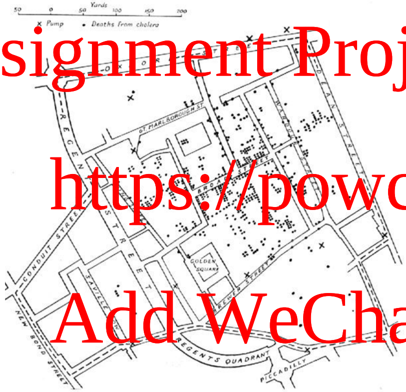
- Voronoi diagrams also have a nice property in terms of path-following.

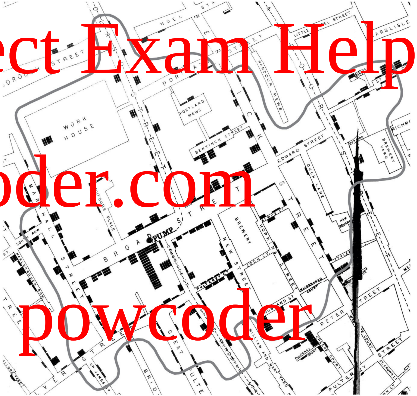
- A robot that is maximising its distance from objects will follow the lines in the Voronoi diagram.
- This means that we can again reduce the path to a set of waypoints.
  - Head to the next waypoint while maximising distance from objects (obstacles).

Asides



- Voronoi diagrams work in 3D also.

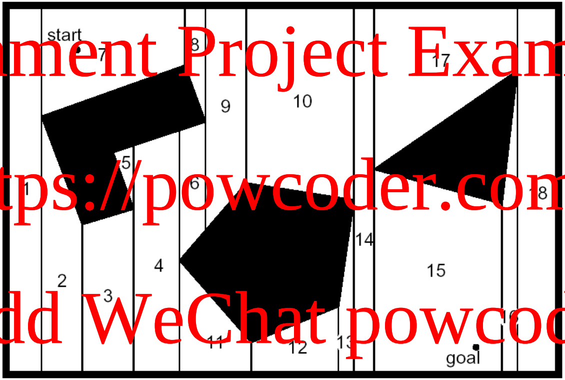




- Voronoi diagrams were also famously used by John Snow to identify the source of the 1854 cholera epidemic in London.

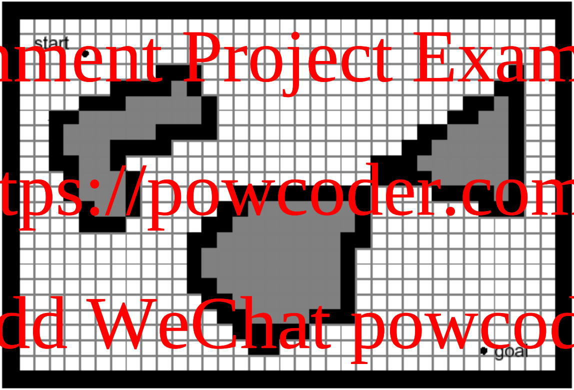
- Previously we talked about a variety of different cell-based maps...

Cell-based maps, p2

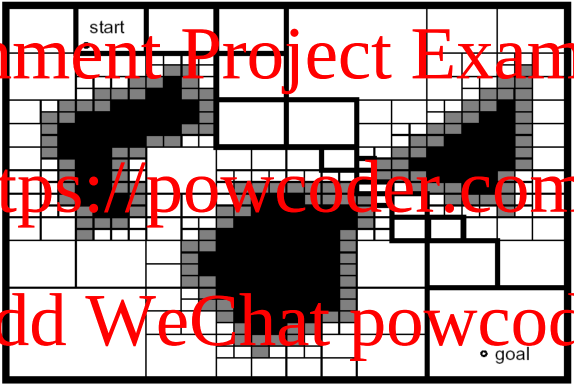


- Exact cell decomposition

Cell-based maps, p3



- Fixed cell decomposition

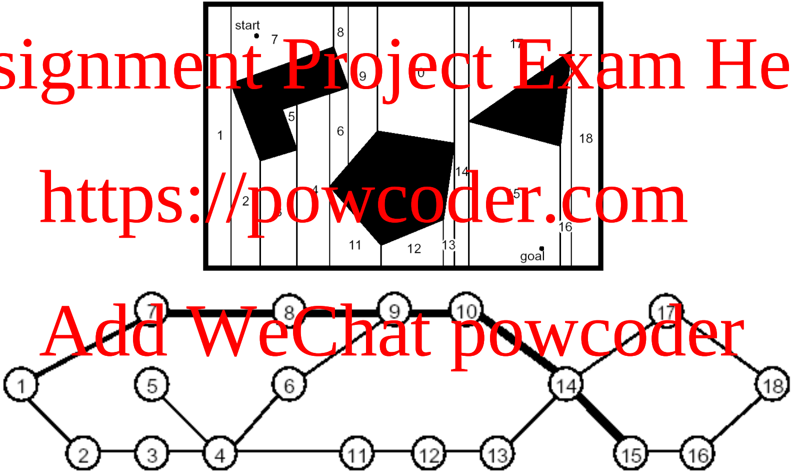


- Adaptive cell decomposition

- Given the maps, we still want to figure out a sequence of line segments.

- Not quite so straightforward for cell-based maps.
- We will look at two general approaches to do path-finding:
  - Explicit search of a connectivity graph
  - Wavefront planning

- These are really the same thing in different guises.



- We need to identify which cells are next to which other cells.

Connectivity graph, p2

- The question is how to figure out a path from the graph.
- When the graph is complex, we need to use search techniques.
- This is also the case for the connectivity graphs we get automatically from the visibility graph or Voronoi diagram approaches.

- Standard approaches to search:
  - Breath first
  - Depth first
  - A*

- Plus there are robotics-specific approaches like D*.

- A general algorithm for search is:

```
agenda = initial node;
while agenda not empty do {
    state <- node from agenda;
    new nodes = nodes connected to state;
    if goal in new nodes
    then {
           return solution;
      }
    add new nodes to agenda;
}
```

- Note that this doesn't generate a set of waypoints, it just looks for the goal state.
- In fact, it assumes that there are already set of possible waypoints.

- Let's think about how this search would work on the connectivity graph:

- To use the algorithm we need to decide how to do the selection in this line:


```
state <- node from agenda;
```

and how to do the addition in this line:

```
add new nodes to agenda;
```

Depth-first search:

- Takes the first node on the agenda;
- Adds new nodes to the front of the agenda.
- Leads to a search that explores “vertically”.

Breadth-first search:

- Takes the first node on the agenda;
- Adds new nodes to the back of the agenda.
- Explores all the nodes at one “level” before looking at the next level.

- A* search focuses the search by giving each node a pair of weights:
  - How far it is from the start; and
  - How close it is to the goal.

- The cost of the node is then the sum of the weights.
- We pick from the agenda by choosing the node with the lowest cost.

  (Choosing like this means we don't have to worry about what order we put nodes onto the agenda).

- In some domains we have to design clever functions to determine what "far" is.

- In robotics we can just use Euclidean or Manhattan distance between points:
  - Euclidean distance

$$d^e_{s,g} = \sqrt{(x_g - x_s)^2 + (y_g - y_s)^2}$$

  - Manhattan distance

$$d^m_{s,g} = |(x_g - x_s)| + |(y_g - y_s)|$$

- Of course the distance to the goal may be an underestimate (may be no route through), but it turns out that this is a good thing for A*.

- Often in robotics we need to replan.
- D* is a version of A* that keeps track of the search that led to a plan and just fixes the bits that need to be fixed.

- Quicker than replanning from scratch.
  - Usually have to replan from the robot to the goal and the only change is near the robot.
  - That is where the robot senses failure

- In all these approaches we have to extract the waypoints after we find the goal.
- First we identify the sequence of cells.

  - As we search we can build a plan for each node we visit.
  - The plan for each node is the route to its parent plus the step to the node.
  - When we get to the goal, we have the plan.

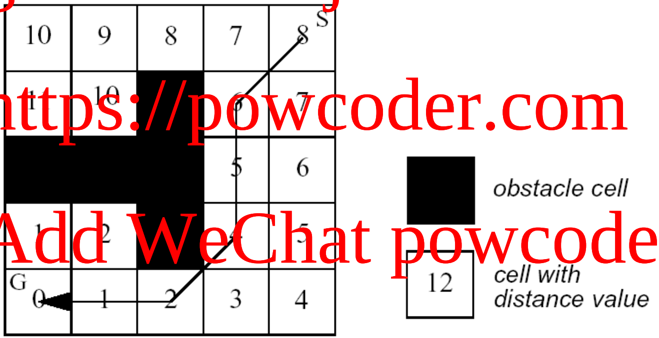
- Then we build a waypoint from each grid cell
  - Typically the centre of gravity of the cell.

- Also known as Grassfire, wildfire or NF1.
- Essentially breadth-first search in a convenient form for application to grid-based maps:
  1. Start at the cell containing the goal and label it 0.
  2. Take every unlabelled cell that is next to a cell labelled n and label it $n + 1$.
  3. Repeat until the cell containing the start is labelled.
- Then read the sequence of cells to traverse by following the labels down from the start, choosing the lowest numbered label at each step.

- Works especially well with occupancy grids, where the obstacles are already factored into the map.

Wavefront planning, p2

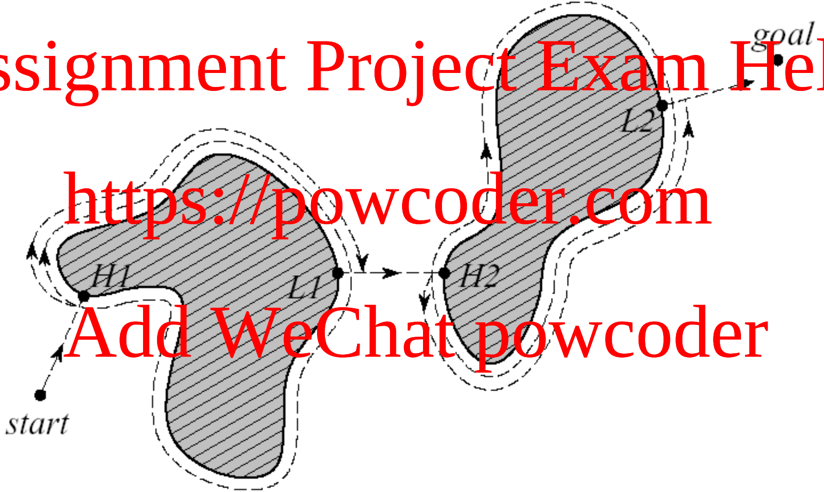
- Here's an example:



- Bug algorithms assume localization but no map.

Bug 1 Algorithm:

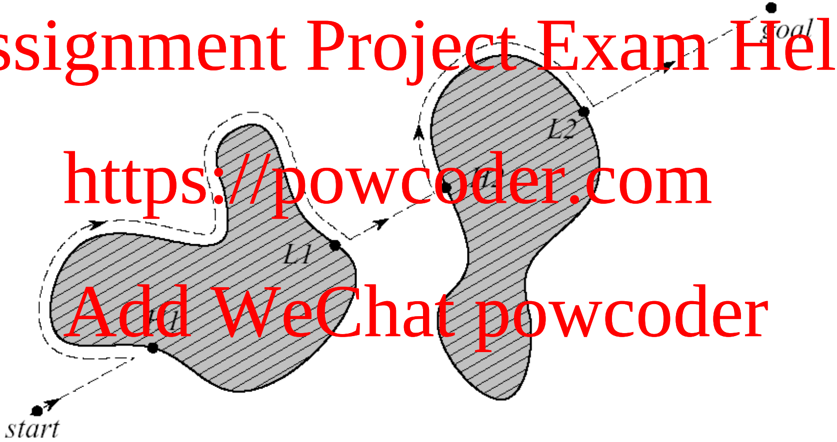
- When you meet an obstacle you follow around the edge.
- Leave the obstacle at the point closest to the goal.
- Circle the obstacle to be sure that you know where this point is.


goal
L2
H1
L1
H2
start

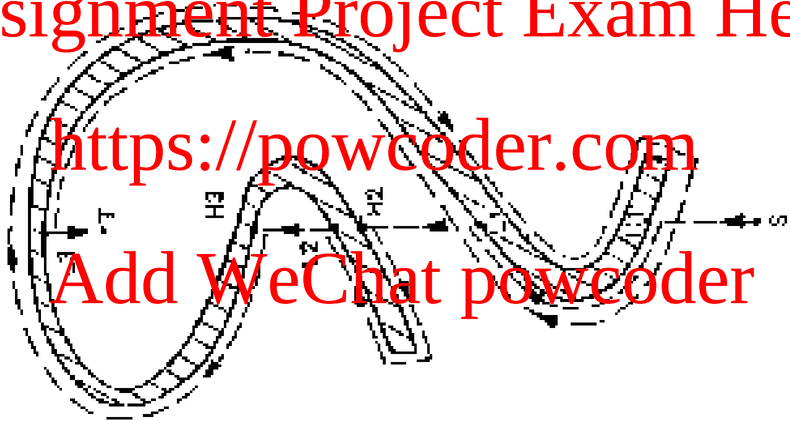
- The second bug algorithm improves on the performance of Bug 1.

Bug 2 Algorithm:

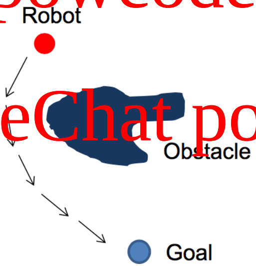
- Follow the obstacle always on the left or right side.
- Leave the obstacle if you cross the direct (line of sight) connection between start and goal.

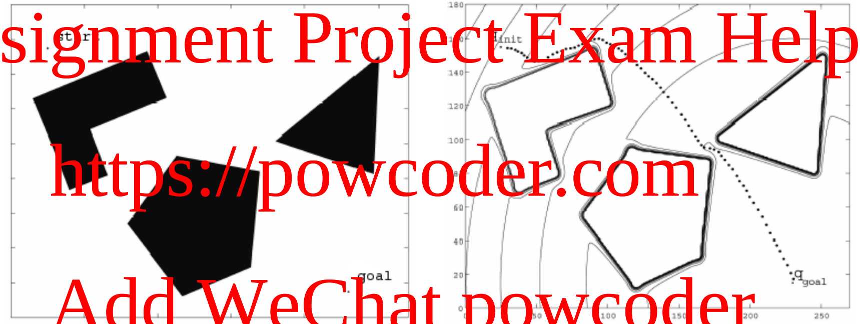

goal
L2
L1
start

- Works even on very complex obstacles



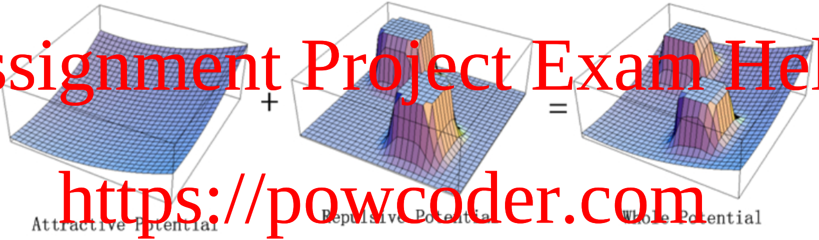
- Robot is treated as a point under the influence of an artificial potential field.
- The goal attracts it and obstacles repel it.
- It's like the goal has a "spring" that draws the robot towards it and away from obstacles that are in its path





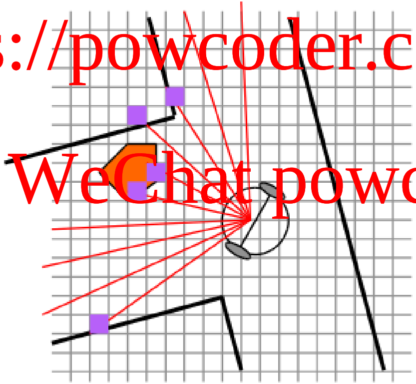
- Generated robot movement is similar to a ball rolling down the hill
- Lots of possibilities to get stuck in local minima.

Potential field, p3



- The idea is that potential energy is stored in the environment.
- The robot wants to minimise its potential energy.
- So it moves down the potential energy gradient.
- Goals "attract" potential energy.
- Obstacles "repel" potential energy.

- Approach that uses sensor readings to tell the robot how to avoid obstacles.
- Representing the area around the robot as a grid, compute the probability that any square has an obstacle.
- Provides a local map to decide how the robot should move.



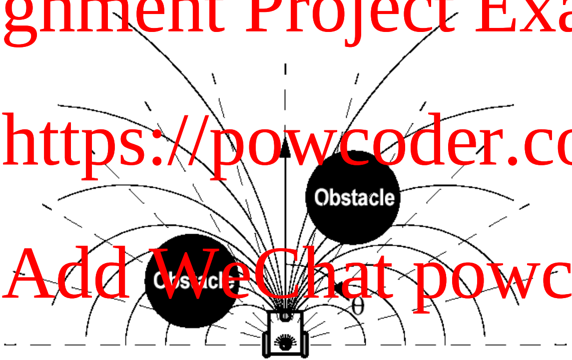
- The local map is reduced to a 1 DOF histogram:

- Then compute the steering angle for the best gap
- Best selected using function G which combines:

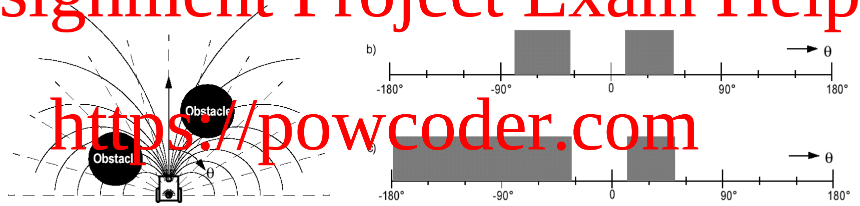
$$G = a.\ \texttt{target-direction} + b.\ \texttt{wheel-orientation} + c.\ \texttt{previous-direction}$$

VFH+

- An issue with VFH is that it does not account for how the robot can really move.



- The best gap could be one that the robot has to stop and do some complex maneuver to go through.

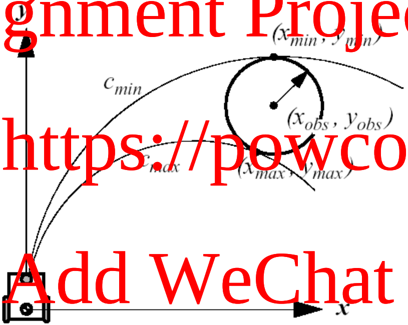


- VFH+ considers motion on trajectories.
- Any turn that has a trajectory that intersects an obstacle is blocked

- VFH in action.

- VFH and VFH+ are limited if narrow areas (e.g., doors) have to be traversed.
- Local minima might not be avoided.
- Reaching the goal can not be guaranteed.
- Dynamics of the robot not really considered.

- Go further than VFH+ in modelling the motion of the robot.



- Transform obstacles into the velocity space of the robot.
- Apply acceleration constraints to determine possible velocities.

- Environment
- State
- Memory
- Knowledge Representation

Robot environments are characterized by various properties:

- accessible vs inaccessible
  - In an accessible environment, a robot has access to all the necessary information required to make an informed decision about its actions.
- deterministic vs nondeterministic
  - In a deterministic environment, any action that a robot undertakes has only one possible outcome.
- episodic vs non-episodic
  - In an episodic environment, activity proceeds as a series of repeated episodes.
- static vs dynamic
  - In a static environment, things change only due to actions effected by the robot.
- discrete vs continuous
  - In a discrete enviroment, sensor readings and actions have a distinct, separable values.

- A robot's state refers to knowledge about itself and its environment.
  - Kinematics is the study of the correspondence between actuator mechanisms and the resulting motion, which can be either:
    - linear; or
    - rotary.
  - Kinematics answers questions like:
    - Did I go as far as I think I went?
      (e.g., the result of linear motion)
    - Did I extend my arm as far as I think i did?
      (e.g., the result of rotary motion)
- A robot's environment is full of information. It is important to determine what is relevant to represent, given the robot's abilities and task.
  - What properties can be sensed?
  - How can the sensed information be stored in a useable and useful way?

Like with humans, a robot's memory is divided into 2 categories according to duration:

- Short Term Memory
- Long Term Memory

- STM is transitory.
- It is used as a buffer to store only recent sensory data.
- It stores data used by only one behaviour.

- For example:
  - avoid-past: avoid recently visited places to encourage exploration of novel areas
  - wall-memory: store past sensor readings to increase correctness of wall detection

- LTM is persistent.
- Metric maps use absolute measurements and coordinate systems.

- Qualitative maps use landmarks and their relative relationships to each other.
- For example:

  - Markov models are graph representations which can be augmented with probabilities for each action associated with each sensed state.

- State — A robot's state can (which comprises knowledge about the robot and its environment) can be totally observable, partially observable or unobservable. States can be discrete or continuous, making it easier or more difficult (respectively) to distinguish one state from another.

- Spatial information — Spatial information is a description of the navigable surroundings in a robot's environment and their structure. Spatial information is typically stored in a metric or topological map.

- Objects — Objects are categories and/or instances of detectable things in the robot's environment.

Components of Knowledge Representation, p2

- Actions — Actions that a robot can perform are part of its knowledge representation. This includes the expected outcomes of specific actions on the robot and on its environment.

- Self/ego — A robot stores proprioception (sensing of its internal state), its own self-limitations and capabilities. These include information about perceptions (how to sense) and behaviours (how to act).

- Intentional — A robot's intentions are comprised of its goals, its plans to achieve those goals and its intended actions that make up the plans.

- Maps
  - Euclidean map — Represents each point in space according to its metric distance to ll other points in the space
  - Topological map — Represents locations and their connections, i.e., how/if they can be reached from one another; but does not contain exact metrics
  - Cognitive map — Represents behaviours; can store both previous experience and use for action. Used by animals that forage and home (animal navigation). May be simple collections of vectors.

Types of representations, p2

- Graphs —
  - Represent the relationship between states (nodes in the graph) and actions (links between nodes). The links indicate how to move from one state to another, based on an action.
  - Can also represent other types of cause/effect or relational information other than state/action pairs.

- Markov models —
  - Associate probabilities with states and actions.
  - In a graph, actions are given probabilities (usually determined experimentally, by observation) indicating how likely it is that an action taken in one state will lead to another state. It is thus possible to have a single action taken in one state to lead to multiple possible subsequent states, each with a different probability.

KING'S College LONDON

Today

Navigation

Cognition

Behaviour-Based Systems

Multi-Robot Systems

- Control models
- Behaviour-based models
- Expressing behaviours
- Behaviour coordination
- Emergent behaviour

There is distinction between:

- Classic model-based control
- Focusses on symbolic representations
- Based on "good old-fashioned AI" – e.g., [McCarthy, 1958]

- Newer behaviour-based control
- Focuses on numeric representations
- Based on "nouvelle AI" – e.g., [Brooks, 1986]

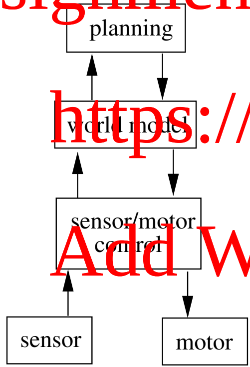
- There are also hybrid models that combine aspects of both model-based and behaviour-based.

- Deliberative models: Sense, Plan, Act
  - Provide a functional decomposition
  - Systems consist of sequential modules achieving independent functions, such as: "Sense world", "Generate plan"

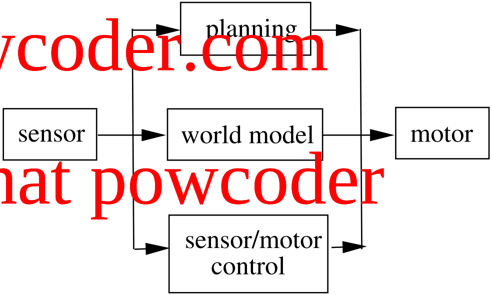
- Reactive models
  - Provide a task-oriented decomposition
  - Systems consist of concurrently executed modules achieving specific tasks, such as: "Avoid obstacle", "Follow wall"

Control models, p3

- Two orthogonal control flows:



vertical



horizontal

- They are characterized by behavioural decomposition, rather than a functional or a task-oriented decomposition.
- Systems consist of sequential modules achieving independent functions.
- They provide a natural fit to robotic behaviour.
  - Behaviour-based models generate a motor response from a given perceptual stimulus.
  - The basis for these systems is in biological studies; and as a result, biology is an inspiration for the design of many behaviour-based models.
  - The field of Artificial Life or ALife focuses on the development of computational models of natural phenomena, including behaviour of individual and groups of animal(s).

- A behaviour is anything observable that the system/robot does.
- How do we distinguish internal behaviours (components of a behaviour-based system) and externally observable behaviours?
- Reactive robots display desired external behaviours. For example: “Avoiding”; “Collecting cans”; “Walking”.
- But a controller consists of a collection of rules, possibly organized in layers (e.g., subsumption architecture).
- Behaviour-based models actually consist of and are programmed in behaviours, which have higher granularity than rules, extended in time (as opposed to rules, which are typically short-term), and capable of using and maintaining sophisticated knowledge representations.

- It is often hard to distinguish between a behaviour-based system and a classical, action-based system.

- We make a comparison here.

A behaviour . . .

- is based on dynamic processes
  - that operate in parallel,
  - that lack of central control, and
  - that provide fast couplings between sensors and motors.
- exploits emergence
  - which are side-effects from combining processes, and
  - often use properties of the environment
- tends to be reactive.

An action . . .

- is discrete in time,
  - with well defined start and end points, and
  - allows for pre- and post-conditions.

- avoids side-effects
  - because it is a small, atomic description of one (or a few related) steps, and
  - because it also avoids conflicts.

- tends to be deliberative.

- Actions are building blocks for behaviours.

Distinguishing behaviour-based systems

Characteristics:

- Achieve specific tasks/goals (e.g., "Avoid Others", "Find Friend")
- Typically execute concurrently
- Can store state and be used to construct world models
- Can directly connect sensors to effectors
- Can take inputs from other behaviours and send outputs to other behaviours (e.g., can be connected in behaviour networks)
- Typically higher-level than actions (e.g., a behaviour might be "Go Home", whereas an action would be "Turn left 45 degrees")
- Typically closed loop, but extended in time

Key Properties:

- Ability to act in real time
- Ability to use representations to generate efficient behaviour
- Ability to use a uniform structure and representation throughout

Distinguishing behaviour-based systems

Challenges:

- How can a representation be effectively distributed over the behaviour structure? The time scale must be similar to that of the real-time components of the system. The representation must use the same underlying behaviour structure for all system components.
- Some components may be reactive.
- Not every component is involved with representational computation.
- Some systems use a simple representation.
- As long as the basis is in behaviours and not rules, then the system is a behaviour-based system.

Expressing behaviours

- Behaviours can be expressed using various representations.
- When a control system is being designed, the task is broken down into desired external behaviours.

- Behaviours can be expressed using a variety of models
  - Functional notation
  - FSA (Finite State Automata) diagrams
  - Subsumption Architecture
  - Stimulus Response (SR) formalism

- Strengths and weaknesses of various behavioural encodings:

Strengths:

- Support for parallelism
- Run-time flexibility
- Timeliness for development
- Support for modularity

Weaknesses:

- Niche targetability
- Hardware retargetability
- Combination pitfalls (local minima, oscillations)

- Mathematical model:
  - represented as triples (S, R, β)

    S = stimulus

    R = range of response

    β = behavioural mapping between S and R
- Easy to convert to functional languages like LISP
- For example:

```
coordinate-behaviours [
   move-to-classroom ( detect-classroom-location ),
   avoid-objects ( detect-objects ),
   dodge-students ( detect-students ),
   stay-to-right-on-path ( detect-path ),
   defer-to-elders ( detect-elders )
] = motor-response
```

FSA diagrams

- Finite State Automata can be used to show sequences of behaviour transitions, where states represent behaviours.
- Situated Automata are a formalism for specifying FSAs that are situated in a particular environment

- Tasks are described in high-level logic expressions, as a set of goals and a set of operators that achieve (`ach`) and maintain (`maint`) the goals.
  - `(ach in-classroom)`
  - `(maint avoid-objects)`

- Once defined, tasks can be compiled into circuits, which are reactive.

- For example:

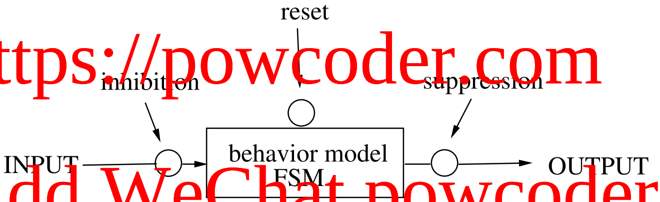
```
(defgoalr (ach in-classroom)
  (if (not start-up)
    (maint (and (maint move-to-classroom)
                (maint avoid-objects)
                (maint dodge-students)
                (maint stay-to-right-on-path)
                (maint defer-to-elders)))))
```

- Subsumption Architecture [Brooks, 1986]

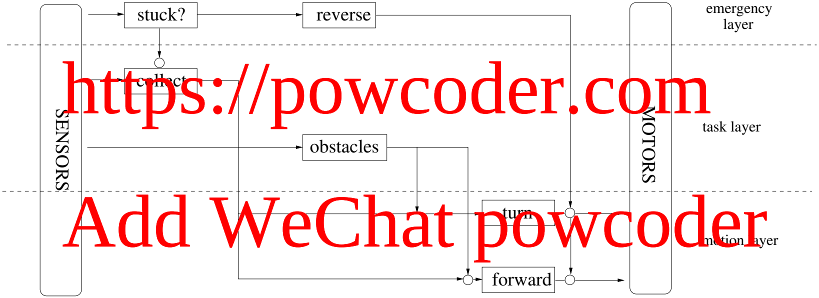
Components:

- Reactive elements
- Behaviour-based elements
- Layered approach based on levels of competence
- Uses an augmented Finite State Machine (FSM) for each behaviour

- Each comoponent:



- Combines FSMs in a sort of layered, circuit diagram:



Stimulus-Response formalism

- The Stimulus-Response formalism is based on the premise that a behavioural response in physical space has a strength and an orientation.
- Expressed formally as (S, R, β), where:
  - S = stimulus, necessary but not sufficient condition to evoke a response; internal state can also be used as a stimulus
  - R = response
  - β = behavioural mapping categories, which can be: null, or discrete, or continuous

- Mapping can either be:
  - Discrete
  - Continuous

Stimulus-Response formalism, p2

Discrete Mapping

- Expressed as a finite set of situation-response pairs/mappings
- Mappings often include rule-based (IF-THEN) formulae.
- Examples:
  - Subsumption language

Continuous Mapping

- Instead of discretizing the input and output, a continuous mathematical function describes the input-output mapping.
- Can be simple, time-varying, harmonic.
- Examples:
  - Potential fields
- However, here are problems with local minima, maxima, oscillatory behaviour.

- Behaviour-based systems consist of collections of behaviours.
- Execution must be coordinated in a consistent fashion.
- Coordination amongst behaviours can be competitive, cooperative, or a combination of the two.

Competitive coordination:

- Perform arbitration: selecting one behaviour among a set of candidates
  - priority-based: subsumption
  - state-based: discrete event systems
  - function-based: spreading of activation action selection

Cooperative coordination:

- Perform command fusion: combining outputs of multiple behaviours
  - voting
  - fuzzy (formalized voting)
  - superposition (linear combinations): potential fields, motor schemas, dynamical systems

- The problem of deciding what to do next is considered either:
  - an Action-Selection Problem, or
  - a behaviour-Arbitration Problem.

- **Emergence** is an important but not well-understood phenomenon. It is often found in behaviour-based robotics.

- Robot behaviours "emerge" from interactions between rules, behaviours, and/or with the environment.
  Coded behaviour is in the programming scheme.
  Observed behaviour is in the eyes of the observer—it emerges!
  There is no one-to-one mapping between the two!

- The notion of emergence depends on two aspects:
  - The existence of an external observer, to observe and describe the behaviour of the system

  - Access to the internals of the controller itself, to verify that the behaviour is not explicitly specified anywhere in the system

- Unexpected vs Emergent
  - Some researchers say the above is not enough for behaviour to be emergent, because above is programmed into the system and the "emergence" is a matter of semantics.

  - So emergence must imply something unexpected, something "surreptitiously discovered" by observing the system. But "unexpected" is highly subjective; it depends on what the observer was expecting. Naïve observers are often surprised. Informed observers are rarely surprised.

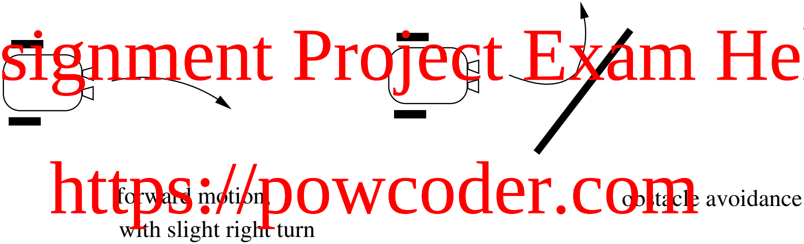
  - Once a behaviour is observed, it is no longer unexpected. Is new behaviour then "predictable"?

Wall following behaviour can be implemented with these rules:

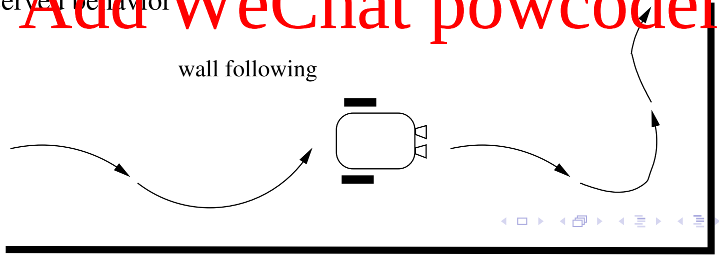
- if too far, move closer
- if too close, move away
- otherwise, keep on going

- Over time, in an environment with walls, this will result in wall-following.

coded behavior



observed behavior



- Is this emergent behaviour?
- It is argued yes because:
  - The robot itself is not aware of a wall, it only reacts to distance readings.
  - The concepts of "wall" and "following" are not stored in the robot's controller.
  - The system is just a collection of simple rules.

Today

Navigation

Cognition

Behaviour-Based Systems

Multi-Robot Systems

- Techniques
- Surveillance Applications
- Localisation
- Planning
- Coordination

- Techniques include:
  - Learning of controllers for teams of robots
  - Swarm-based robotics, which use large numbers of small robots and employ biologically inspired techniques

- Techniques have been developed to ensure that
  - the entire boundary of a space is visited
  - search finds a specific target
  - a human-robot team can exchange search roles flexibly and appropriately

- Foraging: robots systematically sweep an area, search for and gather objects of interest

- Exploration: robots investigate an area and build a shared "map" of the area/resources/features

- Surveillance: robots observe an area for changes over time
- Formation approaches:
  - applying control theory
  - stabilizing formations of multiple agents
  - moving in formation where the environment is not conducive to maintaining formation

- Strategies:
  - Formation: structuring relative positions of multiple robots
  - Caging or Object Closure:
    - instead of single-robot idea of caging, where a multi-fingered hand surrounds an object, multi-robot caging implies a group of robot surrounding or constraining an object restricting an object to a specific set of configurations using multiple robots; uses local information with minimal communication and sensing and is based around vector fields
    - comparing experimental data gathered on real robots with only (omnidirectional) cameras to ground truth supplied by an overhead camera
  - Coverage:
    - surveilling pursuer, while continuing to keep an eye on an evader examining every point on the boundary of a 2D space
    - swarm-based approaches

- Challenges:
  - Maintaining communication (wifi/radio connectivity) during exploration
  - Information fusion fusing data from multiple robots to give position estimates of objects of interest
  - finding an object with multiple robots
  - cooperative robot-human searching of an environment in which there is an "adversarial target" that must be cleared
  - Task/region allocation
  - deciding how to search a space with a group of searchers that will find an evading target to result in "guaranteed search"
  - analyzing range of auction bidding rules for auctions in which robots bid for targets; robots have to visit all targets that they "win", and the team is judged by combinations of the distances travelled by all robots visiting their targets (e.g., traveling salesman problem)
  - implementing foraging using robot swarms

- Multiple robots jointly estimate each other's positions
- A group of robots can localize faster (than they can on their own) if they share information.
- Approaches:
  - adding a detection model to the usual particle-filter, sensor model and motion model; when one robot is spotted by another, it updates its location with information that takes into account the belief the other robot has on its position
  - showing that it is possible to use wireless ethernet—in particular the strength of wireless ethernet signals—as the basis for localization
  - planning the trajectories of multiple robots in order to improve their localization performance

- Multi-robot Simultaneous Localization And Mapping (SLAM)
  - additional information from several robots can speed up single-robot SLAM
  - though multiple robots can also complicate the situation

- Approaches:
  - developing vision-based multi-robot SLAM
  - applying particle filters to multi-robot SLAM
  - providing a strategy for accurate control of a group of robots (i.e., keeping on the desired path while localizing and mapping)
  - considering performance bounds

- Aspects of planning in multi-robot systems:
  - path and motion planning
  - task planning
- Challenges in multi-robot planning:
  - dynamic environments, robots with multiple goals
  - motion planning for multiple robots
  - maintaining radio/wifi communication layered on top of multi-robot route-planning
  - task planning for multi-robot systems under time constraints
- Strategies for multi-robot path planning:
  - multi-robot path planning using probabilistic roadmaps
  - planning in terms of roles, allowing robots to change roles through "exchange" as situations arise
  - shared memory task scheduling for a heterogenous multi-robot team making use of a "shared global unit" to reduce communication overhead

Multi-robot **Coordination**

- Standard tasks:
  - Traffic control
  - Box-pushing
  - Foraging
- Approaches:
  - Merging the plans of multiple robots; this type of coordination is an intersection of several robots' plans—they aren't working together, but they are in the same space at the same time
  - Using parallel stochastic hill-climbing for coordinating a small team of robots in a pursuer-evader task
  - Dynamically coordinating robots using task assignment and integrates data on obstacles
  - Coordinating movement of objects by homogeneous teams of robots

Task Allocation: dividing responsibilities in a group of individuals

- Hard problem because:
  - robot team requirements change over time
  - abilities of individual robots are conditioned on changing locations
  - different robots have different capabilities

Multi-Robot Systems: Task Allocation, p2

- Approaches:
  - providing a local approach to task allocation in a multi-robot team; robots only use information from immediate neighbors
  - performing concurrent mapping and localization and showing improvements with multi-robot data collection
  - formalizing multi-robot task allocation and analyzes the computational and communication requirements of some approaches
  - performing dynamic task allocation when the coordination is distributed
  - learning using Alliance which aims to handle heterogeneous teams of robots in a variety of scenarios
  - learning specialities of robots using reinforcement learning and using a blackboard to distribute knowledge sharing
  - Contrasting task allocation in robot swarms using random assignment with more measured approaches that use different amounts of bandwidth and time to run

- Many apply market-based approaches to multi-robot coordination and task allocation, such as **auctions**
- Example: organizing robots for exploration tasks
- Areas to explore are offered "for sale"; robots bid based on distance to locations on offer; allocation favors lower bids; tends to allocate areas closer to robots; market constructed to ensure that robots are not idle when several robots are close to the same unexplored area.
    - tasks are offered "for sale" to robot team members
    - robots indicate how much they are willing to "pay" for tasks
    - tasks are allocated based on bids for them

- Approaches:
  - Murdoch, auction-based method for coordination
  - Hoplites, uses market-based techniques to provide tight coordination of multi-robot teams
  - using auction-based coordination mechanisms to orchestrate movement of robots to certain points within time windows
  - focusing on bid evaluation in market-based task allocation mechanism

References

- Some content from:

  Introduction to Autonomous Mobile Robots, by Roland Siegwart, Illah R. Nourbakhsh and Davide Scaramuzza (2011), MIT Press.

  Robotic Motion Planning, by Howie Choset, http://www.cs.cmu.edu/~motionplanning/

- Many slides thanks to Prof Simon Parsons.