Lecture 7-1

Think and Act Separately, in Parallel:
Hybrid Control
Think the Way You Act:
Behavior-Based Control
Making your robot behave:
Behavior Coordination:
When the unexpected happens:
Emergent Behavior

The Robotics Primer (Ch. 15 - 18)

C.A. Berry

Course Announcements

- Quiz on **Tuesday, 4/28/09** on **Hybrid and Behavior-Based Control, Behavior Coordination and Emergent Behavior**
- Lab 5 Demo due **Thursday, 4/30/09**
- Lab 5 Memo and code due by midnight on **Friday, 5/01/09**
- Upload memo and code to Angel
- Bring your laptop and robot everyday
- **DO NOT** unplug the network cables from the desktop computers or the walls

Quote of the Week

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

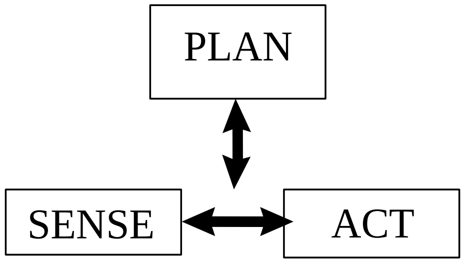
R. Fowler, Refactoring: Improving the Design of Existing Code

Hybrid Control (Ch. 15)

Hybrid Control

- Recall that reactive control is fast but inflexible
- Deliberative control is smart but slow
- Hybrid control exploits the best features of reactive and deliberative control
- *Hybrid control* involves the combination of reactive and deliberative control within a single robot control system
- Therefore, short and long time scales must work together
- No representatives and explicit and elaborate world models must be made to work together effectively

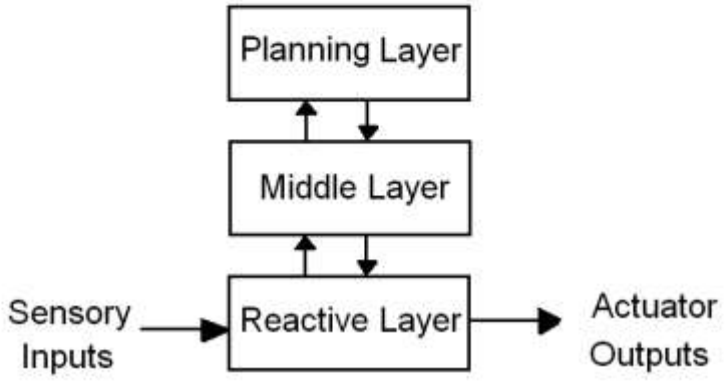
Three-Layer Architecture



- A hybrid system has three components:
  - A reactive layer
  - A planner
  - A layer that links the above two together
- The middle layer has to
  - Compensate for limitations of both the planner and the reactive system
  - Reconcile the two different time scales
  - Deal with their different representations
  - Reconcile any contradictory commands they may send to the robot

Three-layer architecture

- When there is a change in the world, map or task the middle layer informs the deliberative layer to update its representation of the world to generate more accurate and useful plans
- *Dynamic replanning* occurs when the reactive layer discovers it cannot proceed. The planner can tell the robot to stop moving through the reactive layer
- One solution is to have some stored plans based upon common initial and goal states. They are not reactive because they have multiple steps and not plans so they reside in the middle layer

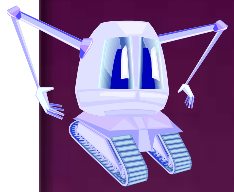


On-line and Off-line Planning

- *Off-line planning* is done to plan for all of the situations that might come up
- This takes place while the robot is being developed
- *On-line planning* takes place while the robot is busy trying to get a job done and achieve goals
- A *universal plan* is a set of all possible plans for all initial states and all goals within the state space. This is a reactive robot because the planning is done off-line not at run time

Function of Time

- Reactive Layer
  - Exists in the ***PRESENT*** or real time response to sensory input
- Deliberative Layer
  - Reasons about the ***PAST*** to create plans
  - Makes projections into the ***FUTURE*** for planning

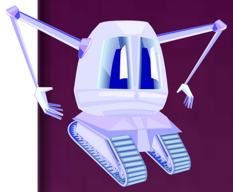


Domain Knowledge

- Precompiled existing optimal plans can be put into the system in a clean, principled way.
- Information about the robot, the task and the environment is called *domain knowledge*
- It is compiled in a reactive controller so that it does not have to be planned on line in real time
- These plans becomes reactive rules to be looked up
- Drawback to situated automata
  - State space is too large for most realistic problems
  - The world must not change
  - The goals must not change

Drawbacks to Hybrid Control

- The middle layer is hard to design and implement, very task and robot specific
- A hybrid system can degenerate into having the planner slow down the reactive system so the reactive system may ignore the planner
- An effective hybrid system is difficult to design and/or debug

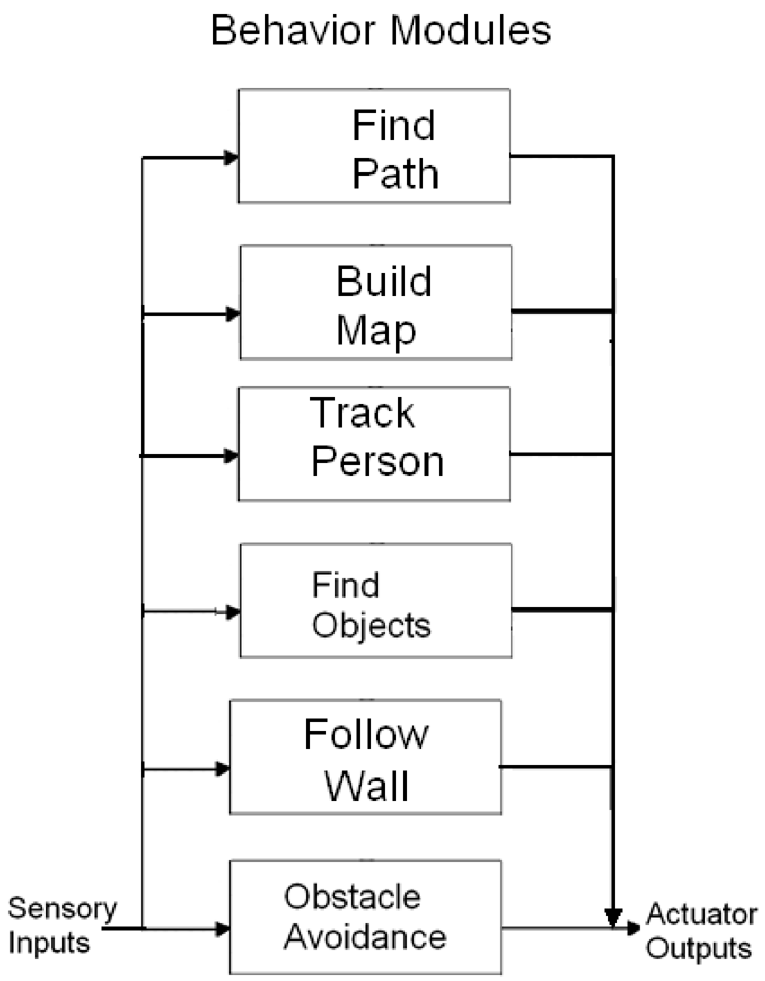


Behavior-based Control (Ch. 16)

Behavior-Based Control

- *Behavior-based control (BBC)* incorporates the best of reactive systems but does not involve a hybrid solution
- BBC is inspired by biological systems
- BBC was inspired by the fact that
  - Reactive systems are too inflexible, incapable of representation, adaptation, or learning
  - Deliberative systems are too slow and cumbersome
  - Hybrid systems require complex means of interaction among the components
- BBC is closest to reactive control and farthest from deliberative control

Behavior-Based Architecture



Behavior definition

- Behaviors achieve and /or maintain particular goals (i.e. homing, wall following)
- Behaviors are time extended, not instantaneous. They may take some time to achieve and/or maintain their goals
- Behaviors can take input from sensors and also other behaviors. Behaviors can send outputs to effectors and to other behaviors
- Behaviors are more complex than actions. A reactive system may use simple actions like *stop*, *turn right*.
- A BBC may have behaviors such as *find object, follow wall, get recharged, hide from the light*.

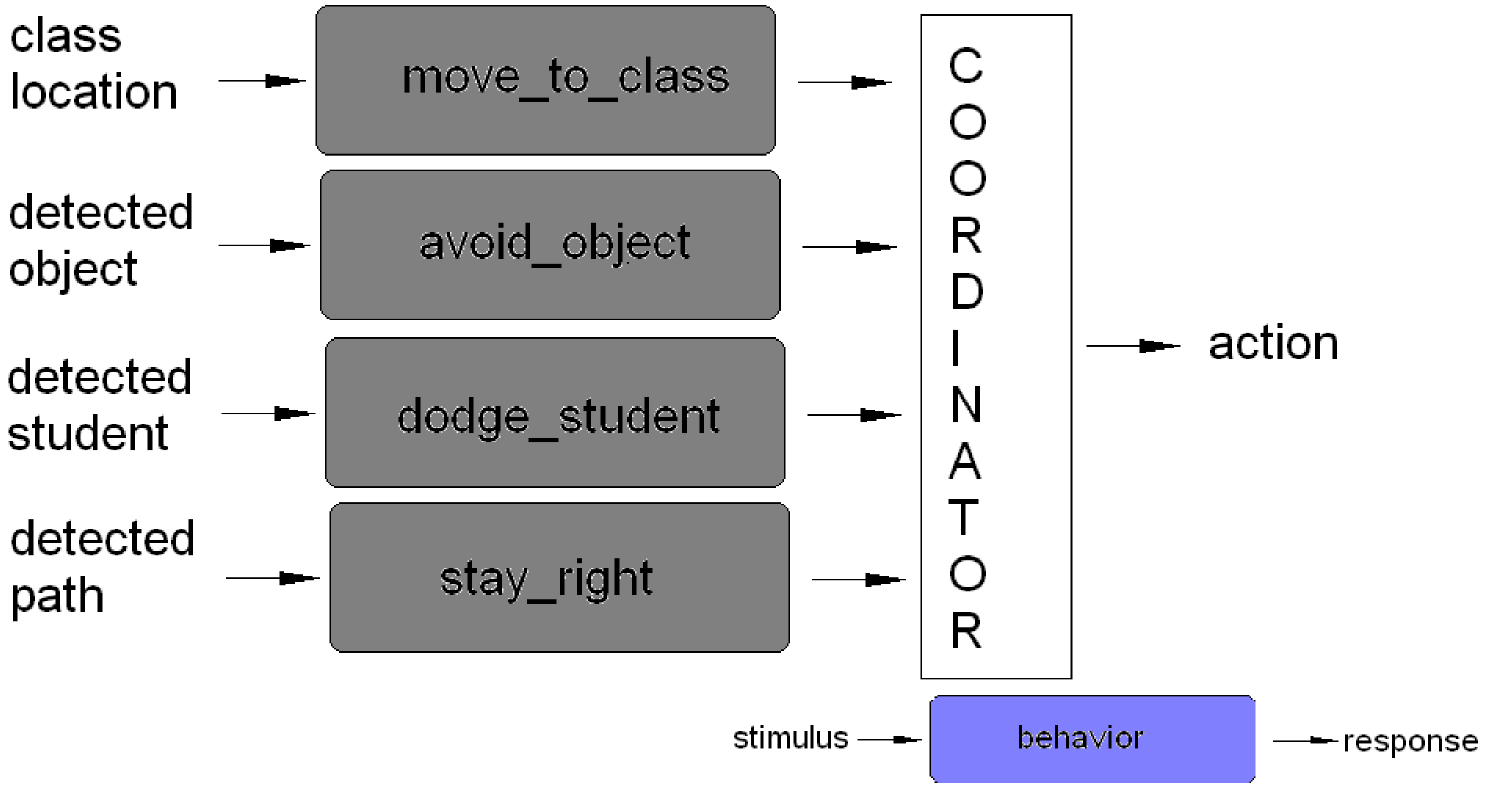
Abstraction

- Behaviors can be designed at a variety of levels of detail or description. This is called *their level of abstraction.*
- To be abstract is to take details away and make things less specific
- Behaviors can take different amounts of time and computation
- Behaviors are flexible and this is one of the key advantages of BBC
- The power and benefit is also in the way they are organized and put together in the control system

Expressing robotic behavior

- There are several methods available for expressing robotic behavior including:
  - Stimulus-response (SR) diagrams
    - Use for graphic representation of specific behavioral configurations
  - Functional Notation
    - Used for clarity in design of systems
  - Finite state acceptor (FSA) diagrams
    - Used for temporal sequencing of behaviors

SR diagram for classroom navigation robot



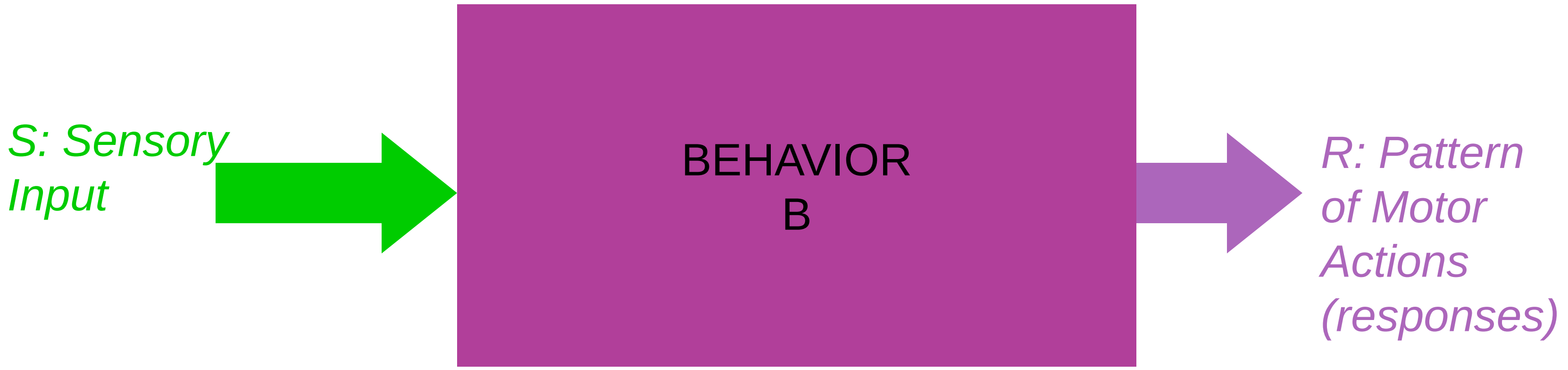
Functional notation for classroom navigation robot

b(s) = r

The behavior b when given stimulus s yields response r

```
Coordinate-behaviors
    [
    move_to_class(detect-class-location),
    avoid_objects(detect-objects),
    dodge_students (detect-students),
    stay_to_right(detect-path)
    Return motor_response
    ]
```

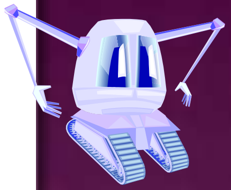
Behavior Definition



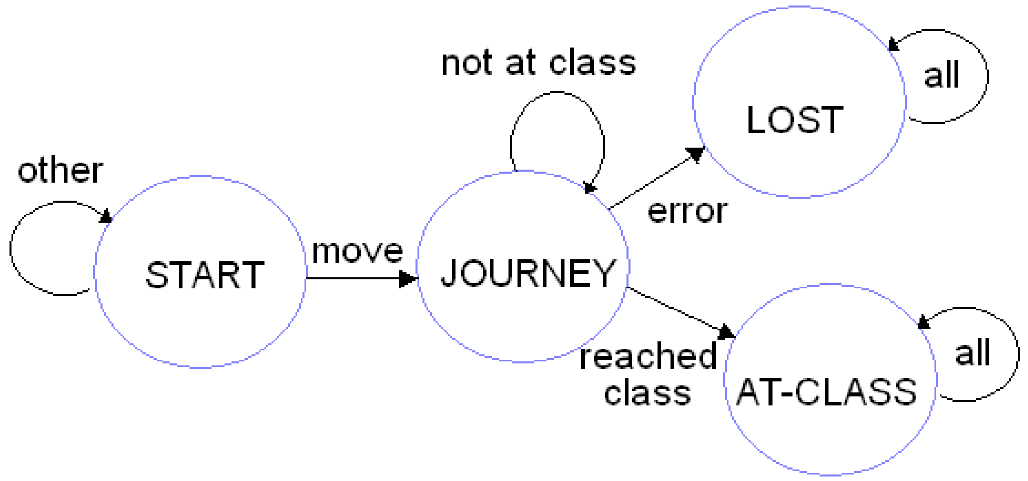
A behavior is a mapping of sensory inputs to a pattern of motor actions which are then used to achieve a task.

Notation: B(S)=R

Finite State Acceptor (FSA) Diagrams for classroom navigation robot



- FSA diagrams describe the aggregations and sequences of behaviors
- It explicitly shows the active behavior at any given time and the transitions between them
- the circle denotes the state where behavior is active and the arrows represent the stimulus input and response
- Note that Journey includes move_to_class, avoid_objects, dodge_students and stay_right



BBC design principles

- Behaviors are typically executed in parallel/concurrently
- Networks or behaviors are use to store state and construct world models/representations
- Behaviors are designed to operate on compatible time scales
- BBC have the following key properties
  - The ability to react in real time
  - The ability to use representations to generate efficient behavior
  - The ability to use uniform structure and representation through the system

Robomow Behaviors

- Random Wander
- Avoid (bump = obstacle)
- Avoid (wire = boundary)
- Stop (tilt = on)
- All active working concurrently

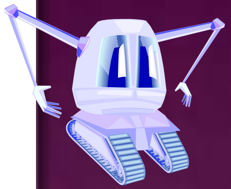
www.friendlymachines.com

Interaction Dynamics and Emergent Behavior

- One of the clever philosophies of behavior-based systems is the effects of the behaviors interacting in the environment rather than internally through the system and this is *interaction dynamics*
- The behaviors that emerge from the interaction of rules and behavior to produce more complex outputs is called *emergent behavior*
- *Behavior-based* controllers are networks of internal behaviors which interact in order to produce external, observable manifested robot behavior

Time and Modularity and Representation

- BBC is modular by using a collection of behaviors
- These behaviors are relatively similar in terms of execution time
- It is based on reactive philosophy where behaviors are incrementally added to the system
- The behaviors are executed concurrently in parallel
- Behaviors are activated in respond to internal and/or external conditions
- Behaviors are more expressive than simple reactive rules
- Behaviors are more complex and more flexible than reactive rules and can be used in clever ways to program robots



Representation

- Behaviors can interact with each other within the robot to store representation
- Behaviors can serve as a basis for learning and prediction which means they can achieve the same things as hybrid systems in a different way. The difference is how representation is used

Distributed Representation

- In BBC, information used in internal representation can not be centralized or centrally manipulated
- The representation (world model) is distributed over the behavior structure
- The representation must be able to act on a similar time scale to the real time components of the system
- The representation must use the same underlying behavior structure as the rest of the system

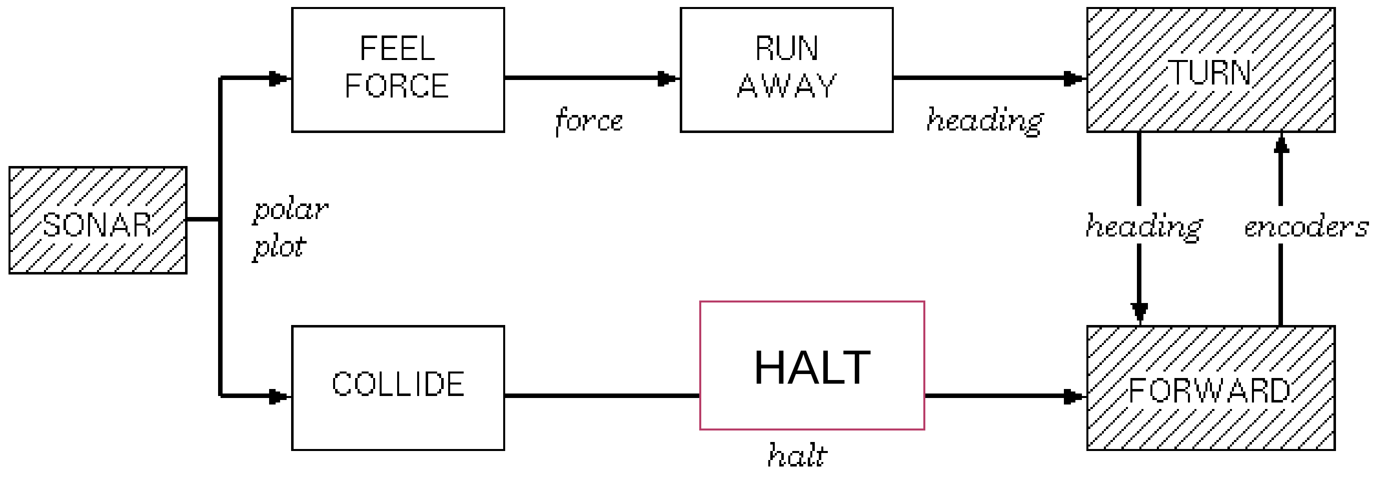
Distributed Mapping Behavior (Toto)

- Toto used behaviors to detect landmarks such as walls and corridors
- By keeping track of the compass readings and length of the walls and corridors Toto could create a map
- This map could be used for localization and to map received landmarks to certain behaviors
- The landmark activates the specific behavior and inhibits other robot behaviors
- The robot would also create a new landmark and behavior and connecting it to previous landmarks when it was discovered

Path Planning in Distributed Mapping

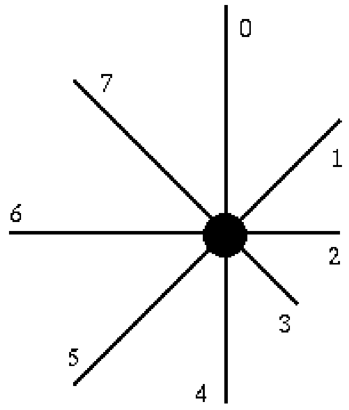
- If Toto had to reach a certain goal then each landmark and its neighbor would send a signal and landmark length as the robot moves toward the goal
- Toto's map behaviors send messages continuously so it could respond to the shortest path received
- This would help with the kidnapped robot problem where someone picks up the robot and moves it to another location
- Toto did not store a map but used the active behavior map to find its way no matter where it was

Level 0: Run Away

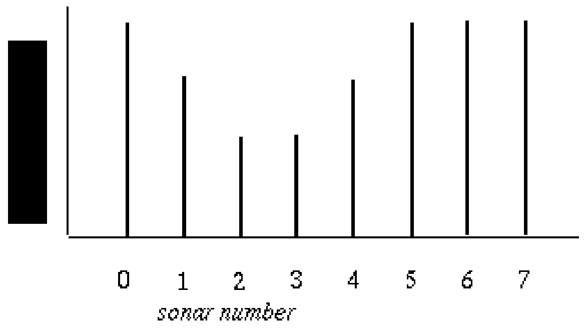


Perception Polar Plot

- The perception plot is egocentric to eliminate a need for memory representation
- Sonar 2 and 3 indicate that there is a wall and no need for memory or reasoning

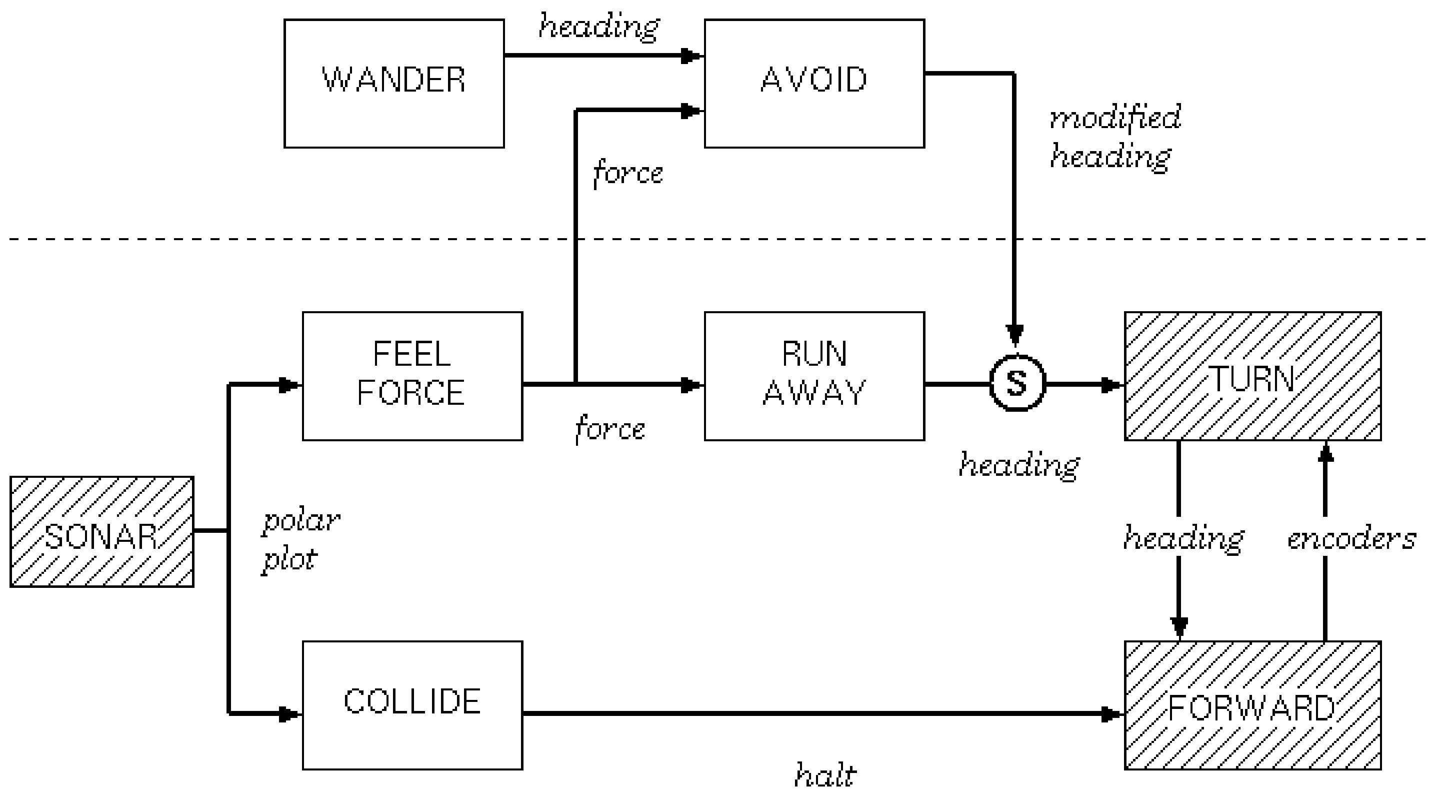


a.

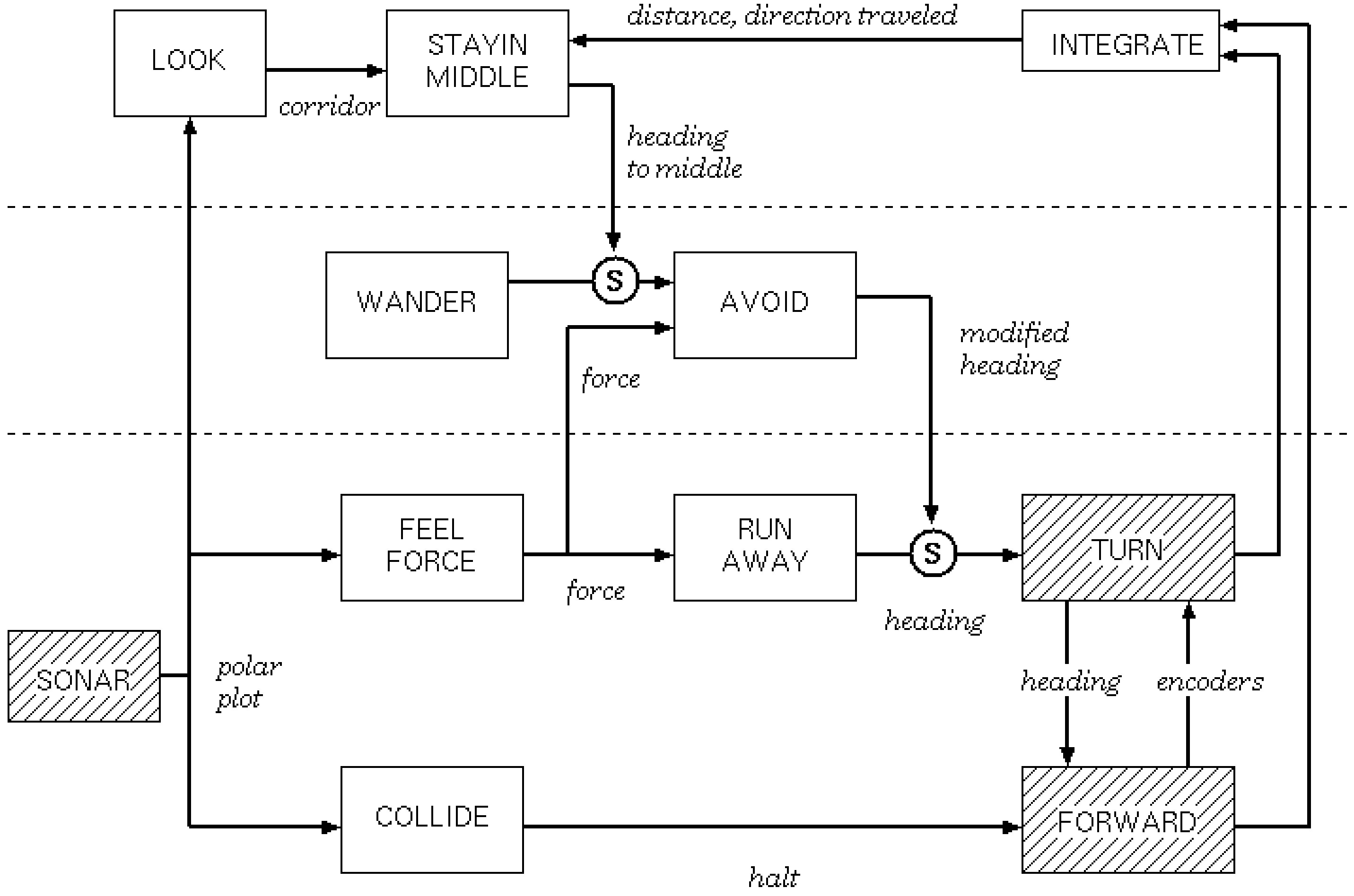


b.

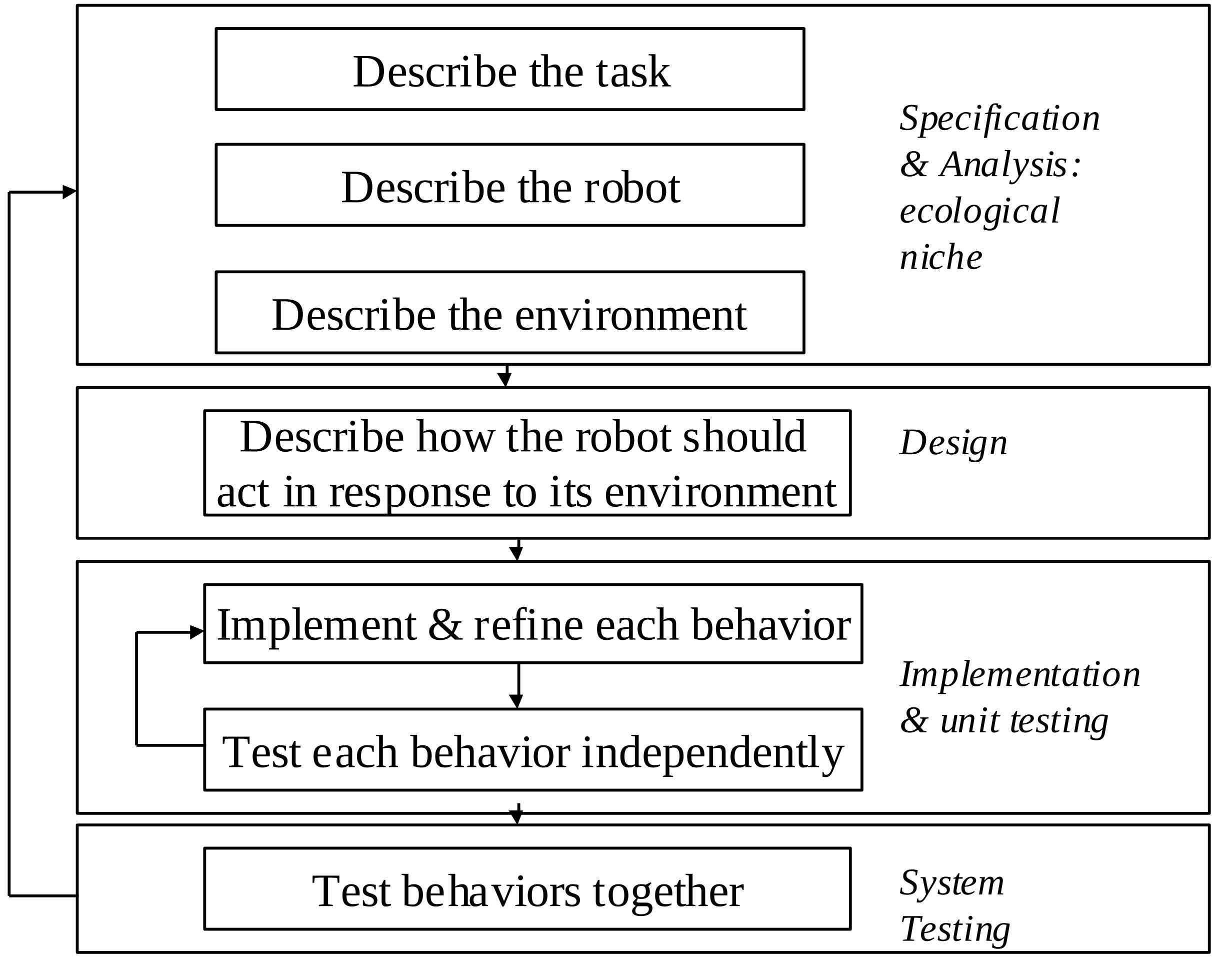
Level 1: Wander



Level 2: Follow Corridor

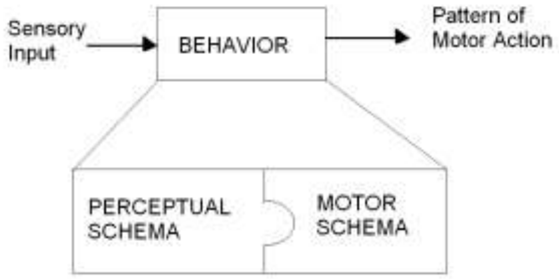


Designing a Behavioral System


Describe the task
Describe the robot
Describe the environment
Specification & Analysis: ecological niche
Describe how the robot should act in response to its environment
Design
Implement & refine each behavior
Test each behavior independently
Implementation & unit testing
Test behaviors together
System Testing

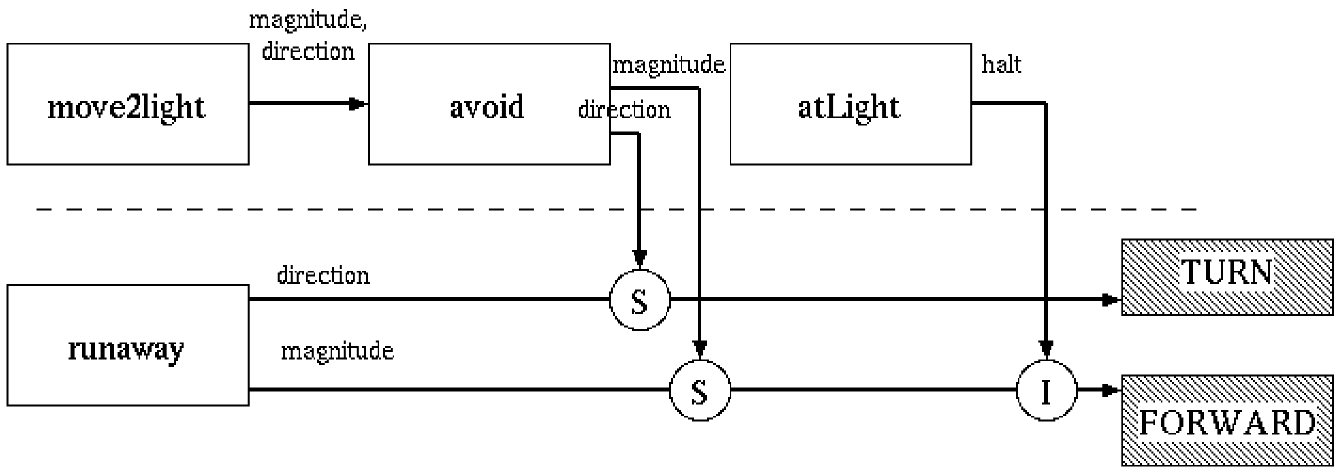
Behaviors as objects in object-oriented programming

- *Objects* consist of data and methods also called attributes and operations
- A *schema* consists both of the knowledge of how to act and/or perceive (knowledge, data structures, models) as well as the computational process by which it accomplishes the activity
- A behavior is a schema that consists of a *perceptual schema* and *motor schema*



Photophilic Behavior Design

Design a robot that is attracted to light. It will head toward the light and if it encounters an obstacle, turn left or right in the direction to favor the light. If there is not light the robot will sit and wait. If an obstacle appears the robot will turn and run away



notice: unlike examples in book, robot doesn't have to go a fixed speed

Photophilic: Behavior Table

Design a robot that is attracted to light. It will head toward the light and if it encounters an obstacle, turn left or right in the direction to favor the light. If there is not light the robot will sit and wait. If an obstacle appears the robot will turn and run away

Releaser	Behavior	Motor Schema	Percept	Perceptual Schema
Light	Photophilic	move2Light()	light: direction & strength	brightness(dir), atLight()

Range	Obstacle avoidance	avoid()
		runaway()
			proximity	obstacle()

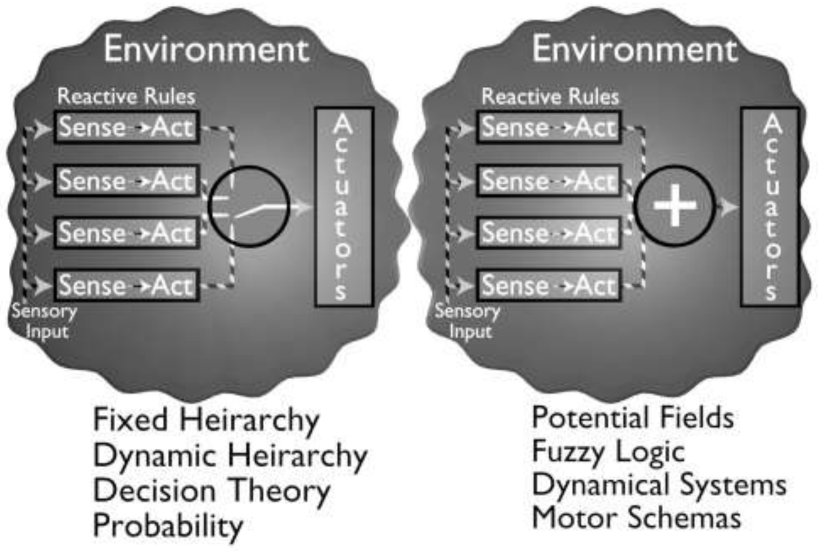
Behavior Coordination (Ch. 17)

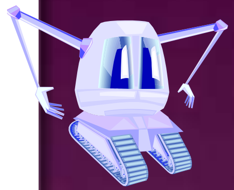
Action Selection or Behavior Coordination

- The problem with *action selection* or *behavior coordination* is what action or behavior should be executed next.
- This problem must be resolved such that the robot always does the right thing over time in order ot achieve its goals
- Hybrid and Behavior-Based systems must resolve the coordination problem
- In Hybrid systems, the middle layer handles coordination
- In behavior-based systems, the coordination may be distributed throughout the control system

Behavior Arbitration versus Fusion

- *Behavior Arbitration* is picking one behavior /action
- *Behavior Fusion* is combining multiple behaviors /actions



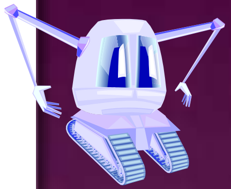


Behavior Arbitration

- Arbitration-based behavior coordination is also called *competitive behavior* because candidate behaviors compete but only one can win
- In a *fixed priority hierarchy*, the behaviors have preassigned priorities
- In a *dynamic hierarchy*, the behavior priorities change at run-time
- *Subsumption architecture* uses a fixed priority hierarchy of behaviors. Some hybrid systems also employ fixed priority hierarchy

Dynamic Hierarchies

- More sophisticated hybrid systems use dynamic hierarchies
- They control the system transition between the reactive and deliberative parts of the controller
- Behavior-based control systems also use dynamic hierarchies to decide which behavior wins and take control of the robot next
- Dynamic arbitration may compute some value to determine who wins such as voting, fuzzy logic, probability, or spreading of activitation



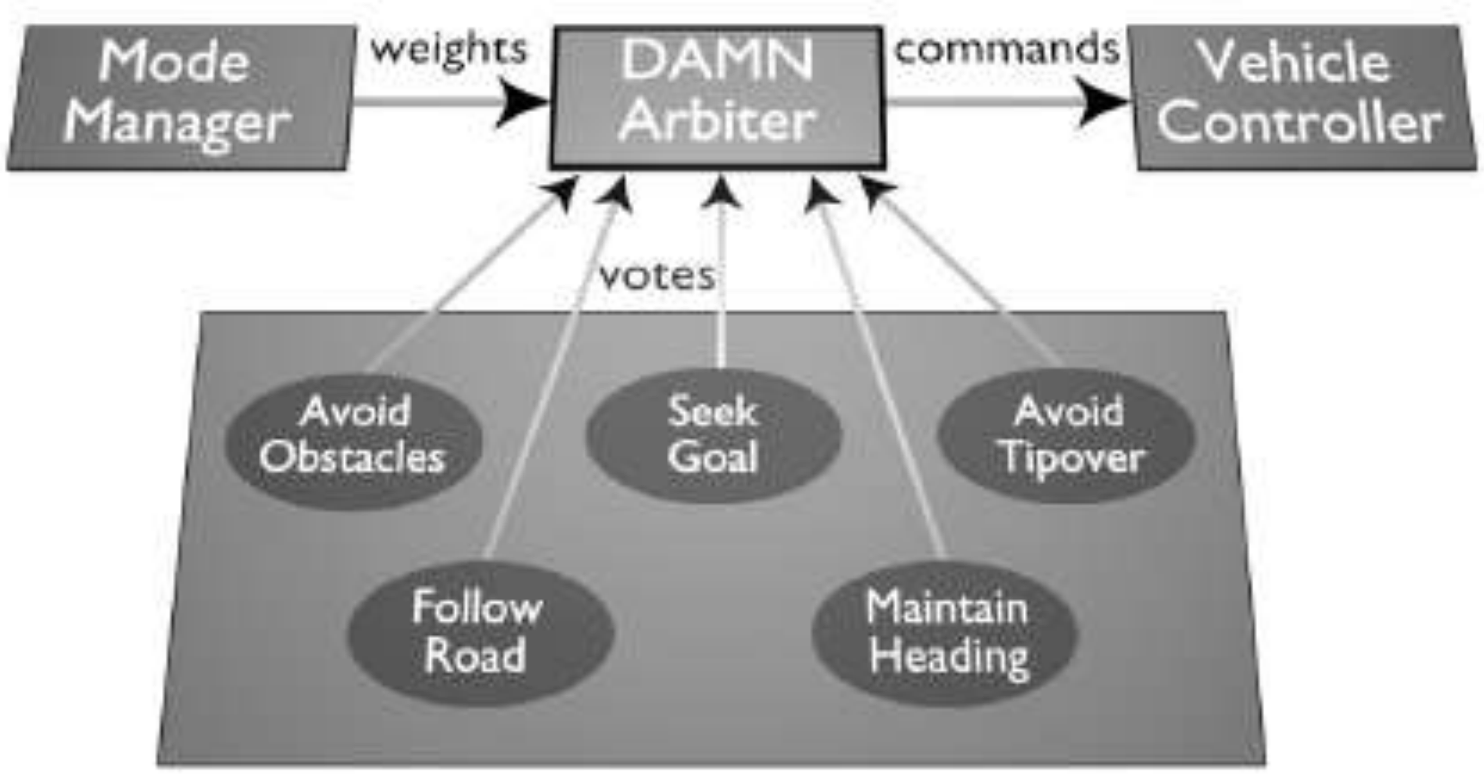
Behavior Fusion

- *Behavior fusion* is the process of combining multiple possible candidates into a single output action/behavior
- *Behavior fusion* is also called a cooperative method because it combines outputs of multiple behaviors to produce a final result
- This result may be an existing behavior or a new one (*emergent behavior*)
- There could be weighting or have some logic in the system to prevent certain combinations and outcomes

Dynamic Autonomous Mobile Navigation (DAMN)

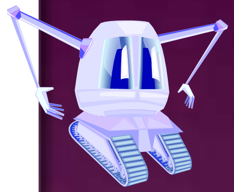
- *DAMN* was used to control a van that autonomously drove on roads over vary large distances
- In *DAMN*, several low level actions are voting members and the result is a weighted sum of actions.
- This was an example of command fusion and was later improved with fuzzy logic
- Formal methods for command fusion include *potential fields*, *motor schema* and *dynamical systems*

DAMN Architecture



Arbitration Selection

- *Fusion methods* are applied to lower-level representations of actions or behaviors (direction, velocity)
- *Arbitration methods* are used for higher-level ones such as (turn, stop, grasp)
- Many control systems are a combination of methods with fusion at one level and arbitration at another



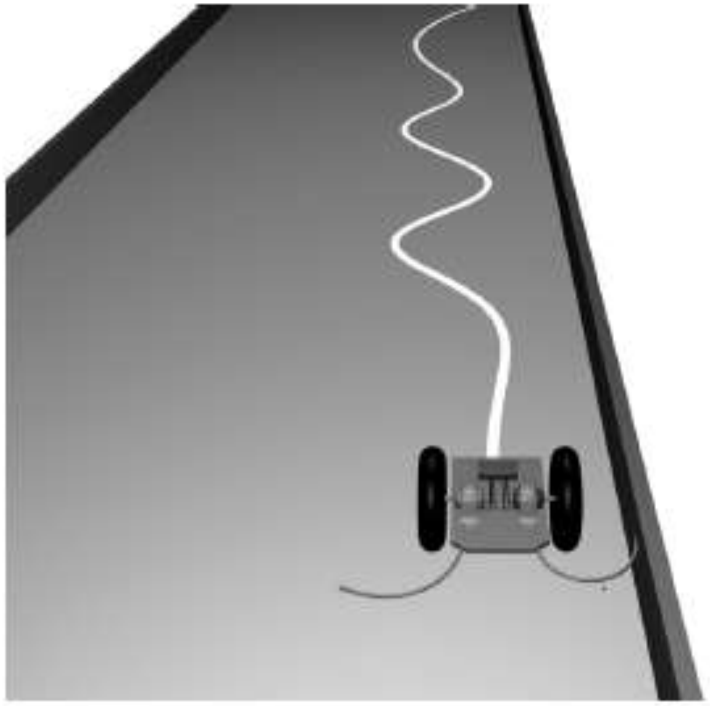
Emergent Behavior (Ch. 18)

Emergent Behavior

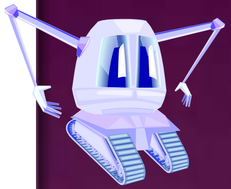
- All robot behaviors result from the interaction of the robot's controller and the environment.
- If these components are simpler then the resulting behavior may be predictable
- If the environment is dynamic or the controller has several interacting components then the resultant robot behavior may be a surprise
- If the unexpected behavior has some structure, pattern, or meaning it is often called *emergent*

Wall Following Emergent Behavior

- Rules:
  - If left whisker bent, turn right
  - If right whisker bent, turn left
  - If both whiskers bent, back up and turn left
  - Otherwise, keep going
- Based upon these behaviors the robot will follow a wall although the robot knows nothing about walls and it is not explicit in the rules



The whole is greater than the sum of its parts



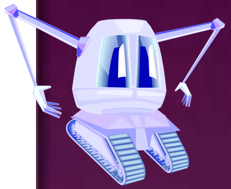
- Emergent behavior appears to produce more than what the robot was programmed to do.
- We get more than we built in or something for nothing
- Roboticists sometimes exploit this potential to design clever and elegant controllers
- In particular, reactive and behavior-based systems are designed to take advantage of such interactions

Flocking Example

- To make robots flock together when they cannot communicate with each other, have their own rules, and own local sensory data, use emergent behavior
- This can be achieved with the following rules
  - Don’t get too close to other robots or obstacles
  - Don’t get too far from other robots
  - Keep moving if you can
- To get the robots to move to a certain destination use another rule which moves each robot tward the goal point

Components of Emergence

- Emergence depends on two components
  - Existence of an external observer to see the behavior and describe it
  - Access to the innards of the controller to verify that the behavior is not explicitly specified in the system
- Alternate definition of emergent behavior
  - *A structured (patterned, meaningful) behavior that is apparent at one level of the system (observer's viewpoint) but not at another (controller's/robot's viewpoint)*



Architectures and Emergence

- Different control architectures affect the likelihood of generating emergent behavior in different ways
- *Reactive* and *behavior-based systems* employ parallel rules and behaviors which interact with each other and the environment which provide the perfect foundation for exploiting emergent behavior
- Deliberative and hybrid systems do not have parallel interaction and would require environment structure and thus minimize emergence