Introduction to Game Programming and Robotics

Unit # 6

Three Laws of Robotics

- The Three Laws of Robotics (often shortened to The Three Laws or Three Laws) are a set of rules devised by the science fiction author Isaac Asimov and later added to.
- The rules are introduced in his 1942 short story "Runaround". The Three Laws are:
  - A robot may not injure a human being or, through inaction, allow a human being to come to harm.
  - A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
  - A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Behavior

- Behaviors are the way to implement functionality when using the behavior framework. A behavior represents an action or reaction to events (triggers) and resulting request to control where the robot moves.
- A behavior may have zero or many triggers connected to its input.
- An example of a behavior with no triggers is CRUISE. It processes no input from the robot. It requests to drive the robot given its two internal constants; speed and steering. Every time the behavior runs, the output is the same.

Behavior (Cont'd)

- An example of a behavior implements a trigger is BUMP ESCAPE. It monitors collisions with an object on its bumpers and requests action from the robot based on that input.
- The behavior does nothing until the bumpers are depressed. The robot requests to back up for a time, spin right or left, move forward when a collision occurs and relinquishes control when the escape is complete.
- Other examples of a behavior trigger are: an amount time passing, a random number generator, a sensor value, battery level, or a combination of inputs.
- There are two types of behaviors:
  - Servo
  - Ballistic

Servo Behavior

- Typically, a servo behavior employs a feedback control loop as its control component.
- These behaviors may or may not use external input, but provide instant feedback as an output.
- CRUISE is the simplest form of servo behavior. It takes no input and outputs a constant speed and steering.
- These behaviors all process input (or lack thereof) and determine motor commands directly from that input in a stateless feedback/control loop.

Ballistic Behaviors

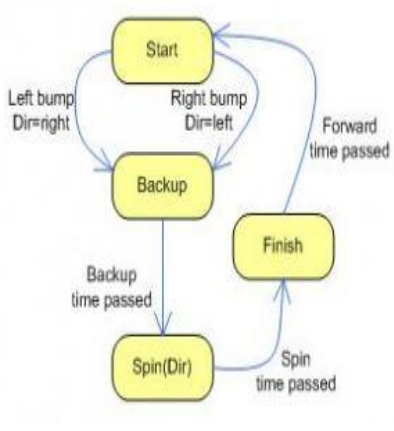
- A ballistic behavior, like a shell fired from a cannon, once triggered, follows a predictable trajectory through to completion.
- In ballistic control, the position trajectory and velocity profile is computed once, then the actuator carries it out.
- There are no “in-flight” corrections, just like a ballistic missile doesn’t make any course corrections.
- In order to accomplish a precise task with ballistic control, everything about the device and how it works has to be modeled and figured into the computation.
- The ballistic control is open-loop control.

Ballistic Behavior (Cont'd)

- Another term for a ballistic behavior is a finite state machine. Ballistic behaviors maintain state between successive calls in order to complete a series of steps before relinquishing control of the robot.
- An interesting note about behavior-based programming is that behaviors each run for a fraction of a second at a time in this framework. That serves servo behaviors well because they may alter their commands with each successive pass providing real-time feedback to robot conditions.
- Ballistic behaviors, on the other hand, must "remember" what they were trying to accomplish in order to complete the operation.

Ballistic Behavior (Cont'd)

- BUMP ESCAPE is a great example of the ballistic behavior. When the robot collides with an object, BUMP ESCAPE is triggered. The robot backs up for a time, spins, positions forward, and returns to its start state for the next collision.



Ballistic Behavior (Cont'd)

- Complexity in implementing ballistic behaviors stems from maintaining state between loops while still keeping the robot on its mission. Here are a few examples to consider:
  - During the backup state above, a higher order behavior is triggered. It takes control of the robot and then releases control. If the BUMP ESCAPE is triggered again what state should it start in?
  - What if another behavior is triggered in the middle of an escape? How will the robot respond so that it does not crash again or worse, oscillate between a collision and another behavior's action.

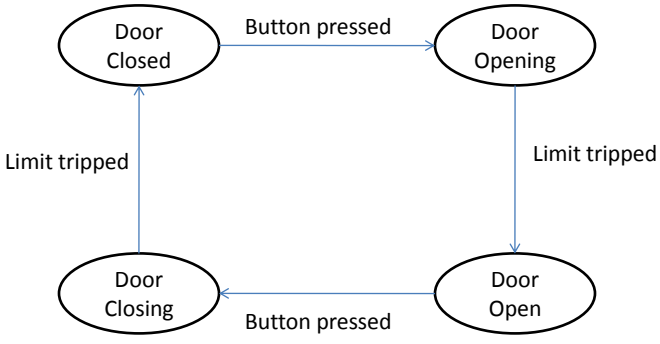
Finite State Machine

- Pure servo behaviors live in the moment.
- Every time such a behavior is called, it computes what to do right now.
- The behavior pays no attention to what it did the last time it was called and it makes no preparation for what it will do next time.
- We say that such behaviors have no state.
- Finite State Machine (FSM) is a type of system that has a limited number of states and has well defined rules specifying how the system is allowed to transition from one state to another.

Garage System

- If the door is closed and I press the open/close button, the door begins moving up.
- When it is all the way open, the door trips a limit switch and motion stops.
- If I push the button when the door is open, it begins moving down.
- As the door reaches the bottom of its travel, it trips another limit switch such that motion stops just when the door is fully closed.

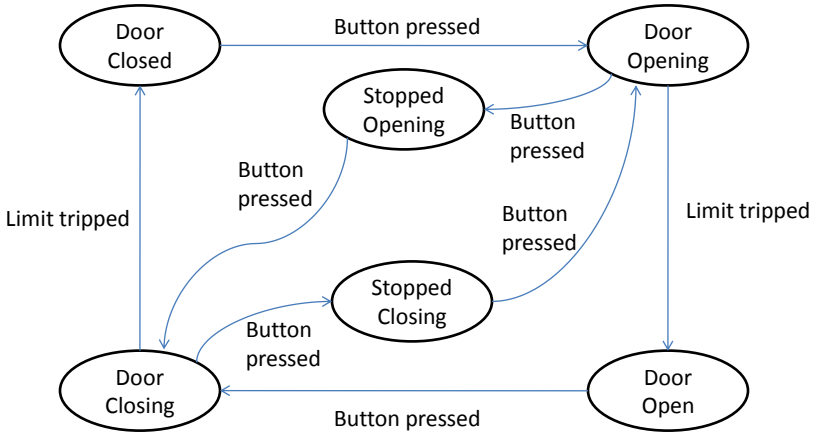
Garage System (Cont'd)



Garage System (Cont'd)

- I don't have to wait until the door reaches its fully up or down position before I press the button again.
- If I push the open/close button while the door is moving, the door halts immediately.
- Pressing the button again from this state makes the door move in the direction opposite to the way it had been going.

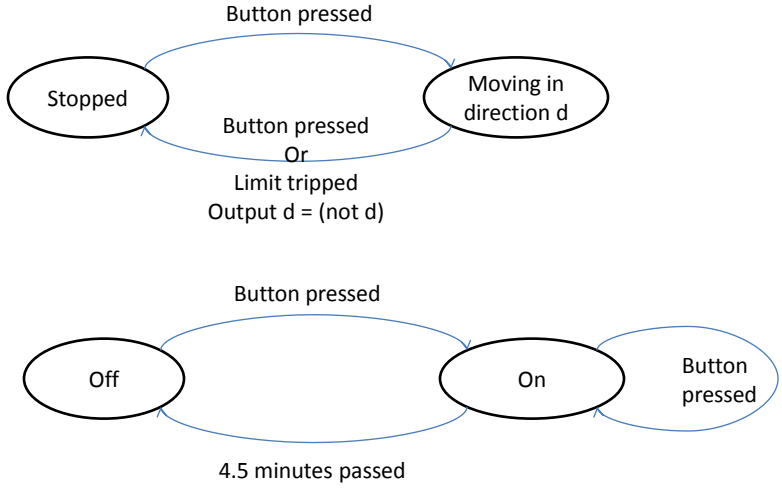
Garage System (Cont'd)



Garage System (Cont'd)

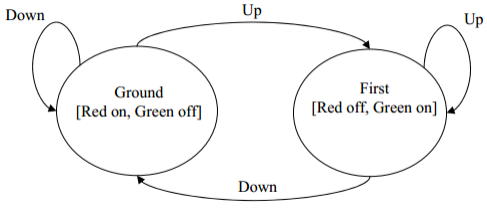
- The garage door opener includes a courtesy light that comes on automatically as soon as I press the open/close button.
- The manufacturer's intent is to give the user time to get into or out of the car at night when, without the courtesy light, the garage would be dark.
- After pressing the open/close button, the light goes on and remains on until 4.5 minutes have passed – then a relay clicks and the light blinks off.
- Further modifications in the FSM would compromise on its readibility.

Simplified FSM

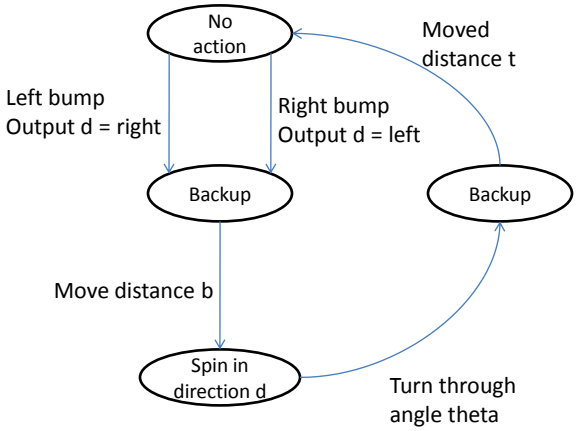


Elevator

- In this example, we'll be designing a controller for an elevator. The elevator can be at one of two floors: Ground or First.
- There is one button that controls the elevator, and it has two values: Up or Down.
- Also, there are two lights in the elevator that indicate the current floor: Red for Ground, and Green for First.
- At each time step, the controller checks the current floor and current input, changes floors and lights in the obvious way.



FSM Example: Escape



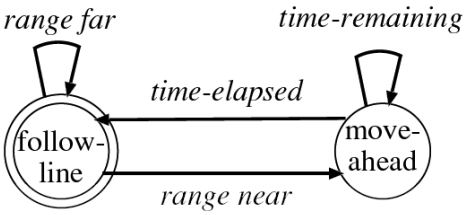
FSM for Sumo Wrestling

SURVIVE	The robot enters this state when it detects the dohyo (ring) edge. Its goal is to survive by not going off the ring. The robot rotates away from the sensor that sensed the edge to face back towards the center of the ring.	1. Rotate away from the line sensor that sensed the edge. 2. Switch to HUNT state when rotation is complete.
HUNT	The robot is not at the edge of the ring but hasn't sensed the opponent. The robot moves around in an arcing pattern so its range sensors will sweep across the ring in hopes of sensing the opponent	1. Switch to SURVIVE state if ring edge detected. 2. Switch to TARGET state if range sensors indicate an object ahead. 3. Otherwise, drive in an arc by applying more power to one wheel than the other

FSM for Sumo Wrestling (Cont'd)

TARGET	The opponent has been sensed ahead. Aim the robot to face the opponent directly.	1. Switch to SURVIVE state if ring edge detected. 2. If still sensing opponent, but opponent is not directly ahead turn slightly to aim at opponent. 3. If opponent is directly ahead and close switch to ATTACK state. 4. If opponent is ahead but not close, move straight forward. 5. Otherwise, switch to HUNT state.
ATTACK	The opponent has been found and aiming is complete. Drive straight ahead at full power to push the opponent off the ring.	1. Switch to SURVIVE state if ring edge detected. 2. Otherwise, drive straight forward at full power.

FSM Formal Notations



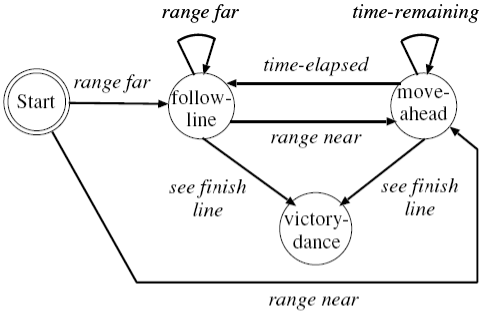
a.

$M : K = \{\text{follow-line, move-ahead}\}, \Sigma = \{\text{range near, range far}\},$
$s = \text{follow-line}, F = \{\text{follow-line, move-ahead}\}$

q	σ	$\delta(q, \sigma)$
follow-line	range near	move-ahead
follow-line	range far	follow-line
move-ahead	time remaining	move-ahead
move-ahead	time elapsed	follow-line

b.

FSM Formal Notations (Cont'd)



a.

$M: K = \{\text{follow-line, move-ahead}\}, \Sigma = \{\text{range near, range far}\},$
$s = \text{follow-line}, F = \{\text{follow-line,move-ahead}\}$

q	σ	$\delta(q, \sigma)$
follow-line	range near	move-ahead
follow-line	range far	follow-line
move-ahead	time remaining	move-ahead
move-ahead	time elapsed	follow-line

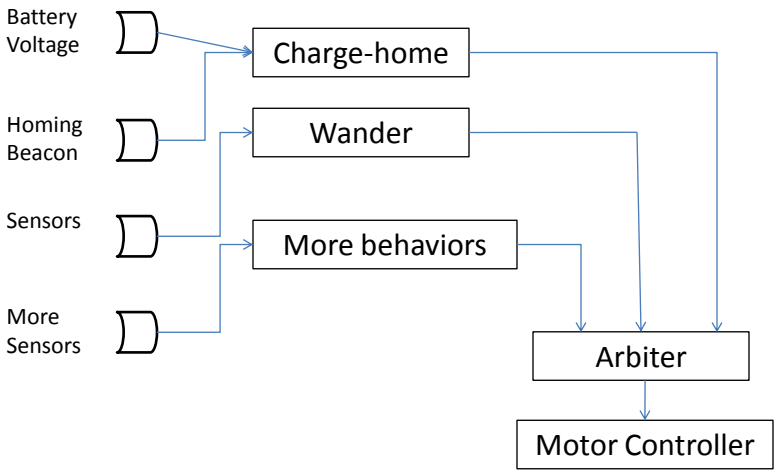
Exercises

- FSM diagram of a flush toilet.
- FSM diagram of a washing machine.

Arbitration

- A primitive behavior consists of a control system that makes the robot act in a certain way and a trigger to decide when it is appropriate to take such actions.
- As long as only one behavior triggers at a time, things work smoothly.
- But what ensues when two or more behaviors happened to trigger at the same time, each one wanting the robot to take a different action?

Arbiter



Fixed-Priority Arbiter

- The most commonly used arbiter is fixed-priority arbiter.
- Fixed-priority means that the programmer has decided in advance which behavior ought to win any time a conflict occurs.
- Each connection from a behavior to the arbiter is given a unique priority.

Graceful Degradation

- Things never go smoothly for robots operating in the real world. In particular
  - A command intended to direct the robot to move in a particular way instead, because of uncontrollable environmental effects, causes the robot to move in a different way.
  - The robot's program makes an assumption about the world that turns out not to be true.
  - The robot's sensors fail outright or produce false negative or false positive results.

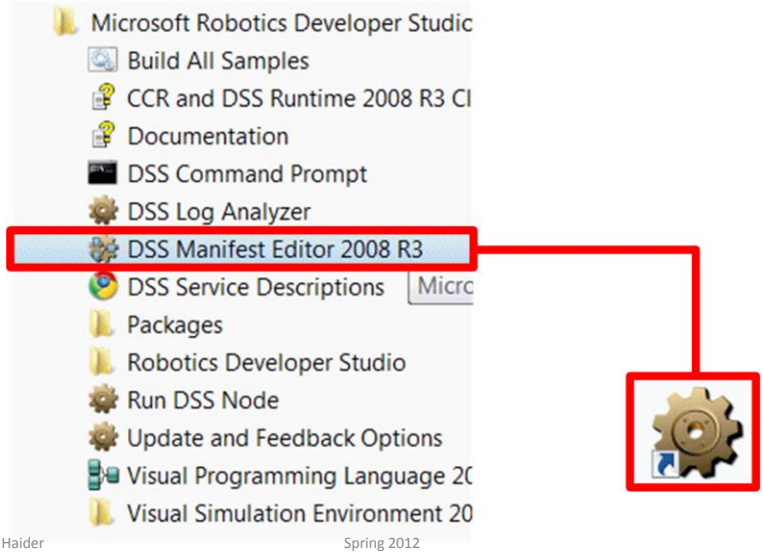
Graceful Degradation (Cont'd)

- Note:
  - A false negative means that the sensor did not react when it should have.
  - A false positive occurs when a sensor reports a condition that does not exist.
- Regardless of all this, we want our robots to soldier on.
- The ability of a system to continue to perform at a reduced level in the presence of subsystem errors and failures is known as *graceful degradation*.

Manifest Building in MRDS

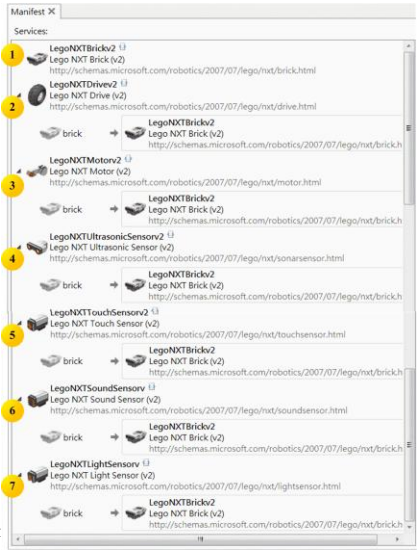
Material taken from "Robot Development using Microsoft Robotics Developer Studio" book by Kang, Chang, Gu and Chi.

Manifest Editor

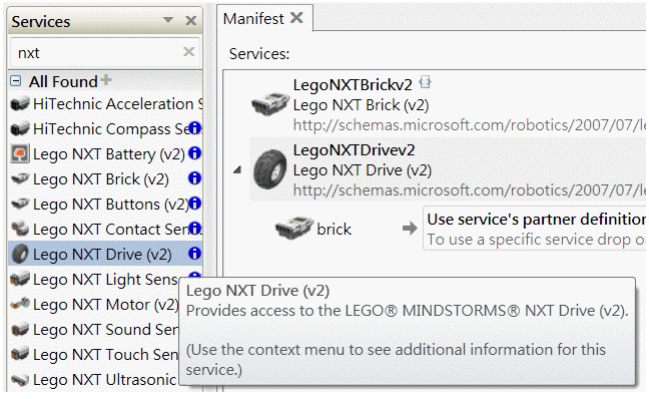


Building Your Robot's Manifest

- You can specify the actuators and sensors of your robots in Manifest Editor.
- As is the case with VPL, Manifest Editor also has drag and drop functionality.

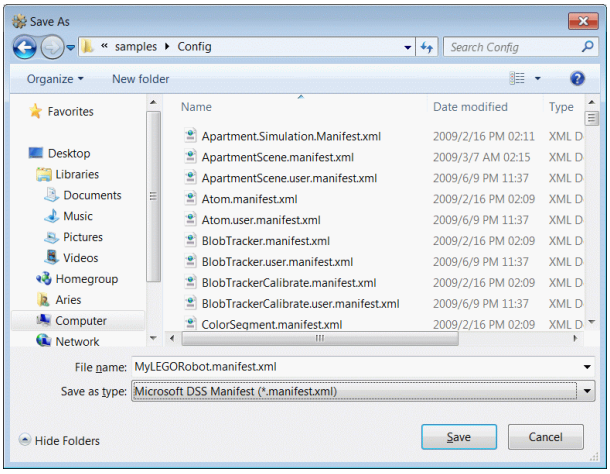


Manifest Editor (Cont'd)



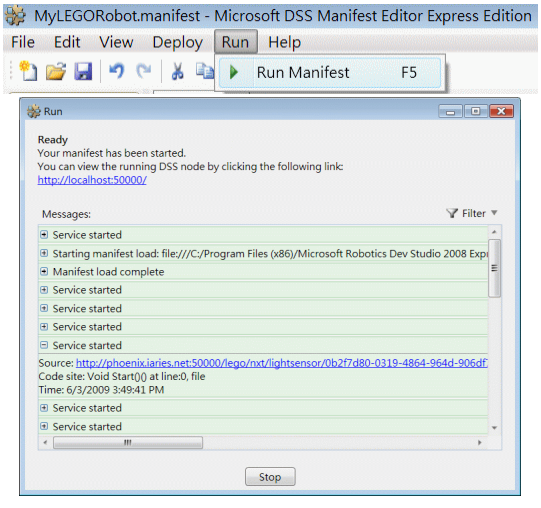
Saving Your Manifest

- Once done, you can save the manifest by clicking on the Save button.



Running Your Manifest

- You can compile and test your manifest by clicking on the Run button



Exercises

- Repeat the following exercises you did with simulated robot
  - Controlling robot movements using game controller
  - Making robot move backward if bumper sensor is pressed
  - Make robot spin if enters white area
- Write escape behavior
  - Back, Turn and then move forward
  - Submit a report on how you implemented this behavior (timer, any logical operator, etc.)
- Check the ballistic behavior of MRDS by writing different behaviors that may access the robot at the same
  - Submit a report describing your observations.