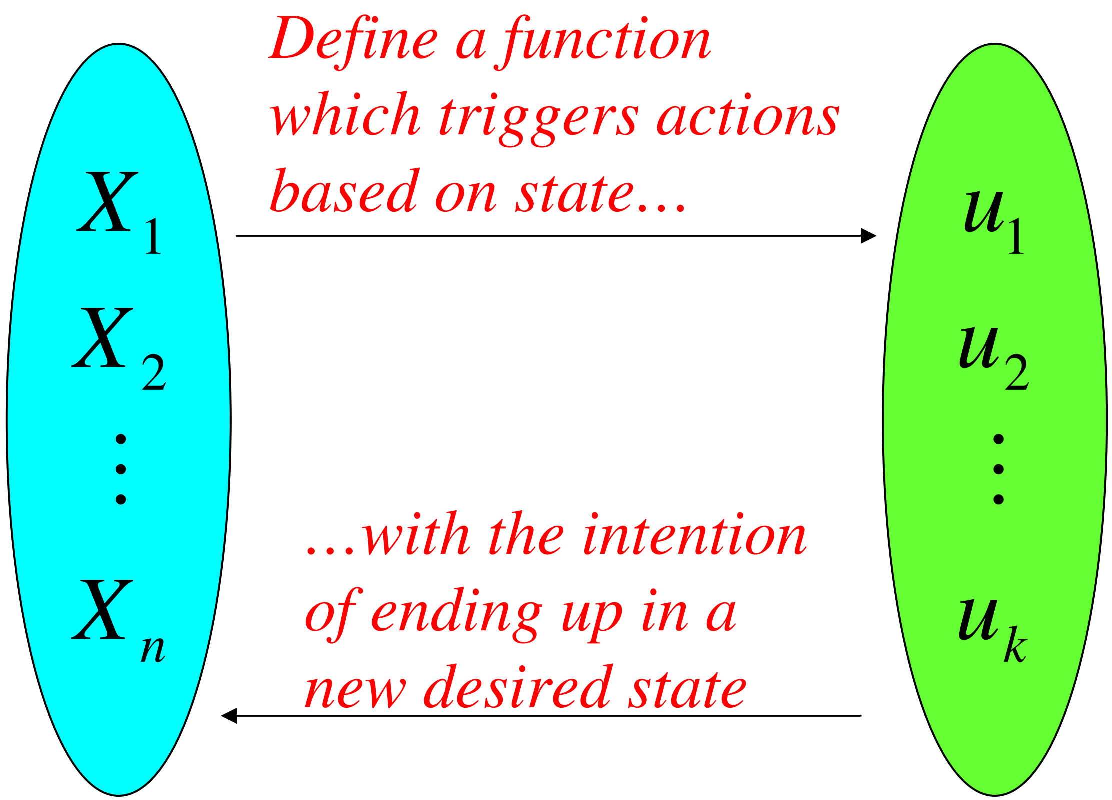
Behaviors

Manuela Veloso
CMRoboBits, 15-491, Fall 2007
http://www.andrew.cmu.edu/course/15-491
Computer Science Department
Carnegie Mellon University

Behaviors as Functions



Domain of state space
(continuous or discrete)

Range of robot actions
(including those of the team)

“Thinking”… Selecting Actions

- Sensory data as input
- “Behaviors” as *processing of input to select actions*
- Actuators perform the actions

Behaviors Approaches

There are three main approaches to behaviors

- Reactive
  - Try to respond directly to the environment
- Deliberative
  - Think ahead about actions before deciding on one to execute (included Planning as special case)
- Hybrid
  - Combination of the above

Reactive Behaviors

- Reactive behaviors map directly from sensors to actions
  - No memory
- Advantages
  - Very responsive to changes in environment
  - Simple and easy to understand
  - Smooth control changes in response to smooth changes in sensor values
- Disadvantages
  - Can't perform different actions from the same state
  - Can get stuck
  - Don't scale well to complex tasks

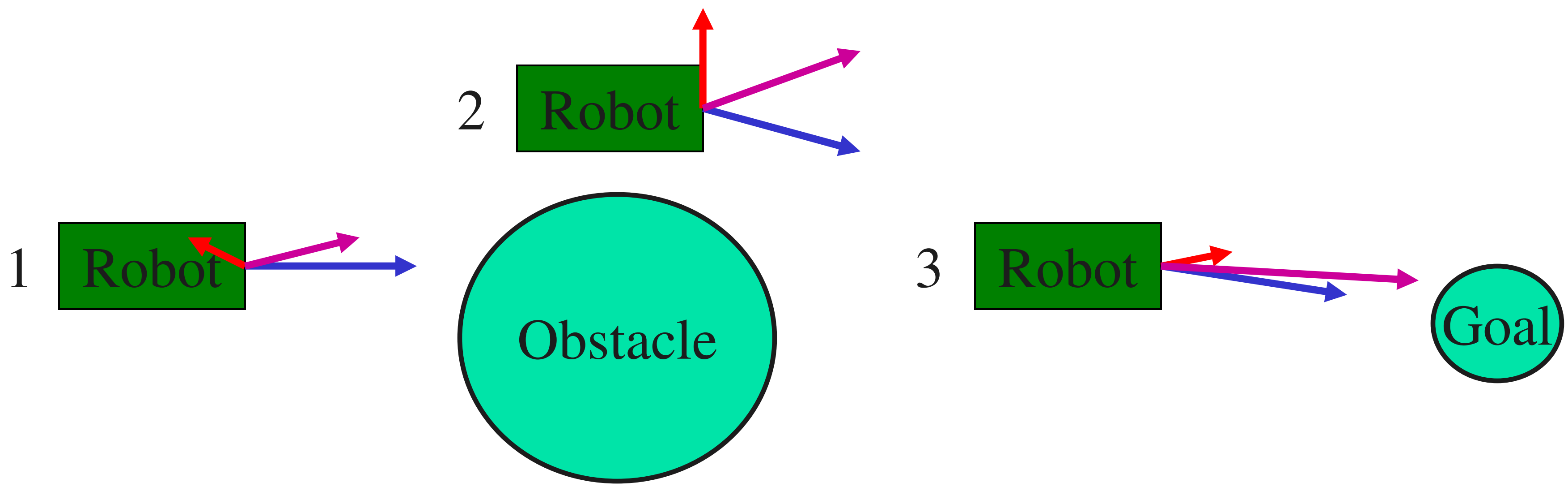
Types of Reactive Behaviors

- Reactive behaviors come under a wide variety of names
- Regardless of the names, they typically behave in the same general way
- Reactive behaviors form the basic building block of most successful behavior system
- An example behavior system:
  - Motor schemas

Motor Schemas

- A *motor schema* is a mapping from sensors to a force vector whose direction dictates the robot's next motion
- Each motor schema calculates a force on the robot due to some constraint
- The force vectors are summed to get the total force on the robot
- Example: navigation in the presence of obstacles
  - One motor schema produces force towards goal
  - Second motor schema produces force away from obstacles

Motor Schemas



Goal vector

Avoidance vector

Resulting vector

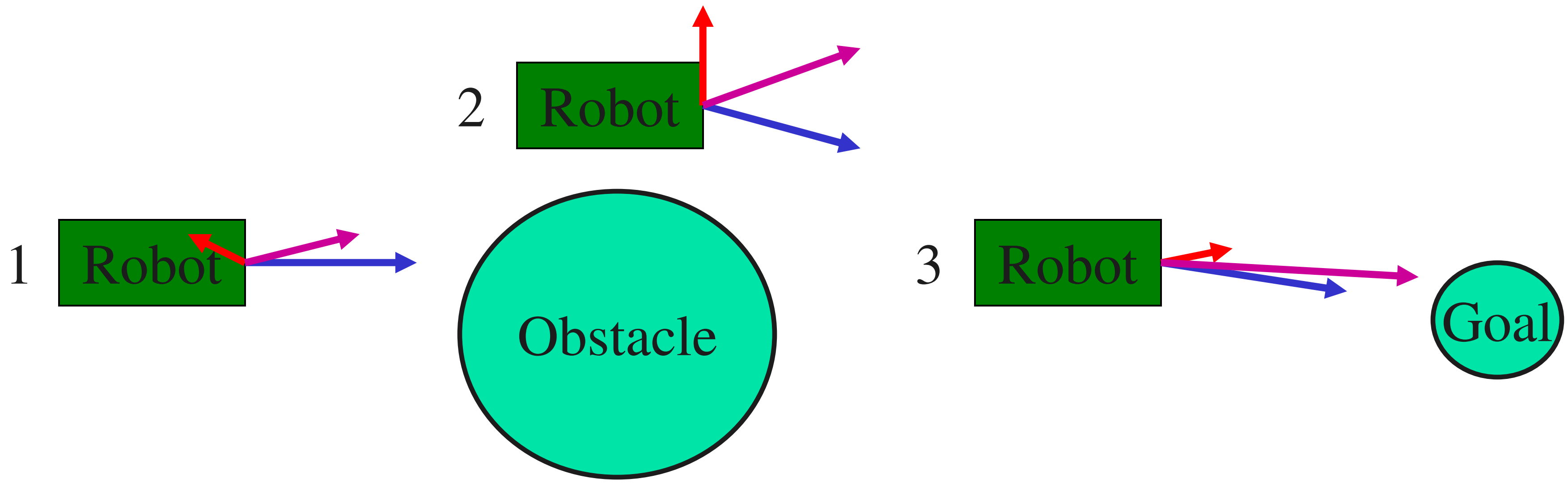
Combining Reactive Behaviors

- Reactive behaviors don't scale very well
  - How to get a reactive system to carry out a conversation with you?
- Reactive behaviors need to be combined into a larger behavior system
- Some combination ideas include:
  - Blending – motor schemas is an example of this
  - Competition – behaviors compete for control
  - Subsumption – reactive behaviors selectively take control
  - Sequencing – reactive behaviors are executed in a sequence based on a higher-level controller

Blending

- Behaviors output an activation magnitude and direction
- Multiple behaviors have their activation values merged into a single unified value
- Easy to implement as long as sensor values can be described by "forces" with direction and magnitude
- Problem: equal but opposing forces can cancel each other out

Blending



Goal vector

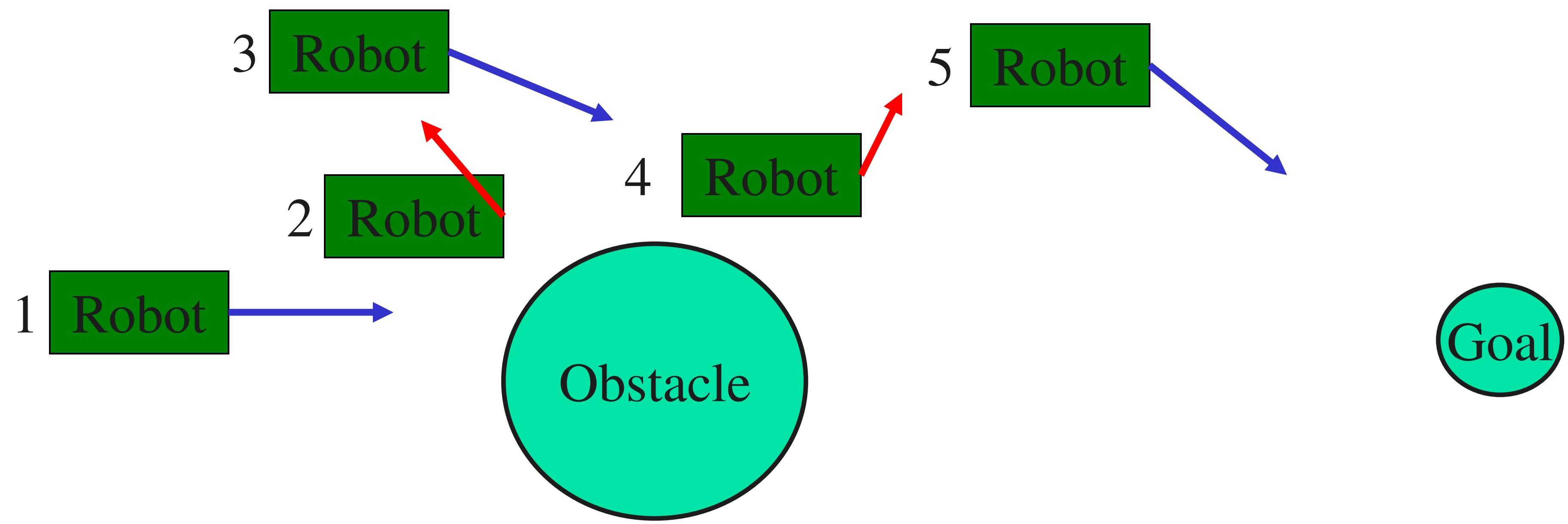
Avoidance vector

Resulting vector

Competition

- Similar to blending, but uses "winner-take-all" for activation
- Reactive behaviors compete for control of the robot
- Very responsive and adaptable to different behavior sets
- Problem: oscillations could occur when two behaviors have very similar strengths

Competition



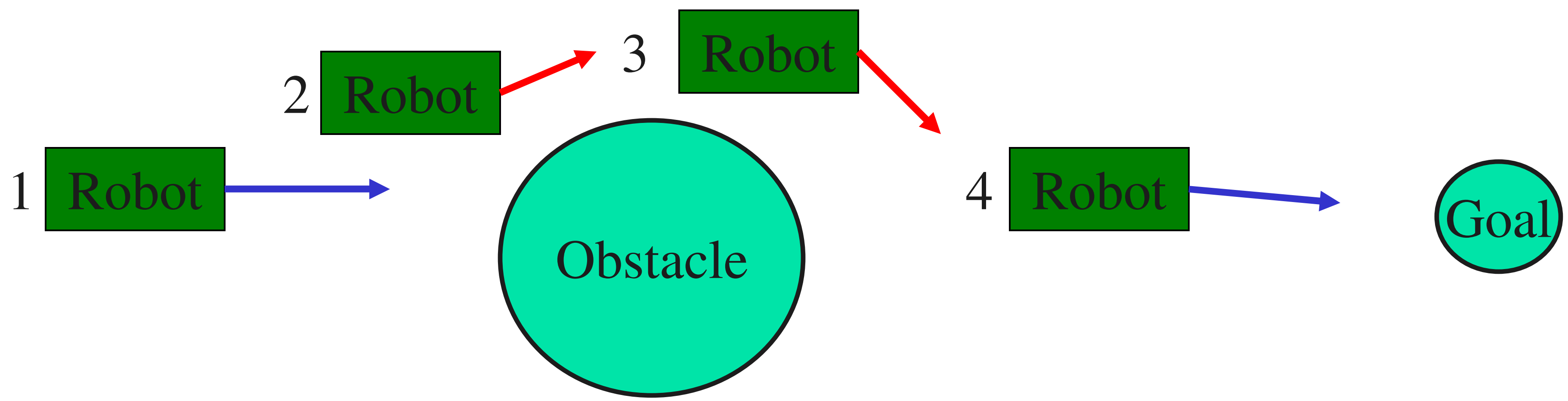
Goal vector

Avoidance vector

Subsumption

- Provide a strict priority ordering for the behaviors
- All behaviors read from sensors and output values to actuators
- Higher priority behaviors override the outputs from lower-level behaviors
- Better scalability due to strict ordering and notion of abstraction

Subsumption



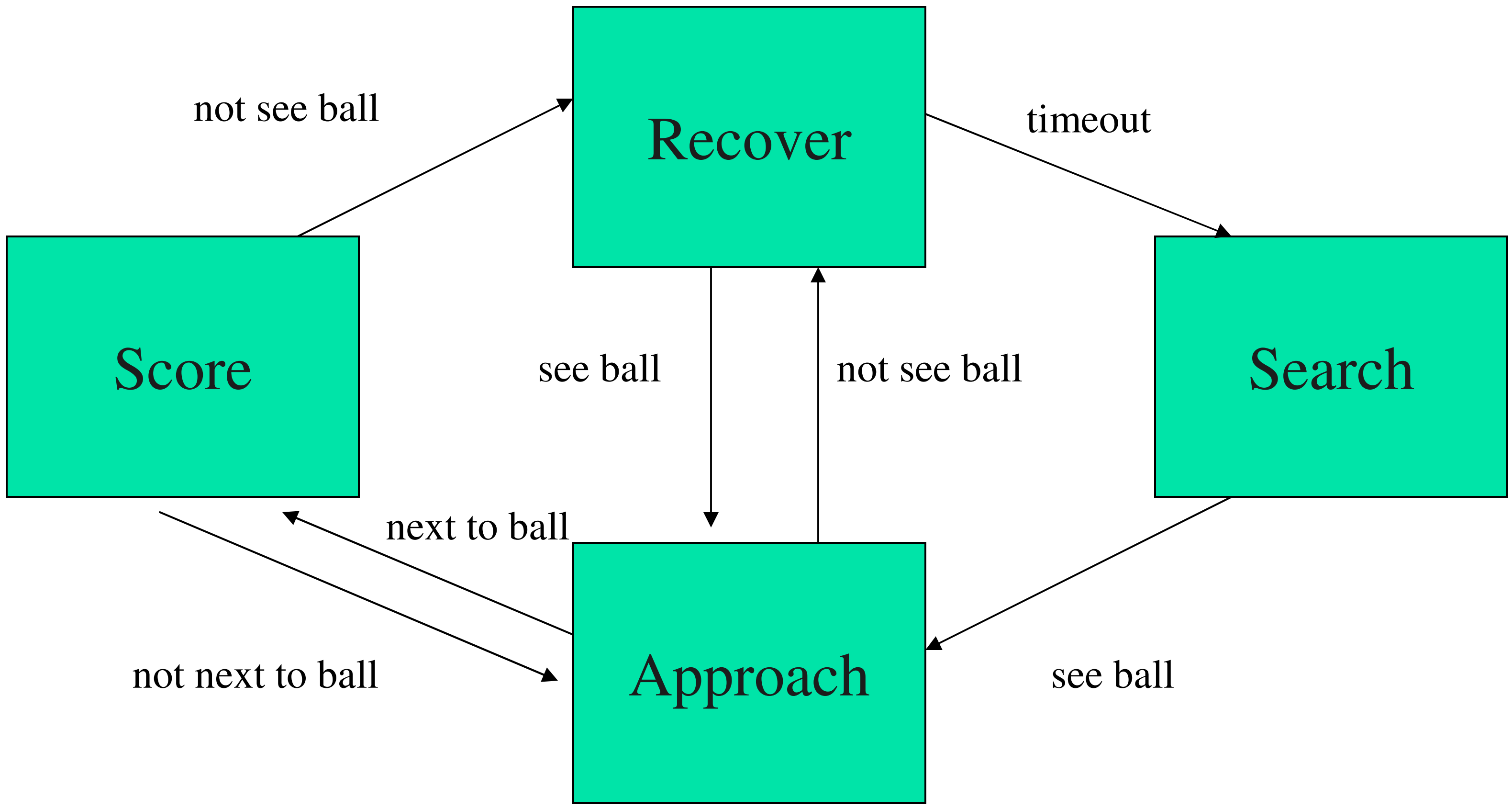
Goal vector

Wall follow vector

Sequencing

- Run only a single reactive behavior at a time and switch the active behavior based on change in robot/environment state
- Convenient notation for sequencing the behavior is a finite state machine (FSM)
- Each state of the FSM has an associated reactive behavior
- Each transition of the FSM has a rule that must be satisfied before a transition can occur
- Approach used by CMRoboBits code

Behaviors as Finite State Machines



Behavior FSM Semantics

- Each behavior is a function which must return a value every time new sensor data is called
- Takes as input the sensor features and returns the actuator commands
- Inside the function is the FSM
  - First, remember which state the FSM is in
  - Do computations on persistent values
    - Time in state, as an example
  - Decide whether to exit the function or whether to transition to a new state
    - Why shouldn't the FSM make the transition state and then exit?

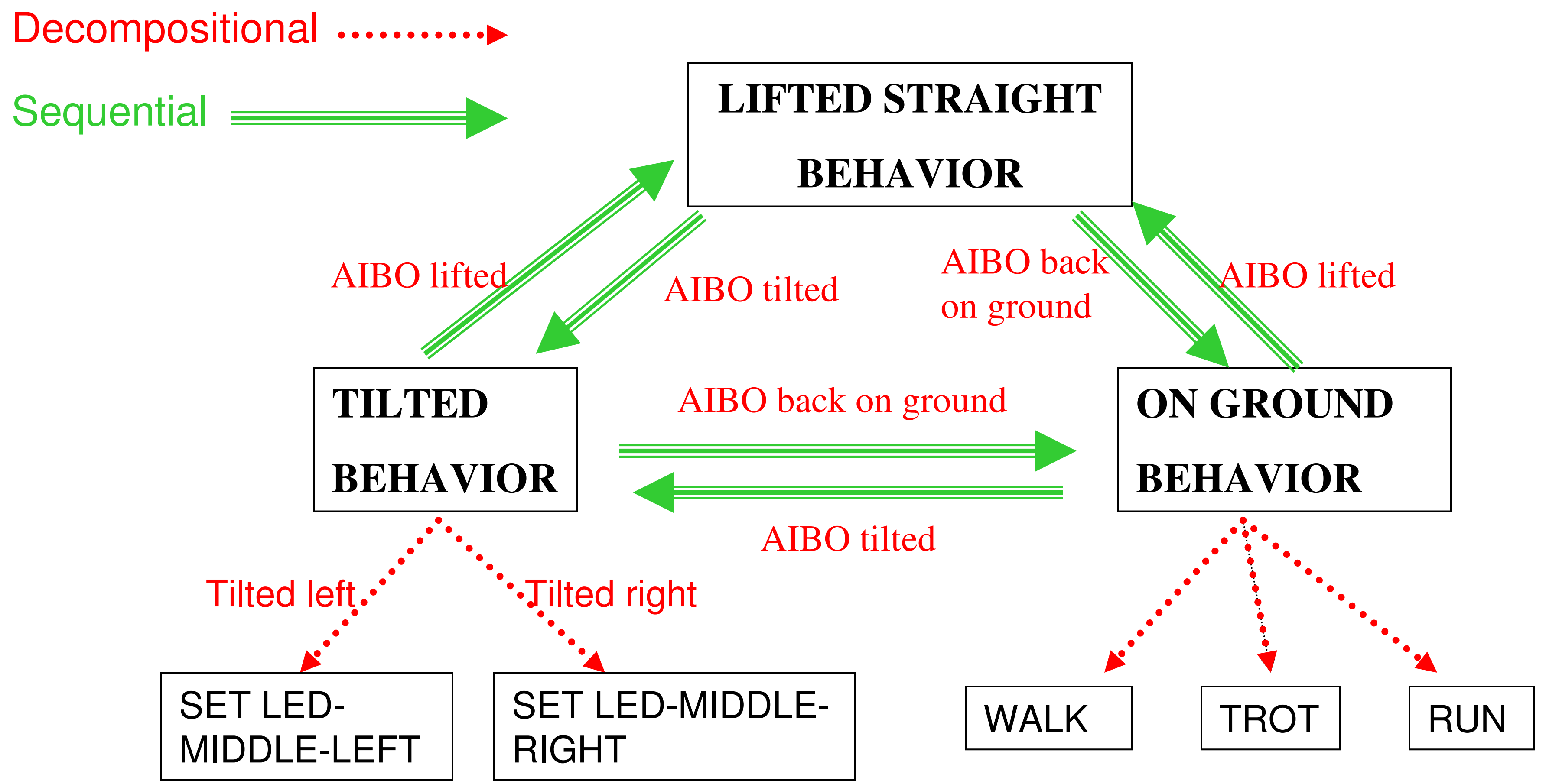
Sequencing Advantages

- Problems with oscillation are greatly reduced by the transition rules
- Can be very reactive to environment
- Can select different actions from same perceptions using context and memory
- Easy to chain together into larger actions

Hierarchy – Adding Scale

- In order to scale to large behaviors, we can reuse collections of lower-level behaviors
  - Libraries of lower-level behaviors form the building blocks for all AIBO behaviors
- Each state of FSM can be either a single reactive behavior, or another FSM with its own behaviors (or FSMs)

Example of Behavior/FSM



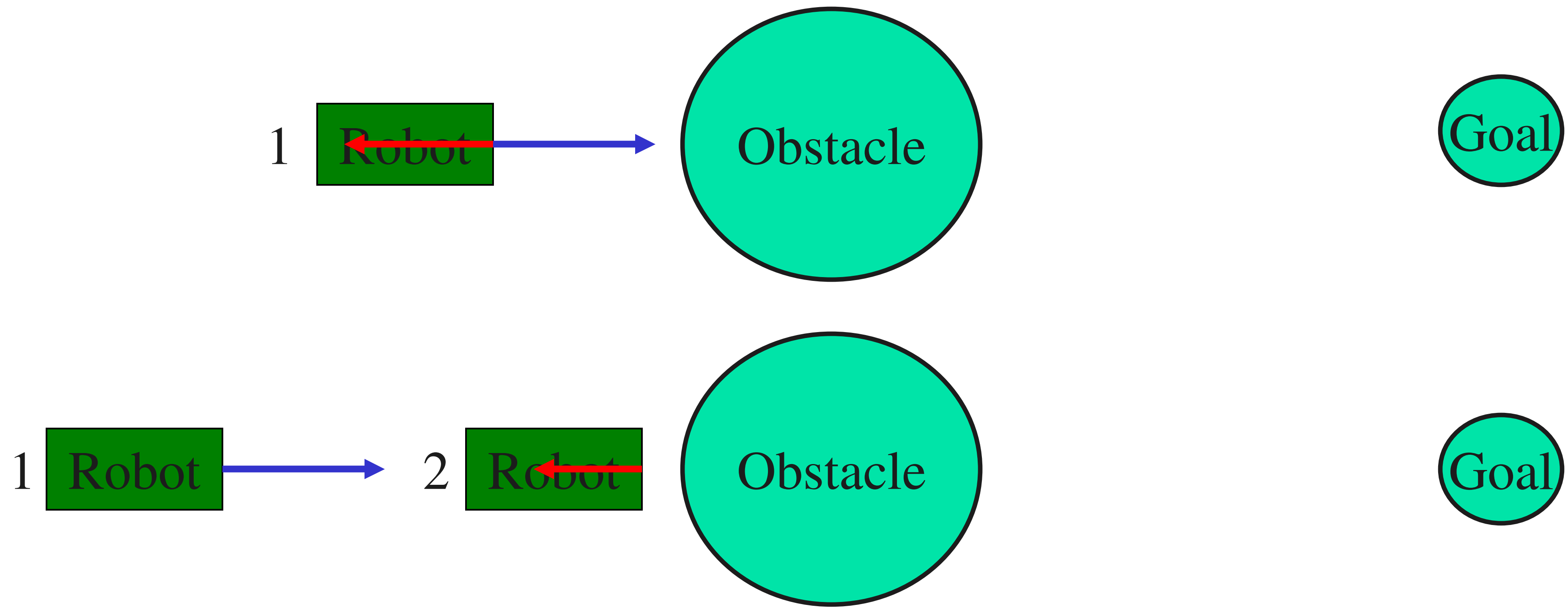
Implementation Details

- Behavior design is more of an art
- Good behaviors produce smoothly varying control signals
- Control signals that oscillate lead to poor control performance
  - Control target changes before controller can achieve the previous target
- Oscillation in behaviors needs to be avoided because it will lead to oscillations in control signals

Problems of Oscillation

- Behavior design can feel like more of an art
- Good behaviors produce smoothly varying control signals
- Control signals that oscillate lead to poor control performance
  - E.g. Control target changes before controller can achieve the previous target
- Oscillation in behaviors needs to be avoided because it will lead to oscillations in control signals

Oscillation



Goal vector

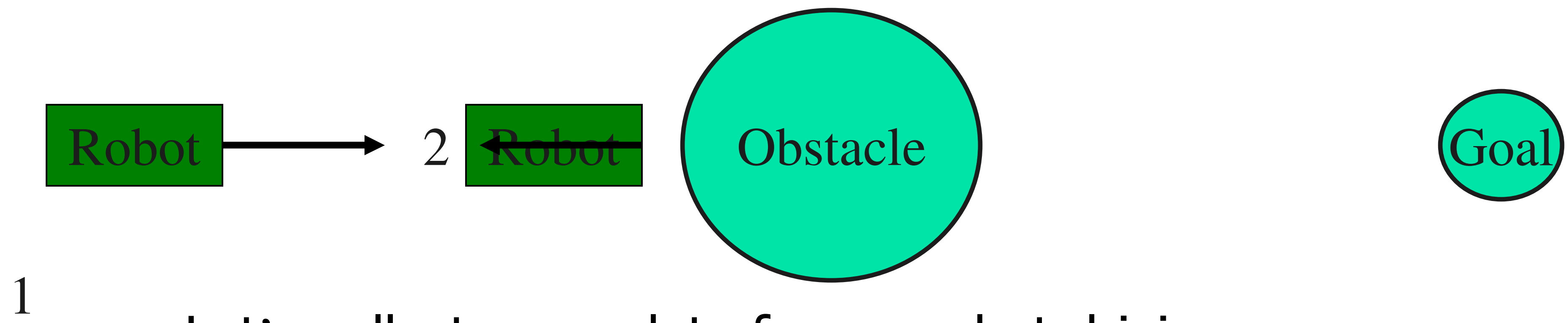
Avoidance vector

Avoiding Oscillation

- If oscillation occurs, one choice is to merge the states where the oscillation is occuring
- A more general solution is to add *hysteresis* to the transition rules
  - A system exhibits hysteresis when the behavior depends not only on the current state but also on its history
  - This refers to the creation of a buffer zone between states
- Important for first sensors homework

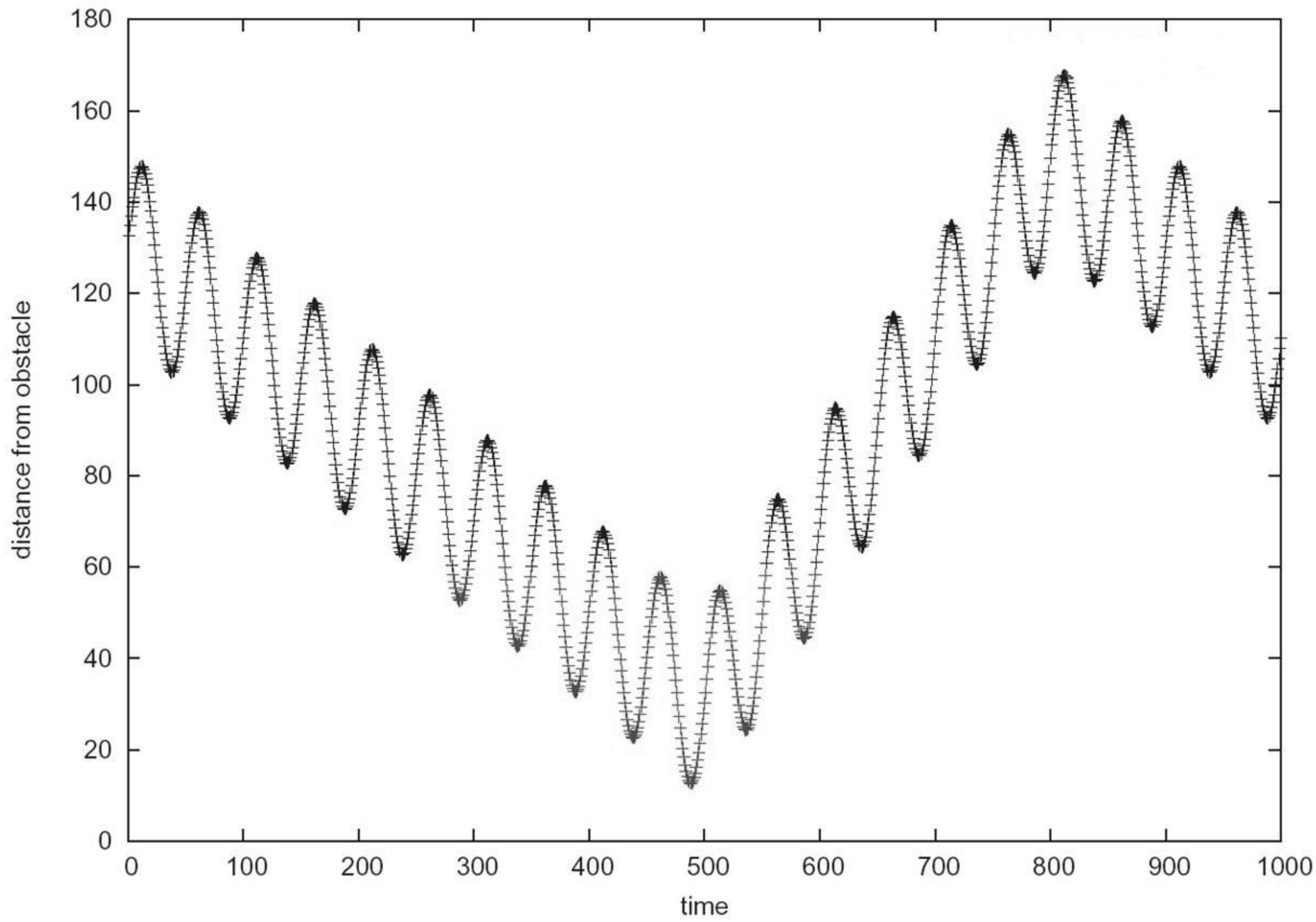
Example: When to Invoke the Different Behaviors?

Challenge: sensors are ***noisy*** , actuation can be noisy



Let's collect some data from a robot driving towards and away from an obstacle.

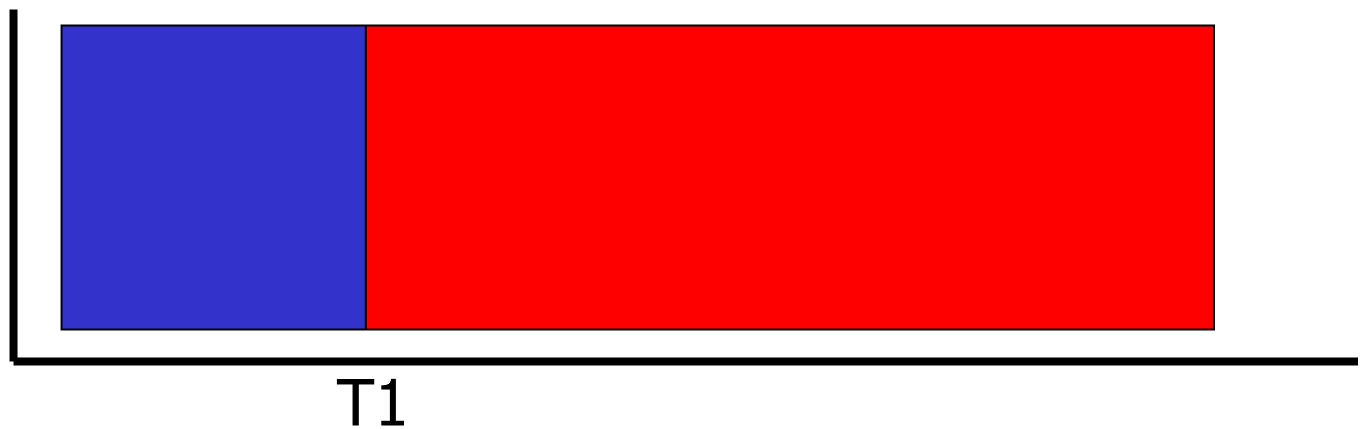
Raw Data



distance from obstacle
time
0
20
40
60
80
100
120
140
160
180
0
100
200
300
400
500
600
700
800
900
1000

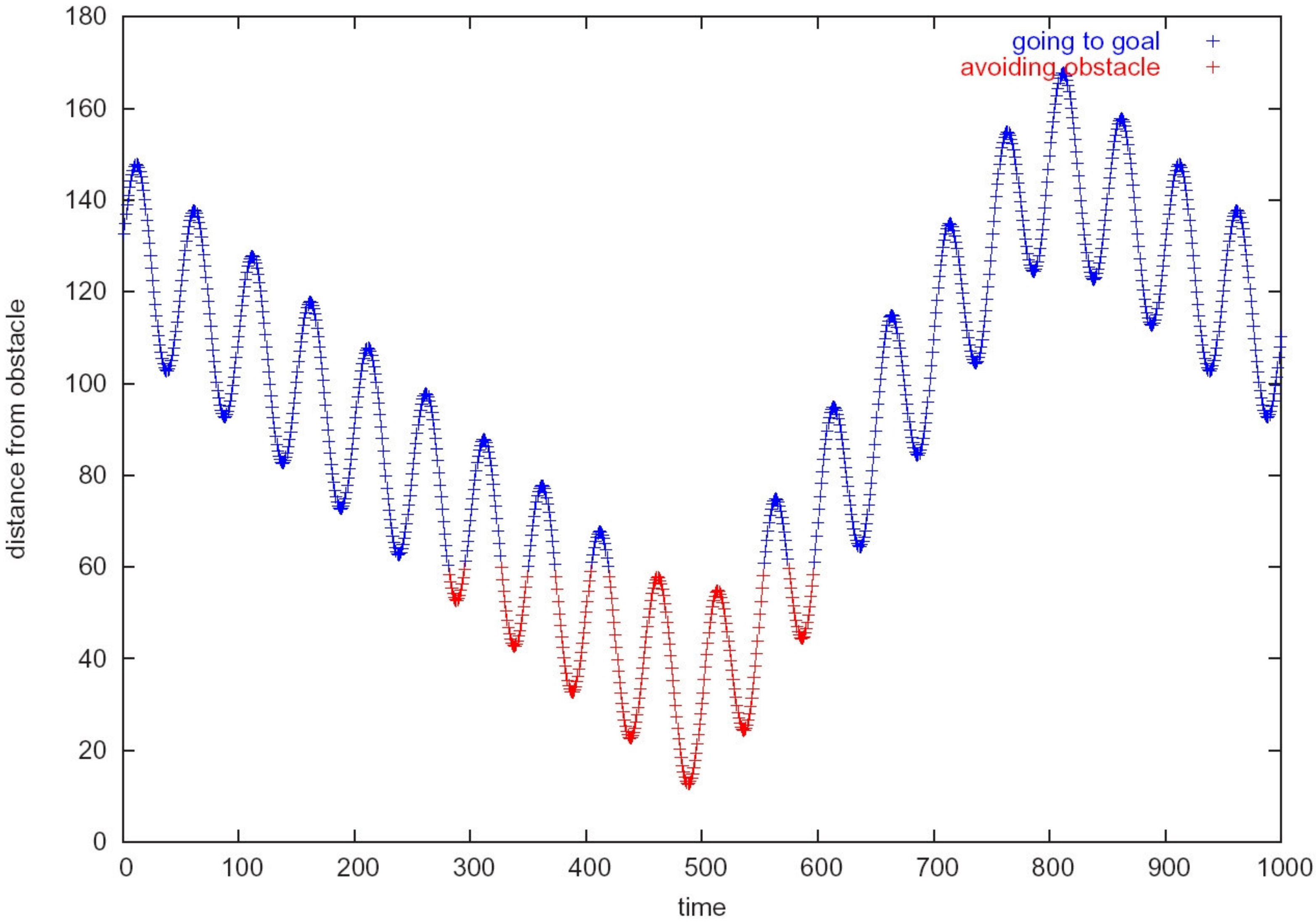
Handling Uncertainty

- Sensory data is noisy
  - How to decide between two conflicting sensor readings?
  - One solution is thresholding



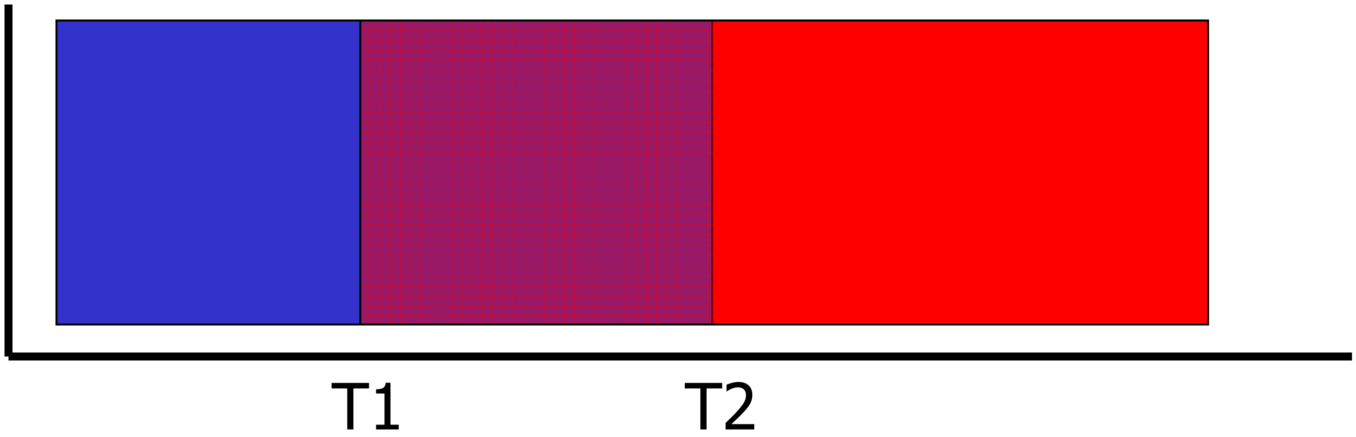
Question: is a single value a good solution?

Naïve Control



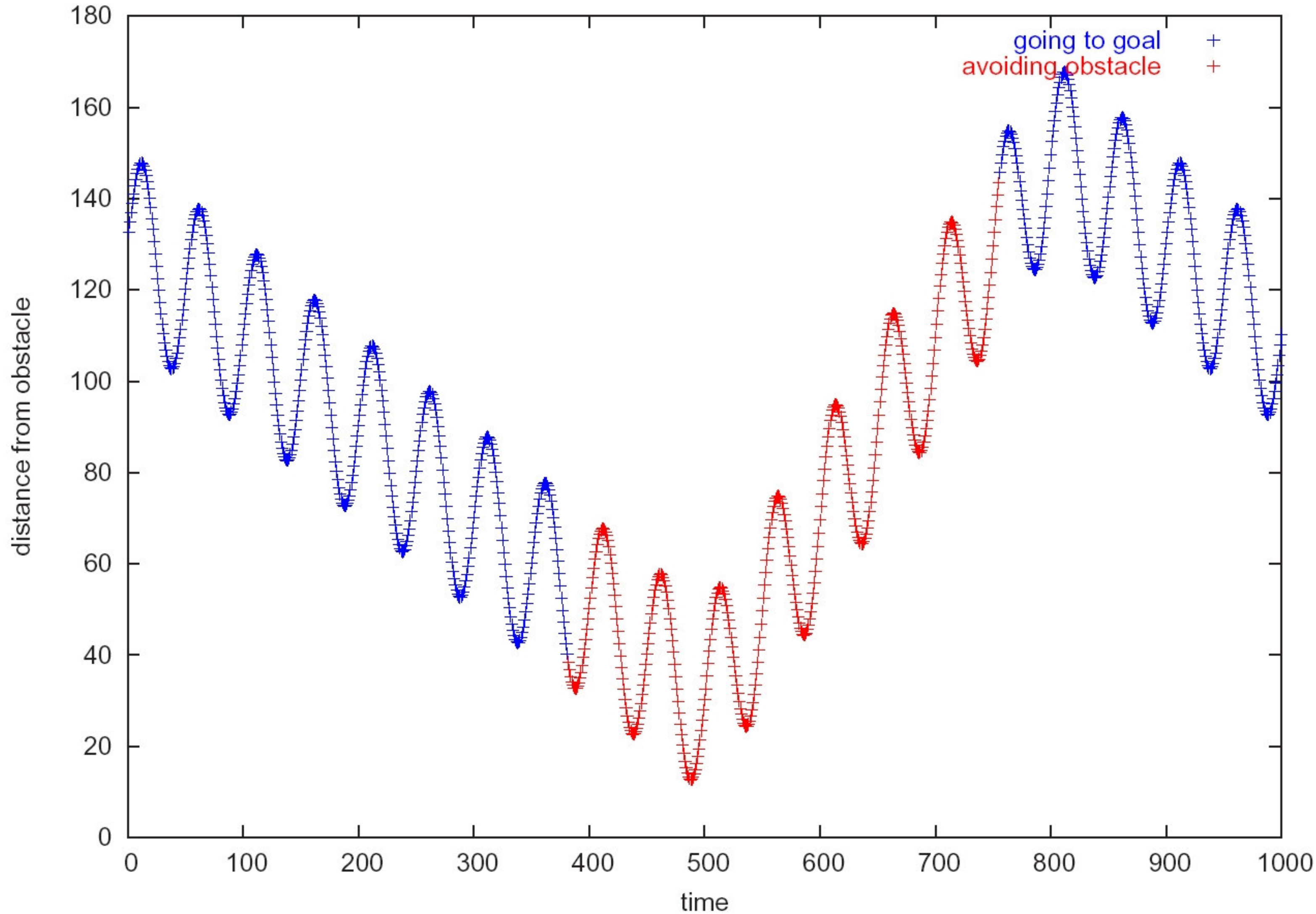
Handling Uncertainty

- Two threshold method: hysteresis
  - The lagging of an effect behind its cause



- The state switches from blue to red when values rise above T2. It switches from red to blue when values fall below T1.

Hysteresis



going to goal
avoiding obstacle
distance from obstacle
time
0
20
40
60
80
100
120
140
160
180
0
100
200
300
400
500
600
700
800
900
1000

Behavior Design Principles

- Design behavior in stages
  - Work on only one state at a time
- Start with initial “entry” state
  - Continue with each successive state
  - Test and debug one transition at a time
- Make one change at a time and thoroughly test the states and transitions

Behaviors: Working within the Perception/Cognition/Action Loop

- Sensors obtain data at a fast rate
- One pass through the loop should not slow this processing down
  - Too much time in cognition might cause data data be missed or lost
- Computation cannot take "too much" time…