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 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: a 3-bit counter that increments when “count” input is received

• States: ?
Finite State Machines (FSMs)

An example: a counter

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

- Inputs: ?
Finite State Machines (FSMs)

An example: a counter

• Inputs (events):
  – Only one: “count”
  – We will call this “C”

• Outputs: ?
Finite State Machines (FSMs)

An example: a counter

• Outputs: same as the set of states

• Transition function: ?
Finite State Machines (FSMs)

An example: a counter

- **Transition function:**
  - On the count event, transition to the next highest value
FSM Example: Synchronous Counter

A Graphical Representation:

A set of states
FSM Example: Synchronous Counter

A transition

C/001

000

001

010

011

100

101

110

111
 FSM Example: Synchronous Counter

A transition

The event
FSM Example: Synchronous Counter

A transition

The output
FSM Example: Synchronous Counter

A transition

The output: The Zyante book calls these “Mealy Actions”
FSM Example: Synchronous Counter

The next transition

![Diagram of Synchronous Counter FSM]
FSM Example: Synchronous Counter

The next transition
FSM Example: Synchronous Counter

The full transition set
FSM Example:
Synchronous Counter

Initial condition

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

x/000

C/000 → C/001 → C/010 → C/011 → C/100 → C/101 → C/110 → C/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

- From state 000, an up transition (U) leads to state 001.
- From state 000, a down transition (D) leads to state 111.

States:
- 000
- 001
- 010
- 011
- 100
- 101
- 110
- 111
Example II: An Up/Down Counter

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

The full transition set

Diagram: Connections between states with transitions labeled as follows:

- U/x represents an up transition
- D/x represents a down transition

States:
- 000
- 001
- 010
- 011
- 100
- 101
- 110
- 111

Transitions:
- U/000 to U/001
- U/001 to U/010
- U/010 to U/011
- U/011 to U/100
- U/100 to U/101
- U/101 to U/111
- U/111 to U/110
- U/110 to D/101
- D/101 to D/100
- D/100 to D/011
- D/011 to D/010
- D/010 to D/001
- D/001 to D/000
- D/000 to U/000

This diagram illustrates the full transition set for an up/down counter.
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
  – These are typically “high level” actions: e.g., set the goal orientation to 125 degrees
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>
</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:
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>
<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?

What does the modified diagram look like?

<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>
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>
</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:

- States: $0, $.05, $.10, $.15, $.20
- Transitions:
  - $0 to $.05: x/Z, N/Z, D/RN
  - $.05 to $.10: J/Z, BW/Z, D/Z
  - $.10 to $.15: N/Z, D/Z
  - $.15 to $.20: J/Z, BW/Z
  - $.20 to $.10: N/Z, D/RN
  - $.10 to $.05: J/Z, BW/Z
  - $.05 to $.10: N/Z
  - $.15 to $.20: N/Z, D/RN
  - $.20 to $.15: J/Z, BW/Z

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
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>
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>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:
• End for day…
FSM Design Pattern

- The system is always in exactly one state
- Think of transitions as happening instantaneously
FSM Design Pattern

Think of transitions as happening instantaneously

• FSM actions are also instantaneous

• For an activity that must take a finite amount of time:
  – The FSM action is to initiate the activity
  – The next state is one in which the system is waiting for activity completion
  – The next event signals completion
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
• Once beacon is found, move toward it and stop once the beacon is reached

What is the FSM representation?
Robot Description

Mobile robot with sensor turret on top

- Mobile robot turns take time
- Turret turns are relative to the mobile base and do not take time
Events

- Robot Turn Complete (TC)
- Beacon (B)
- No Beacon (NB)
Actions

- Look left (LL): turn turret to be facing left (relative to the mobile base)
- Look right (LR)
- Look forward (LF)
- Turn left (TL): initiate a turn of the robot base by 90 degrees to the left
- Turn right (TR): initiate right turn
- Move forward (F): initiate forward movement
- Stop (S)
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?
Vending Machine FSM
FSMs and Control

How do we relate FSMs to Control?

• States are our memory of recent inputs

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

• Outputs are the control actions
FSMs in C

Implementation in the Arduino environment

```c
void loop()
{
    fsm_step();  // Evaluate the FSM
}
```
FSMs in C

```c
fsm_step() {
    static State state = STATE_0;  // Initial state

    <do some processing of the sensory inputs>
    switch(state) {
        case STATE_0:
            <handle state 0>
            break;
        case STATE_1:
            <handle state 1>
            break;
        case STATE_2: ...
    }
}
```
Creating an Enumerated Variable Type

• Definition:

typedef enum {
    STATE_0, STATE_1, STATE_2
} State;

• Use:

State s = STATE_1;

s can only take on these 3 values
Locally Defined Variables

• Local variables defined inside of a function are allocated to memory only when the function is called
  – Memory region called **the stack**
• When the function returns, the memory is reclaimed for use by other functions
Static Variables

Declaring a variable inside a function as static:

```c
static State state = STATE_0;  // Initial state
```

• The variable acts like a global variable:
  – The memory continues to exist after a return from the function
  – This means that the value from the last call to the function can be used in the next call
  – But: the variable can only be “seen” by this function
Static Variables

Declaring a variable inside a function as static:

```c
static State state = STATE_0;  // Initial state
```

- Other key thing to remember: the assignment is executed exactly once (before the main() function is executed)
- We can use this to set the initial value of the static variable
FSMs in C

fsm_step() {
    static State state = STATE_0;  // Initial state

    <do some processing of the sensory inputs>
    switch(state) {
        case STATE_0:
            <handle state 0>
            break;
        case STATE_1:
            <handle state 1>
            break;
        case STATE_2: ...
    }
}
fsm_step() {
    static State state = STATE_0; // Initial state

    <do some processing of the sensory inputs>
    switch(state) {
        case STATE_0:
            <handle state 0>
            break;
        case STATE_1:
            <handle state 1>
            break;
        case STATE_2: ...
    }

    <do some low-level control>
}
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

```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;

};
```

Andrew H. Fagg: Embedded Real-Time Systems: FSMs
Handling Each State

Some events do not fall neatly into one of several categories

• This precludes the use of the “switch” construct for events

• 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

```c
fsm_step() {
    static State state = STATE_0; // Initial state

    <do some processing of the sensory inputs>
    switch(state) {
        case STATE_MISSION_PHASE_3:
            <handle phase 3>
            break;
        case STATE_MISSION_PHASE_4:
            <handle phase 4>
            break;
        case STATE_MISSION_PHASE_5:
            :
    }
}
```
FSMs in C: Processing for Individual States

case STATE_MISSION_PHASE_3:
    if(heading_error < 10.0 &&
       heading_error > -10.0)
        {
            // Accelerate forward!
            desired_velocity = .2;  // Action
            state = STATE_MISSION_PHASE_4;
        }
    break;

   :
FSMs in C: Processing for Individual States

case STATE_MISSION_PHASE_4:
    if (distance_left < 20.0 || distance_right < 20.0)
    {
        // Brake!
        desired_velocity = 0;
        counter = 0;    // Reset the clock
        state = STATE_MISSION_PHASE_5;
    };
    break;
:
FSMs in C

New tweak: `fsm_step()` is called by `loop()` once per 50 ms (we will discuss the mechanism in the coming weeks)

```c
fsm_step() {
    static State state = STATE_0; // Initial state
    static int counter = 0;
    counter++;

    switch(state) {
        case STATE_MISSION_PHASE_3:
            <handle phase 3>
            break;
        case STATE_MISSION_PHASE_4:
            <handle phase 4>
            break;
        case STATE_MISSION_PHASE_5:
            :
    }
}
```
FSMs in C: Processing for Individual States

: 

```
case STATE_MISSION_PHASE_4:  
    if(distance_left < 20.0 ||
        distance_right < 20.0)  
    {  
        // Brake!
        desired_velocity = 0;
        counter = 0;    // Reset the clock
        state = STATE_MISSION_PHASE_5;
    };  
    break;  
:  
```
FSMs in C: Processing for Individual States

```c
: case STATE_MISSION_PHASE_5:
    if(counter > 20)
    {
        // A fixed amount of time has gone by
        heading_goal = heading_goal - 90.0;
        if(heading_goal <= -180.0) heading_goal += 360;
        state = STATE_MISSION_PHASE_6;
    }
    break;
: How much time has gone by?
```
FSMs in C: Processing for Individual States

```c
case STATE_MISSION_PHASE_5:
    if(counter > 20)
    {
        // A fixed amount of time has gone by
        heading_goal = heading_goal - 90.0;
        if(heading_goal <= -180.0) heading_goal += 360;
        state = STATE_MISSION_PHASE_6;
    }
    break;
```

How much time has gone by?   1 sec
FSM Implementation Notes

• FSM code should not contain delays or waits
  – No delay_ms() or while(…){}
  – Remember that your FSM code will be called once per control cycle: use “if” to check for an event during that control cycle

• Use LEDs and/or fprintf() to indicate current state

• Implement and test incrementally
FSM Implementation Notes

For your future projects: you will use an enumerated data type to represent your set of states.

• Allows us to be very clear what the possible values are
• Affords type checking by the compiler