Designing Behavior-Based Systems

September 24, 2002

Class Meetings 10-11
Schedule Reminder

• NO CLASS: Thursday, 9/26
• NO CLASS: Tuesday, 10/1

• Makeup class for Thursday, 9/26:
  1. Today (meeting until 6:10)
  2. Or, tomorrow (Wednesday, 9/25), 8:00-9:00, room C223

• Makeup class for Tuesday, 10/1:
  1. Thursday, 10/3 (meeting until 6:10)
  2. Or, Friday, 10/4, 9:00 – 10:00, room 223
Today’s Objectives

• Review: Exam #1
• Preview: Assignment #3
• Preview: Final Project

• Understand how to design and program behaviors
• Understand how to develop a complete behavioral system
• Understand how to develop a “behavioral table” for behaviors
• Understand how to sequence behaviors
• If time allows: introduction to sensors, tour of robots in Distributed Intelligence Lab
Designing Behaviors

• Issues:
  – More “art” than “science”
  – How to build up to integrated systems?

• Approaches:
  – View behaviors as objects in object-oriented programming
  – Sequence behaviors using:
    • Finite state automata
    • Scripts
Behaviors in Object-Oriented Programming

• Object-Oriented Programming: data-centered view of programming

• Data and behavior are conceived of as classes whose instances are objects

• Objects:
  – Class variables, responsible for own behavior
  – Consists of:
    • Data (or, attributes)
    • Methods (or, operations)

• Inheritance: used to conveniently derive a new type from an existing type

NOTE: If you haven't studied object-oriented programming before, find a descriptive text on the topic and read the first chapter for further background.
Behaviors in Object Oriented Programming

Schema is a class
Optional method for schema class
Three classes derived from schema class

Perceptual Schema
Motor Schema
Behavior

Perceptual schema:
- Has at least one method, which takes sensor input and transforms it into a data structure called a *percept*
- Linked to sensor(s)

Motor schema:
- Has at least one method, which transforms *percept* into a vector representing an action
- Linked to actuator(s)

Behavior can have multiple perceptual and motor schema, and even (recursively) other behaviors
• **Primitive behaviors:**
  – Only one perceptual schema
  – Only one motor schema
  – No need for coordinated control

  – Usually programmed as single method
  – Concept of perceptual and motor schemas hidden for sake of implementation
Example Primitive Behavior: Move-to-goal

• **Murphy text:** robot competition requiring picking up Coca-Cola cans (red) and putting them in the recycle bin (blue)

• Rather than build:
  – Move_to_red
  – Move_to_blue

• Instead, build:
  – Move_to_goal(color)

• Minimizes opportunity for introducing multiple bugs
Move_to_goal as Primitive Behavior

```python
move_to_goal(goal_color):
```

<table>
<thead>
<tr>
<th>Object</th>
<th>Behavioral Analog</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>percept</td>
<td>goal_angle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>goal_strength</td>
</tr>
<tr>
<td>Methods</td>
<td>perceptual_schema</td>
<td>extract_goal(goal_color)</td>
</tr>
<tr>
<td></td>
<td>motor_schema</td>
<td>pfields.attraction(goal_angle, goal_strength)</td>
</tr>
</tbody>
</table>

- “pfields” is a class
- “Attraction” is a method within the pfields class
Important Points about Programming with Behaviors

• Behavior is the “glue” between perceptual and motor schemas.
  – The schemas don’t communicate with each other.
  – Behavior puts “percept” created by perceptual schema in a local place where the motor schema can get it

• Behaviors should use libraries of schemas
  – E.g., “pfields” class can encapsulate five primitive potential fields, which any motor schema can use

• Behaviors can be reused if written properly
  – E.g., “go_to_goal” can be reused with different colors
More Complex Example: Follow-corridor

- Two different implementations possible:
- First, use primitive potential fields as motor schemas
- Recall:

1 Uniform

+ 2 Perpendicular

= Follow_corridor
Primitive Potential Fields as Motor Schema

Vector summation of MS

Follow corridor

Perceptual Schema 1: find-walls
Motor Schema 1: perpendicular
Motor Schema 2: uniform
Second Implementation: Summation of Behaviors

- Here, follow corridor composed of two instances of follow wall behavior

```
Follow corridor
Vector summation of behaviors

Behavior 1: Follow-wall (left)
Behavior 1: Follow-wall (right)

Motor Schema: follow-wall (which-side)
Perceptual Schema: Find-walls
```
Use of Releasers in OOP

• Recall: Perception serves two purposes:
  – To release a behavior
  – To guide a behavior

• How to “attach” releaser to behavior?

• Can either:
  – Have releaser be separate perceptual schema not bound to a motor schema
    • When triggered, send signal to main program to note releaser for another behavior
  – Or, have releaser as part of behavior
    • Behavior always active
    • But, if releaser isn’t satisfied, coordinated control short-circuits processing
    • Behavior returns identify function, or potential field of (0,0), which is the same as if behavior weren’t active
Steps in Designing Behavioral System

• Recall three design approaches we studied earlier:
  – Ethologically-guided
  – Situated activity-based
  – Experimentally driven
1. Ethologically Guided/Constrained Design

- Use studies of animal behavior (recall last class discussions) for inspiration/guidance
2. Design Methodology for Situated Activity

Assess Agent-Environment Dynamics

Partition Into Situations

Create Situational Responses

Import Behaviors to Robot

Run Robotic Experiments

Evaluate Results

Enhance, Expand, Correct Behavioral Responses

Done
3. Experimentally Driven Design

- Created in a bottom-up manner
- Iterative design

Diagram:
- Build Minimal System
  - Exercise Robot
  - Evaluate Results
  - Add New Behavioral Competence
  - Done
More Comprehensive Design Methodology

1. Describe the task
2. Describe the robot
3. Describe the environment
4. Describe how the robot should act in response to its environment
5. Implement and refine each behavior
6. Test each behavior independently
7. Test behaviors together

Specification and Analysis:
Ecological niche

Design

Implementation and unit testing

System testing
Case Study: 1994 Unmanned Ground Robotics Competition

- **Objective:** have small unmanned vehicle autonomously navigate around an outdoor course of white lines painted on grass

- **Step 1: Describe the task.**
  - Follow path with hairpin turns, stationary obstacles, and sand pit
  - Robot that went furthest without going out of bounds is winner
  - Tie breaker: robot that goes fastest
  - Max. velocity: 5 mph
  - Penalties: for going out of bounds, for hitting and moving obstacle
  - 3 runs allowed, after 2 days of practice
Case Study (con’t.)

• Step 2: Describe the robot

  – Usually: robot provided, which gives fixed constraints on what is possible
    • Sensors: only specific sensors available
    • Effectors: max speed, turning radius

  – In the case study, additional constraints specified:
    • Robot footprint within given size
    • Robot carries own power, on-board computing
• Step 3: Describe the Environment
  – Grassy field with gentle slopes
  – Ten-foot wide lane marked in white paint
  – Exact length and layout of course not known in advance
  – Obstacles were stationary, and were bales of hay wrapped in white or red plastic
  – Sonar could detect hay bales at 8 feet
  – Run times would be between 9AM and 5PM on May 22, rain or shine
Case Study (con’t)

• **Step 4:** Describe how the robot should act in response to its environment
  – Define candidate primitive behaviors
  – In case study: follow-line
  – Helpful: Behavior Table
Behavior Table

- Helpful organizing tool

<table>
<thead>
<tr>
<th>Releaser</th>
<th>Behavior</th>
<th>Motor Schema</th>
<th>Percept</th>
<th>Perceptual Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>always on</td>
<td>follow-line()</td>
<td>stay-on-path(c_x)</td>
<td>c_x</td>
<td>compute-centroid(image,white)</td>
</tr>
</tbody>
</table>
Case Study (con’t.)

• Step 5: Refine each behavior:
  – Focus on design of each individual behavior
  – Consider:
    • Both normal range of environmental conditions,
    • And conditions in which behavior will fail

  – In case study:
    • Follow-line behavior analysis assumed only white things in environment were lines and plastic covered bales of hay
      – Other white objects: judges shoes, dandelions, etc.
Case Study (con’t)

• Step 6: Test each behavior independently
  
  – Helpful to use simulation

  – Keep in mind that simulators do not usually incorporate accurate models of the robot’s perceptual abilities

  – Often, only way to verify perceptual schema is to try it in real world
Case Study (con’t.)

• Step 7: Test with other behaviors
  – Integration testing, where behaviors are combined
  – Testing in actual environment

  – In case study:
    • Follow_line fooled by hay bales;
    • Perceptual schema included bright pixels from the hay bales
    • Solution: “close robot eyes” for about 2 seconds if hay bale nearby (detected by sonar)

Revised Behavior Table:

<table>
<thead>
<tr>
<th>Releaser</th>
<th>Inhibited by</th>
<th>Behavior</th>
<th>Motor Schema</th>
<th>Percept</th>
<th>Perceptual Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>always on</td>
<td>near = read_sonar()</td>
<td>follow-line()</td>
<td>stay-on-path(c_x)</td>
<td>c_x</td>
<td>compute-centroid(image,white)</td>
</tr>
<tr>
<td>always on</td>
<td>far = read_sonar()</td>
<td>move_ahead(dir)</td>
<td>uniform(dir)</td>
<td>dir</td>
<td>dead_reckon(shaft-encoders)</td>
</tr>
</tbody>
</table>
Assemblages of Behaviors

• Many applications: require some concurrent behaviors and some behaviors in sequence

• Question: How to formally represent the releasers so that the robot executes the behaviors correctly, and so that the human designer can understand what is going on?

• Three common methods for behavior sequencing:
  – Finite state automata
  – Scripts
  – Skills

• Key concept: make world trigger, or release, the next step in the sequence, rather than relying on an internal model of what the robot has done recently
Recall: Finite State Automata Sequencing of Motor Schemas

- Can sequence motor schemas if one activity needs to be completed before another.
- Recall Foraging – FSA diagram:
• **Specify**
  - Finite number of discrete states
  - Each state: list of behaviors that should be active at the same time
  - Start state
  - Final state
  - Inputs = behavioral releasers
  - Transition function: what state the robot transitions to after it encounters an input stimulus
  - Arrows in FSA: indicate releasers for behaviors
Another Method of Sequencing Behaviors: Scripts

- Scripts: templates of behavior assemblages
- Think of scripts as “screen plays”
- Main sequence of events: causal chain
## Comparison of Script Structures to Behaviors

<table>
<thead>
<tr>
<th><strong>Script</strong></th>
<th><strong>Collection of behaviors</strong></th>
<th><strong>Example</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Task</td>
<td>Pick up and throw away a Coca-Cola can</td>
</tr>
<tr>
<td>Places</td>
<td>Applicability</td>
<td>An empty arena</td>
</tr>
<tr>
<td>Actors</td>
<td>Behaviors</td>
<td>WANDER_FOR_GOAL, MOVE_TO_GOAL, GRAB_TRASH, DROP_TRASH</td>
</tr>
<tr>
<td>Props, Cues</td>
<td>Percepts</td>
<td>Red, blue</td>
</tr>
<tr>
<td>Causal Chain</td>
<td>Sequence of Behaviors</td>
<td>WANDER_FOR_GOAL(TRASH), MOVE_TO_GOAL(TRASH), GRAB_TRASH, DROP_TRASH</td>
</tr>
<tr>
<td>Subscripts</td>
<td>Exception Handling</td>
<td>If have trash and drop, try GRAB_TRASH three times</td>
</tr>
</tbody>
</table>
\ For each update...
\ look for props and cues first: cans, trash cans, gripper

rState = extract_color (red, rcs, rSize); \ ignore rSize
if (rState == TRUE)
    SEE_RED = TRUE;
else  SEE_RED = FALSE;

bStatus = extract_color (blue, bcx, bSize);
if (bStatus == TRUE)
    { SEE_BLUE = TRUE;  NO_BLUE = FALSE;}
else {SEE_BLUE = FALSE;  NO_BLUE = TRUE; }

AT_BLUE = looming(size,bSize);
gStatus = gripper_status();
if (gStatus == TRUE)
    { FULL = TRUE; EMPTY = FALSE; }
else  {FULL = FALSE; EMPTY = TRUE; }

Resulting Script for Collecting Coke cans (con’t.)

```c
\ index into the correct step in the causal chain

if (EMPTY)
  { if (SEE_RED)
    move_to_goal(red);
    else wander();
  }
else
  { grab_trash();
    if (NO_BLUE)
      wander();
    else if (AT_BLUE)
      drop_trash();
    else if (SEE_BLUE)
      move_to_goal(blue);
  }
```
Summary of Designing a Behavioral Implementation

- Describe the task
- Describe the robot
- Describe the environment
- Describe how the robot should act in response to its environment
- Refine each behavior
- Test each behavior independently
- Test with other behaviors

- Repeat as needed
Introduction to Sensing/Perception

• Typical robotic sensors:
  – Proprioceptive:
    • Dead reckoning (encoders)
    • Inertial navigation system (INS)
    • Global positioning system (GPS)
    • Compass
    • Gyroscopes
  – Tactile and Proximity:
    • Ultrasound (sonar)
    • Infrared (IR)
    • Bumpers and feelers
  – Computer vision:
    • CCD cameras
    • Laser striping
    • Laser/video ranging
  – Many others…. 
Tour of Robots in Distributed Intelligence Laboratory

http://www.cs.utk.edu/~parker/Distributed-Intelligence-Lab

The Distributed Intelligence Laboratory:

• Engaged in research in cooperative robotics and distributed artificial intelligence

• Characterizes distributed intelligent systems as multiple entities (such as robots or agents) that integrate perception, reasoning, and action to perform cooperative tasks under circumstances that are insufficiently known in advance, and dynamically changing during task execution.
Preview of Next Class (Thursday, Oct. 3rd)

• Sensing/Perception
  – Commonly used sensors
  – Types of information available from sensors
  – Common sensing techniques
  – Sensor noise issues
  – Etc., etc.