EGN 4060c: Introduction to Robotics

Lecture 6:

Robot Architectures: Implementation

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu

Announcements

- Grader: Roberto Alberdeston
  Office hours: HEC 242, MW 1-2pm
- Class cancelled on Sept 17th while I attend the IROS conference; lab report is still due on webcourses
- Reading link fixed on the website
- Notes about the curving line section of Lab 2:
  - Robot should drive continuously
  - Recommend using sleepUnlessStop() method to sleep your thread during loops
- HW1 grades released

HW1: Common Problems

- Spelling and grammar issues
  - Fixing these minor problems is relatively easy and can help make a better impression on the reader.
  - Have a friend proof-read your document.
- Writeup was a synopsis of a single source document written for a general audience.
  - Better to synthesize material from several sources. This add more value for the reader rather than just being a summary of existing material.
- Include references after technical details.
  - When you include specific details (e.g., the robot weighs 10 kgs) it is usually good to include a citation after the sentence or the fact.

HW1: Common Problems

- Summary isn't coherent but more a collection of slightly related facts
  - Make an outline
  - Introduce the project, the purpose of the project, what makes it interesting/hard
  - Method---hardware, software, any details which might be useful. Break it up into appropriate subtopics
  - Results/evaluation
  - Conclusion: what has been achieved, future things the researchers are planning to work on
  - Move from general ideas to specific ones

References

- Most commonly, citations will either be numbers or [author, year] format.
- Generally a reference should include a title and author even if it is only a web page.
- When you use images from other sources it is important that a citation appear in a caption to the image. For work that is published, you need to obtain permission from the original source to use the image.

Types of Publications

- Group website: non-archival documents and videos summarizing group's research
- Workshop: preliminary results
- Conference: complete results of one aspect of the project
- Journal: longer, most complete and detailed version of research project
- Popular magazine article: high-level description of project (skimps on methods and results)
- Textbook: endeavors to present a balanced view covering multiple research projects in a given topic area

Guide to Reading Papers

- Skim the introduction
- Skip to the method section to determine precisely what the authors did
  - Simulation, partial-robot, full implementation, field-test?
- Look at both the references and the related work section to determine what else has been done in that area
- If still interested, give the results and the discussion section a detailed reading to discover and evaluate what the authors accomplished

Architecture Summary

- Deliberative
  - STRIPS
- Reactive
  - Subsumption
  - Potential fields
- Key differences:
  - Ordering of perception/cognition/action steps
  - Existence of world model

Designing a deliberative robot..

- List actions that the robot will need to take to execute the task
- Create the representation for the world model
  - *Representation* is the form in which information is stored or encoded in the robot.
  - Representation of the world around the robot is the *world model*.
- Identify costs for actions
- Decide on search algorithm/heuristics
  - Commonly used ones: A*, DFS, wavefront (navigational)

Representations: Maps

- Landmark based:
  - Store lists of objects in the environment
- Occupancy grid:
  - Discretize the world into a grid and mark the whether each grid cell can be entered by the robot

The basic information is the same; however representing it in different ways can make the computation more efficient…..

Design Issues

- **Closed world assumption**:
  - World model contains everything the robot needs to know.
  - Reasoning process never considers options outside the world model.
- **Frame problem**:
  - How to express a dynamical domain without having to explicitly express which conditions are not affected by an action
  - STRIPS handles this by using add/delete list to enumerate all changes in the world. This can become computationally intractable.

Frame Problem

Robot moves Object A to Location (x,y).

As humans we know:

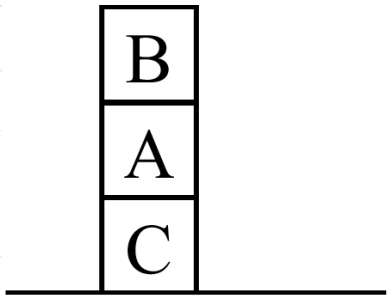
- Object A maintains certain properties after the move action has been applied.
  - Color
  - Weight
  - Size
- Location related properties of the object change:
  - Object A in Room 1 might no longer be true depending on the location (X,Y))
  - Object A might no longer be OnFloor so the value of this proposition might change from true to false.

Frame Problem

- *Robot breaks/slices Object A.*
- As humans we know:
  - Object A is no longer likely to be the same weight or size.
  - However, object A is still probably at the same position and in the same room.
- How can we imbue a robot with the same intelligence as a human?
  - Frame axioms
  - STRIPS add-delete lists

STRIPS: describing goals and state

- On(B,A)
- On(A,C)
- On(C,F1)
- Clear(B)
- Clear(F1)




```
On(B,A)
On(A,C)
On(C,Fl)
Clear(B)
Clear(Fl)
```

- **State descriptions**: conjunctions of ground literals.
- **Start state** contains the list of ground literals which are currently true.
- **Goal state** is the list of ground literals which the robot needs to achieve to complete its task.

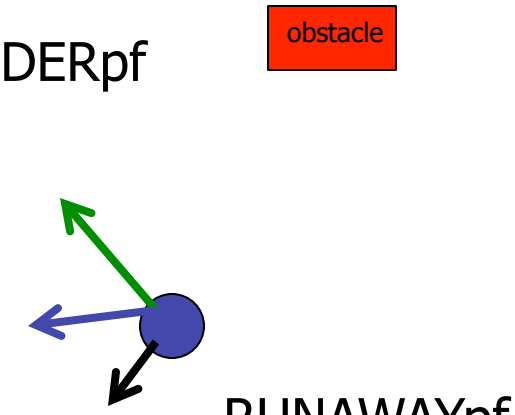
(slide from R. Dechter)

Reactive Architectures

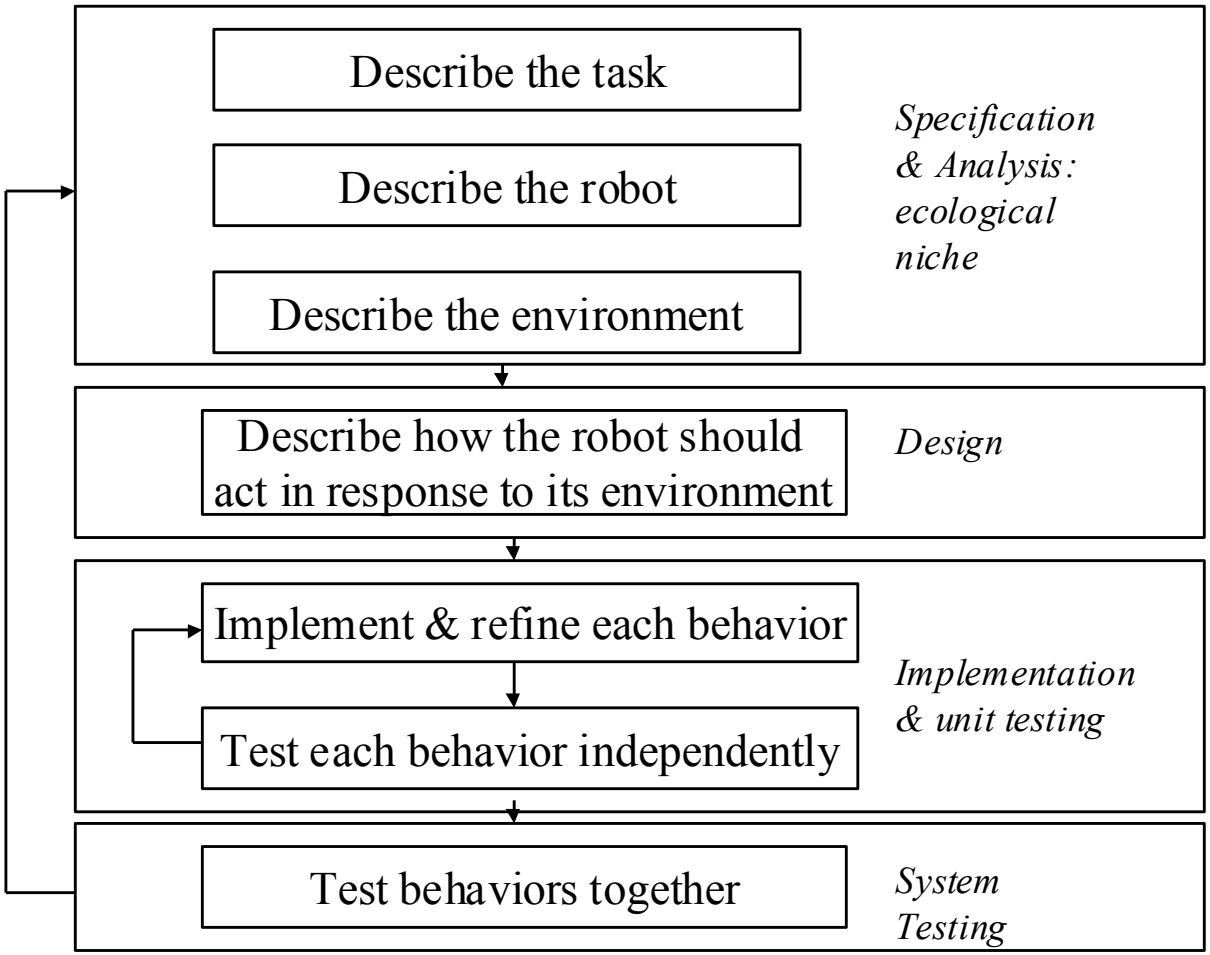
- Switching:
  - Robot switches between following actions followed by different modules (e.g. like in subsumption)
- Fusion:
  - Robot's actions simultaneously combine output from different modules

Potential Fields: Example

```
While (robot==ON)
{
Vector.magnitude=vector.direction=0
For (i=0; i<= numberSonars; i++) {
      Reading=readSonar();
      //perceptual schema
      currentVector=runaway(reading);
       //motor schema
      Vector=vectorSum(vector,currentVector);
      }
      Turn(vector.direction);
      Forward(vector.magnitude*MAX-
         VELOCITY);
}
```



Designing a Reactive System



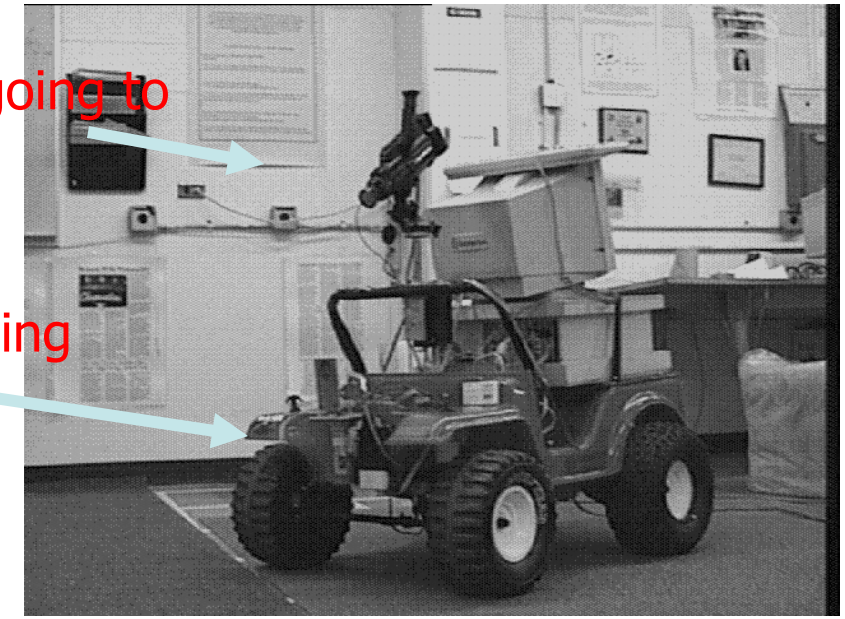
Case Study: CSM 1994 UGV Competition

- Predecessor to the Intelligent Ground Vehicle Competition
- Fully autonomous vehicle, all computation onboard, navigate a course 10ft wide, about 800ft long
- Robot had to follow a path with hair pin turns, stationary obstacles and a sand pit
- Lane marked with white paint
- Outdoor lighting could change
- Carry a 70 pound payload
- Author's team (Colorado School of Mines) won

Describe the Task, Robot, Environment

Camcorder on a panning mast, going to a framegrabber

Sonar on a panning mast



PC running Lynx (commercial Unix)

- Max speed was 5 mph
- Robot's width was smaller than lane so could avoid obstacles by staying at the center of the lane
- Computer vision issues: black & white, slow processor speeds
  - White (bright) should be in the center of the image
  - Reflections on grass are white, but random so average out
- 150ms update rate needed for steering to stay in control at ~1.5mph

Behavior Table

Releaser	Behavior	Motor	Percept	Perceptual Schema
Always on	Follow_line()	Stay on path (c_x)	C_x	Compute-centroid (image, white)

Design Process

- Create behavior
  - Follow-line
- Refine and test behavior
  - Indoor tests: toilet paper
  - Outdoor tests: athletic line tape
- “Full dress rehearsal”
  - Discovery: obstacles (bales of hay) are bright compared to grass and change the centroid to cause collision
- Go back to behavior design:
  - Follow line until “see” an obstacle, then just go straight until things return to normal
    - Hard to do visually in real time
    - Use sonar to distinguish between bales and line

Revised Behavior Table

Releaser	Inhibited	Behavior	Motor	Percept	Perceptual
Always on	Near= read_sonar()	follow_line()	Stay-on-path(c_x)	C_x	Compute _centroid (image,white)
Always on	Far= read_sonar()	Move_ahead (dir)	Uniform(dir)	Dir	Dead _reckon (encoder)

Design Issues

- Reactive architectures often exhibit **emergent** (often unexpected) behavior.
- “The whole is greater than the sum of the parts.”
- Creating and designing many simple behaviors will result in the system having a more complicated global behavior which is difficult to explain.

Emergent Behavior: Flocking

- Combine simple behaviors:
  - Don't get too close to the other robots
  - Don't get too far from other robots
  - Keep moving if you can
- Produce: flocking
  - Robots move together in the same direction without explicitly programming that behavior into the system

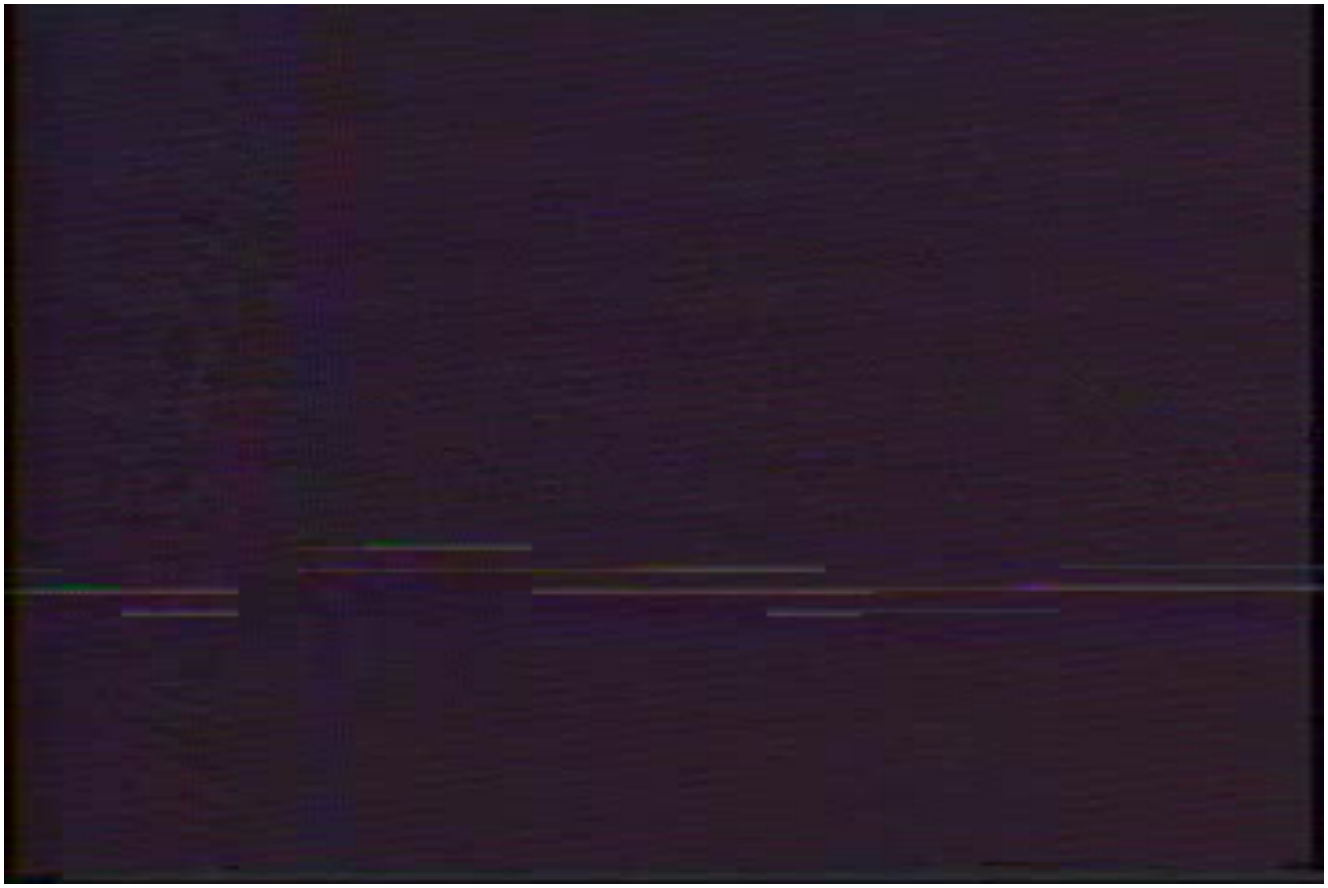
Flocking Video

https://www.youtube.com/watch?v=n_qRuHkD5lc

UGV 1994 Video

https://www.youtube.com/watch?v=Cc-UhlfSLu0&feature=youtu.be

UGV 1994 Video



Adding State: Behavior-based

- What happens when the world proves to be an inadequate model for solving the problem?
- Add a bit of state to the reactive system (behavior-based control)
- Rather than having a full world model, store enough information about previous decisions to improve future decisions.
- Can help by:
  - Preventing local minima
  - Enabling the robot to perform sequential assemblages of behavior
  - Two common representations:
    - Finite state automata
    - Scripts

Finite State Automata: M={K,Σ,δ,s,F}

- K: all the states, each is "q"- behaviors
- δ: transition function, δ(q,σ)= new behavior
- Σ: inputs that agent can "see", each is σ– stimulus/affordances
- q_0: Start state(s)- part of K
- F: Terminating state(s)- part of K

FSA Summary

- "State" doesn't mean "state of world"; it means more "state of the robot's execution"
- Advantages
  - Formal mechanism
  - Enumerating possible transition options helps the designer spot unforeseen effects
- Disadvantages
  - Hard to express implicit behaviors
    - Ex. Avoid obstacle is often running throughout ALL states, gets tedious to express it formally
  - Tend to add explicit variable for transitions, rather than rely on emergent behavior from environment

Tips about FSAs

- States: correspond to robot behavior modes
- Directed links between states: triggered by sensory events or internal state variables (e.g. a counter)
- Directed links can loop back to the same state
- Termination state is useful to include
- Consider all possible state transitions

Scripts

- Generic template for assemblages of behavior
- Originally used by the natural language processing community
- Scripts are have equivalent computational power as a FSA but may be more natural or readable for sequences

Scripts

Script	Behavior Analog	Examples
Goal	Task	Find victims in rubble
Places	Environment, applicability or "taskability" for new tasks	Collapsed buildings
Actors	Behaviors	explore(), dance(), avoid(), crawl, move2void, move2victim
Props, cues	Percepts	Voids: Dark, concave Victims: heat, motion, color
Causal Chain	Sequence of behavior	Explore, dance, move2void, crawl, move2victim, dropRadio
Subscripts	Exception handling	If lose communications, return home

Scripts Summary

- Advantages
  - A more storyboard like way of thinking about the behaviors
  - If-then, switch style of programming like FSA
  - Since a script is “in” a behavior, other behaviors such as avoid can be running concurrently without having to appear “in” the script
  - Exception handling is a big plus in Real Life
- Disadvantages
  - Can be a bit of overkill for simple sequences

Summary

- Studying architectures developed by other robot designers can give you an idea of how to structure your own code.
- Try to think about code readability and reusability.
- Design of reactive systems is identical to design of object-oriented software systems
  - Highly modular, can test behaviors independently
- Follows the basic steps in the Waterfall Lifecycle
  - Describe task, robot, environment, how robot should act, refine behaviors, test independently, test together.
- Finite state machines and scripts are two methods for constructing behaviors.

Design a Robot

1. Find the battery dock with a map of the environment (deliberative)
2. Find the battery dock without a map but with an IR detection sensor (reactive)
3. Design a soda-can collecting robot (subsumption)
4. Design a soda can collecting robot (behavior-based)
5. Design a search and rescue robot (hybrid)

Think about….

- What type of information do you need to give the robot initially
- What type of modules would it be useful to have
- How does the system arbitrate between modules? (decide which module to listen to at a given time)