Control of Time-Varying Behavior

Proportional-Derivative (PD) controller: react to the immediate sensory inputs
• E.g.: yaw control
• Need a reference (or “desired”) heading

Where does this reference come from?
Control of Time-Varying Behavior

Where does the reference come from?

• Determined by what our task is (or subtask)

• E.g.: at the current state of a mission, it may be appropriate to orient the craft in a particular direction so that it can fly back “home”
Control of Time-Varying Behavior

Can often express a “mission” in terms of a sequence of sub-tasks (or a plan)
• But: we also want to handle contingencies when they arrive

Finite state machines are a simple way of expressing such plans and contingencies
Finite State Machines (FSMs)

Pure FSM form is composed of:

- A set of states
- A set of possible inputs (or events)
- A set of possible outputs (or actions)
- A transition function:
  - Given the current state and an input: defines the output and the next state
Finite State Machines (FSMs)

States:

- Represent all possible “situations” that must be distinguished
- At any given time, the system is in exactly one of the states
- There is a finite number of these states
Finite State Machines (FSMs)

An example: our synchronous counter
• States: ?
Finite State Machines (FSMs)

An example: our synchronous counter

• States: the different combinations of the digits: 000, 001, 010, … 111

• Inputs: ?
Finite State Machines (FSMs)

An example: our synchronous counter

• Inputs:
  – Really only one: the event associated with the clock transitioning from high to low
  – We will call this “C”

• Outputs: ?
Finite State Machines (FSMs)

An example: our synchronous counter

• Outputs: same as the set of states

• Transition function: ?
Finite State Machines (FSMs)

An example: our synchronous counter

• Transition function:
  – On the clock event, transition to the next state in the sequence
FSM Example: Synchronous Counter

A Graphical Representation:

A set of states
FSM Example:
Synchronous Counter

A transition

C/001 → 001

States:
- 000
- 001
- 010
- 011
- 100
- 110
- 111
- 101
FSM Example:
Synchronous Counter

A transition

The event
FSM Example: Synchronous Counter

A transition

The output
FSM Example: Synchronous Counter

The next transition
FSM Example: Synchronous Counter

The next transition

- From state 000, transition to state 001 on input C/001.
- From state 001, transition to state 010 on input C/010.
- From state 010, transition to state 011 on input C/011.
- From state 100, transition to state 101 on input C/100.
- From state 110, transition to state 111 on input C/110.
FSM Example: Synchronous Counter

The full transition set

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
FSM Example: Synchronous Counter

Initial condition

000 → 001 → 010 → 011 → 100 → 101 → 110 → 111 → 000

x/000 → C/000
C/001 → 001
C/010 → 010
C/011 → 011
C/101 → 100
C/110 → 110
C/111 → 111
Example II: An Up/Down Counter

Suppose we have two events (instead of one): Count up and count down
• How does this change our state transition diagram?
Example II: An Up/Down Counter

From state 000, there are now two possible transitions

- U/001 from state 000 to state 001
- D/111 from state 000 to state 111
Example II: An Up/Down Counter

Likewise for state 001…
Example II: An Up/Down Counter

The full transition set

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
FSMs and Control

How do we relate FSMs to Control?
• States are ?
FSMs and Control

How do we relate FSMs to Control?
• States are our memory of recent inputs
• Inputs are?
FSMs and Control

How do we relate FSMs to Control?
• States are our memory of recent inputs
  • Inputs are some processed representation of what the sensors are observing
  • Outputs are ?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs are some processed representation of what the sensors are observing

• Outputs are the control actions
FSMs: A Control Example

Suppose we have a vending machine:

- Accepts dimes and nickels
- Will dispense one of two things once $.20 has been entered: Jolt or Buzz Water
  - The “user” requests one of these by pressing a button
- Ignores select if < $.20 has been entered
- Immediately returns any coins above $.20
Vending Machine FSM

What are the states?
Vending Machine FSM

What are the states?

- $0
- $.05
- $.10
- $.15
- $.20
Vending Machine FSM

What are the inputs/events?
Vending Machine FSM

What are the inputs/events?

• Input nickel (N)
• Input dime (D)
• Select Jolt (J)
• Select Buzz Water (BW)
Vending Machine FSM

What are the outputs?
Vending Machine FSM

What are the outputs?
• Return nickel (RN)
• Return dime (RD)
• Dispense Jolt (DJ)
• Dispense Buzz Water (DBW)
• Nothing (Z)
Vending Machine Design

What is the initial state?
Vending Machine Design

What is the initial state?

- $S = 0$
Vending Machine Design

What can happen from $S = $0?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

What can happen from $S = $0? 

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.05</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$.10</td>
<td>Z</td>
</tr>
<tr>
<td>J</td>
<td>$0</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$0</td>
<td>Z</td>
</tr>
</tbody>
</table>
Vending Machine Design

A piece of the state diagram:

- $0
- $.05
- $.10

States:
- N/Z
- D/Z
- J/Z
- BW/Z
Vending Machine Design

What can happen from $S = $0.05?  

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

What can happen from $S = \$0.05$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$$.10</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$$.15</td>
<td>Z</td>
</tr>
<tr>
<td>J</td>
<td>$$.05</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$$.05</td>
<td>Z</td>
</tr>
</tbody>
</table>

What does the modified diagram look like?
Vending Machine Design

A piece of the state diagram:
Vending Machine Design

What can happen from $S = \$0.10$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

What can happen from $S = \$0.10$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.15</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$.20</td>
<td>Z</td>
</tr>
<tr>
<td>J</td>
<td>$.10</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$.10</td>
<td>Z</td>
</tr>
</tbody>
</table>
Vending Machine Design

A piece of the state diagram:
Vending Machine Design

What can happen from $S = $0.15?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vending Machine Design

What can happen from $S = $0.15?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.20</td>
<td>Z</td>
</tr>
<tr>
<td>D</td>
<td>$.20</td>
<td>RN</td>
</tr>
<tr>
<td>J</td>
<td>$.15</td>
<td>Z</td>
</tr>
<tr>
<td>BW</td>
<td>$.15</td>
<td>Z</td>
</tr>
</tbody>
</table>
Vending Machine Design

A piece of the state diagram:
Vending Machine Design

Finally: what can happen from $S = 0.20$?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Finally, what can happen from $S = $0.20?

<table>
<thead>
<tr>
<th>Event</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>$.20</td>
<td>RN</td>
</tr>
<tr>
<td>D</td>
<td>$.20</td>
<td>RD</td>
</tr>
<tr>
<td>J</td>
<td>$0</td>
<td>DJ</td>
</tr>
<tr>
<td>BW</td>
<td>$0</td>
<td>DBW</td>
</tr>
</tbody>
</table>
Vending Machine Design

The complete state diagram:
FSMs and Control

How do we relate FSMs to Control?
• States are ?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs are ?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs are some processed representation of what the sensors are observing

• Outputs are ?
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

• Inputs are some processed representation of what the sensors are observing

• Outputs are the control actions
A Robot Control Example

Consider the following task:

- The robot is to move toward the first beacon that it “sees”
- The robot searches for a beacon in the following order: right, left, front

What is the FSM representation?
Robot Control Example II

Consider the following task:

- The robot must lift off to some altitude
- Translate to some location
- Take pictures
- Return to base
- Land
- At any time: a detected failure should cause the craft to land

What is the FSM representation?
FSMs As Controllers

• Need code that translates sensory inputs into FSM events
• An FSM output can require an arbitrary amount of time
  – We will often implement this control action as a separate function call
• Control actions will not necessarily be fixed (but could be a function of sensory input)
FSMs As Controllers (cont)

• We might choose to leave some events out of the implementation
  – Only some events may be relevant to certain states

• When in a state, the FSM may also issue control actions (even when a new event has not arrived)
  – Again, this may be implemented as a function call
int state = 0;  // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
    case 0:
        <handle state 0>
        break;
    case 1:
        <handle state 1>
        break;
    case 2: ...
    }
}
int state = 0; // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
    case 0:
      <handle state 0>
      break;
    case 1:
      <handle state 1>
      break;
    case 2: ...
  }
}
FSMs in C

```c
int state = 0; // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        case 1:
            <handle state 1>
            break;
        case 2: ...
    }
}
```

Loop forever
FSMs in C

```c
int state = 0;   // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        case 1:
            <handle state 1>
            break;
        case 2: ...
    }
}
```

“pseudo code”: not really code, but indicates what is to be done
FSMs in C

```c
int state = 0;  // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        case 1:
            <handle state 1>
            break;
        case 2: ...
    }
}
```

In this case: we will translate the current sensory inputs into a representation of an event (if one has happened)
FSMs in C

```c
int state = 0; // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        case 1:
            <handle state 1>
            break;
        case 2: ...
    }
}
```

Switch/case syntax allows us to cleanly perform many "if(x==y)" operations.
FSMs in C

```c
int state = 0; // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
    case 0:
        <handle state 0>
        break;
    case 1:
        <handle state 1>
        break;
    case 2: ... 
    }
}
```

If state==0, then execute the following code.
FSMs in C

int state = 0; // Initial state
while(1) {
    // do some processing of the sensory inputs
    switch(state) {
        case 0:
            // handle state 0
            break;
        case 1:
            // handle state 1
            break;
        case 2: ...
    }
}
int state = 0;  // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        case 1:
            <handle state 1>
            break;
        case 2: ...
    }
}
FSMs in C

```c
int state = 0;  // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        case 1:
            <handle state 1>
            break;
        case 2: ...
    }
}
```

If state==1, then ...
int state = 0; // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        case 1:
            <handle state 1>
            break;
        case 2: ...
    }
}
FSMs in C (some other possibilities)

```c
int state = 0;   // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        :
        default:
            <handle default case>
            break;
    }
    <do some low-level control>
}
```
int state = 0; // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        <default:
            <handle default case>
            break;
        }
    <do some low-level control>
}
FSMs in C (some other possibilities)

```c
int state = 0; // Initial state
while(1) {
    <do some processing of the sensory inputs>
    switch(state) {
        case 0:
            <handle state 0>
            break;
        :
        default:
            <handle default case>
            break;
    }
    <do some low-level control>
}
```

(possibly) alter some control outputs (e.g., steering direction)
Handling Each State

• You will need to provide code that handles the event processing for each state

• Specifically:
  – You need to handle each event that can occur
  – For each event, you must specify:
    • What action is to be taken
    • What the next state is
Handling Each State

In our vending machine example:

- Events are easy to describe (only a few things can happen)
- It is convenient in this case to also “switch” on the event
FSMs in C: Processing for Individual States

case STATE_10cents:
    // $.10 has already been deposited
    switch(event) {
        case EVENT_NICKEL:    // Nickel
            state = STATE_15cents;  // Transition to $.15
            break;
        case EVENT_DIME:      // Dime
            state = STATE_20cents;  // Transition to $.2
            break;
        case EVENT_JOLT:      // Select Jolt
        case EVENT_BUZZ:      // Select Buzzwater
            display_NOT_ENOUGH();
            break;

        case EVENT_NONE:      // No event
            break;                 // Do nothing
    }
    break;

break;
FSMs in C: Processing for Individual States

case STATE_10cents:
  // $.10 has already been deposited
  switch(event) {
    case EVENT_NICKEL:  // Nickel
      state = STATE_15cents;  // Transition to $.15
      break;
    case EVENT_DIME:  // Dime
      state = STATE_20cents;  // Transition to $.2
      break;
    case EVENT_JOLT:  // Select Jolt
    case EVENT_BUZZ:  // Select Buzzwater
      display_NOT_ENOUGH();
      break;
    case EVENT_NONE:  // No event
      break;  // Do nothing
  }
  break;

Another integer
FSMs in C: Processing for Individual States

case STATE_10cents:
    // $.10 has already been deposited
    switch(event) {
        case EVENT_NICKEL:   // Nickel
            state = STATE_15cents;  // Transition to $.15
            break;
        case EVENT_DIME:   // Dime
            state = STATE_20cents;  // Transition to $.2
            break;
        case EVENT_JOLT:   // Select Jolt
            display_NOT_ENOUGH();
            break;
        case EVENT_BUZZ:   // Select Buzzwater
            display_NOT_ENOUGH();
            break;
        case EVENT_NONE:   // No event
            break;  // Do nothing
    }
    break;  // Do nothing
FSMs in C: Processing for Individual States

```c
case STATE_10cents:
    // $.10 has already been deposited
    switch(event) {
        case EVENT_NICKEL:  // Nickel
            state = STATE_15cents; // Transition to $.15
            break;
        case EVENT_DIME:     // Dime
            state = STATE_20cents; // Transition to $.2
            break;
        case EVENT_JOLT:     // Select Jolt
        case EVENT_BUZZ:     // Select Buzzwater
            display_NOT_ENOUGH();
            break;
        case EVENT_NONE:     // No event
            break;    // Do nothing
    }
    break;
```
FSMs in C: Processing for Individual States

case STATE_10cents:
    // $.10 has already been deposited
    switch(event) {
        case EVENT_NICKEL: // Nickel
            state = STATE_15cents; // Transition to $.15
            break;
        case EVENT_DIME: // Dime
            state = STATE_20cents; // Transition to $.2
            break;
        case EVENT_JOLT: // Select Jolt
        case EVENT_BUZZ: // Select Buzzwater
            display_NOT_ENOUGH();
            break;
        case EVENT_NONE: // No event
            break; // Do nothing
    };
break;

If any of these match, then execute the following code (which does nothing in this example)
Handling Each State

Some events do not fall neatly into one of several categories

• This precludes the use of the “switch” construct

• For example: an event that occurs when our hovercraft reaches a goal orientation

• For these continuous situations, we typically use an “if” construct …
FSMs in C: Processing for Individual States

```c

case STATE_MISSION_PHASE_3:
    if (heading_error < 100 &&
        heading_error > -100)
    {
        // Accelerate forward!
        duty_forward = 126;
        state = STATE_MISSION_PHASE_4;
    }
    break;

```
FSMs in C: Processing for Individual States

```c
: case STATE_MISSION_PHASE_4:
    if(distance_left < 200 &&
        distance_right < 200)
    {
        // Brake!
        duty_forward = 0;
        duty_middle = 127;
        middle_thrust_dir(0);
        state = STATE_MISSION_PHASE_5;
        counter = 0;  // Reset the clock
    }
    break;
:
```
FSMs in C: Processing for Individual States

: case STATE_MISSION_PHASE_5:
    if (counter > 20) {
        // One second has gone by since we started the brake: Stop the brake
        duty_middle = 0;
        state = STATE_MISSION_PHASE_6;
    }
    break;
:

NOTE: clock is being incremented once per control cycle
A Note on “Style” in C

• The numbers that we assigned to the different states are arbitrary (and at first glance, hard to interpret)
• Instead, we can define constant strings that have some meaning

• Replace: 0, 1, 2, 3, 4, 5
• With: `STATE_00`, `STATE_05`, `STATE_10`, `STATE_15`, `STATE_20`
A Note on “Style” in C

In C, this is done by adding some definitions to the beginning of your program (either in the .c file or the .h file):

```
#define STATE_00cents 0
#define STATE_05cents 1
#define STATE_10cents 2
#define STATE_15cents 3
#define STATE_20cents 4
```