What is a Finite State Machine?

And why should I care?

What does Wikipedia say?

A Finite State Machine is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of *states*. The machine is in only one state at a time; the state it is in at any given time is called the *current state*. It can change from one state to another when initiated by a triggering event or condition; this is called a *transition*. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

Dissecting the Definition

A Finite State Machine is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

Mathematical model. (Way too high brow)

Used to design/write computer programs. (Sounds promising)

Used to design/write sequential logic circuits. (FPGA code)

Dissecting the Definition

A Finite State Machine is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

Abstract machine is a fancy way to say ‘programming tool’

Finite is better than infinite! We have a chance to code a finite number of things.

In only one state at a time. No quantum physics here!

Can change from one state to another. Would not be much use if it was stuck!

Dissecting the Definition

A Finite State Machine is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

Transition; What has to happen to go from one state to another.

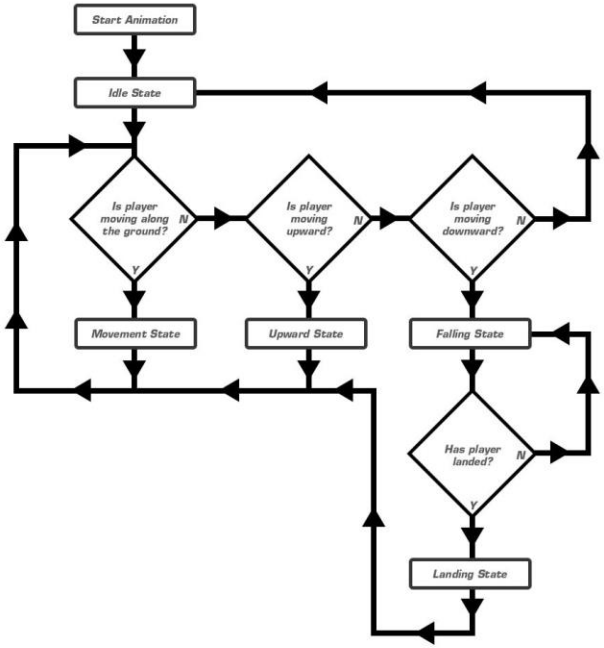
An FSM is defined by its states and transitions. Clear as mud.

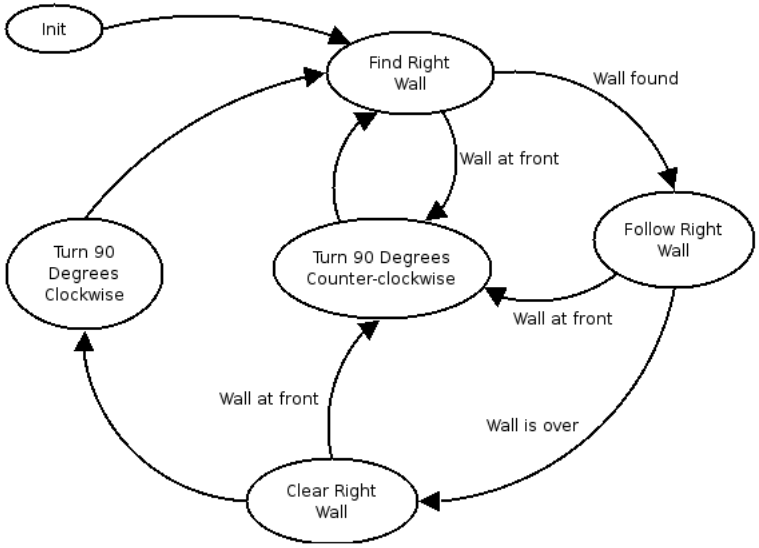
Simpler Definition

A Finite State Machine is a concept/tool that can aid in writing computer programs or FPGA logic. It is a tool that can be in only one state at a time (the current state). It can change from one state to another when some triggering event or transition occurs. You execute your code on each transition.

Simplest definition

- A Finite State Machine is a way to implement in software some types of flowcharts.





An traditional example

You need to write a routine that will read in a series of characters looking for a legal floating point number. Some good examples are:

1.0, +2.0, -3.0, +0.004, -0.0005, +1e6, -2.E03, -4e-005

Some bad examples are:

., -, a.3, 3.a, 3..0, q3.4, +-1, E3, 1.2E3.2

The processing of bytes stops when

1) You run out of characters in the string
2) You find a character that does not match a number.

How to you start to write this?

If this is the first character and it is not legal, return an error

if (nChar == 1) and (c==‘+’) or (c==‘-’) or ((c>=’0’) and (c<=‘9’)) or (c==‘.’)

If the first character is a + sign then the whole number is positive

if (nChar==1) and (c==‘+’) sign = +1

If the first character is a - sign then the whole number is negative

if (nChar==1) and (c==‘-’) sign = -1

If the first character is a ‘.’ than the number is positive and we need to start a fraction

if (nChar==1) and (c==‘.’) fraction = true

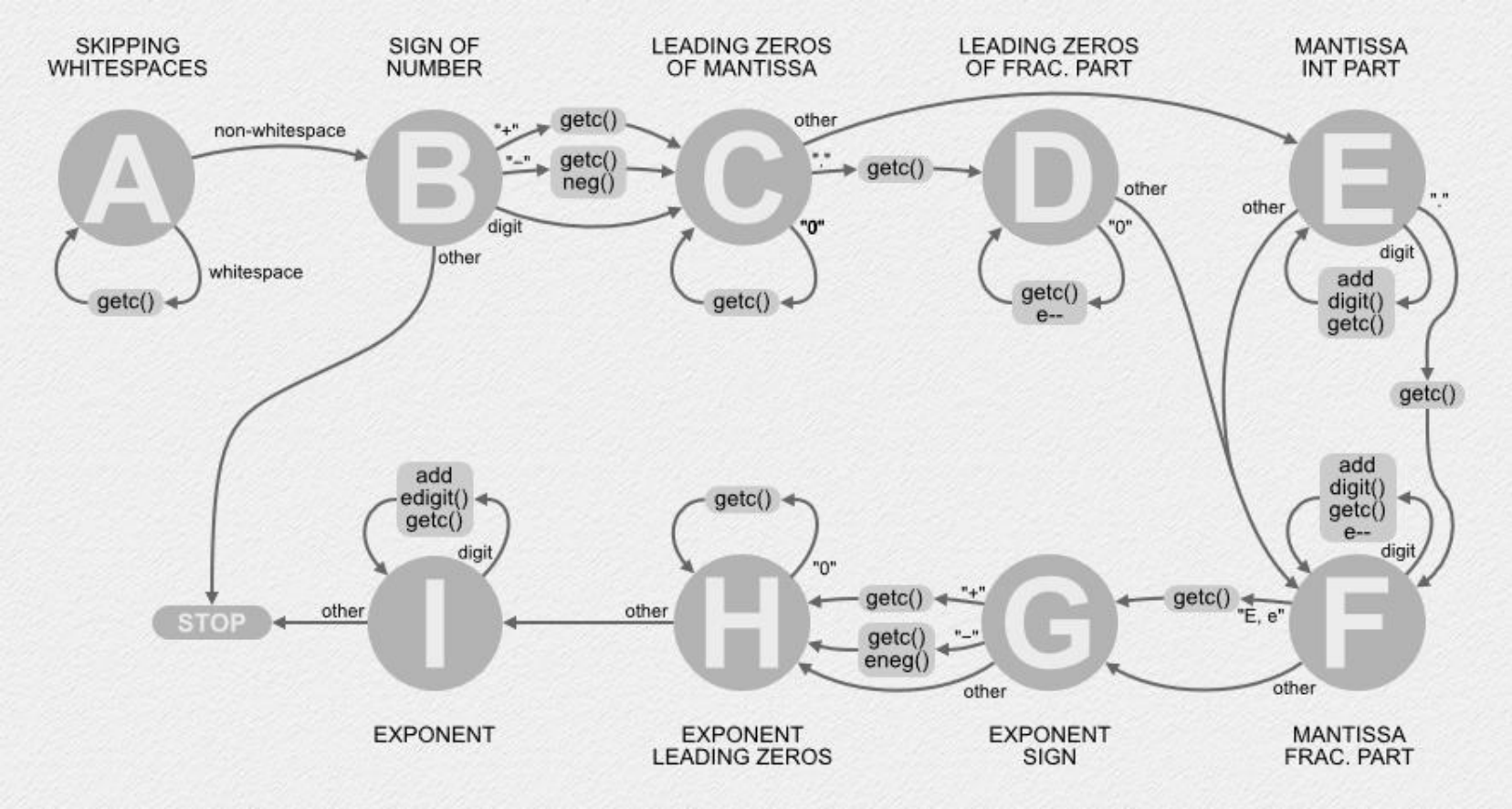
If the first character is a number, then we have a positive number and we need to start a whole number.

if (nChar==1) and ((c>=’0’) and (c<=‘9’)) {sign=+1; fraction = false}

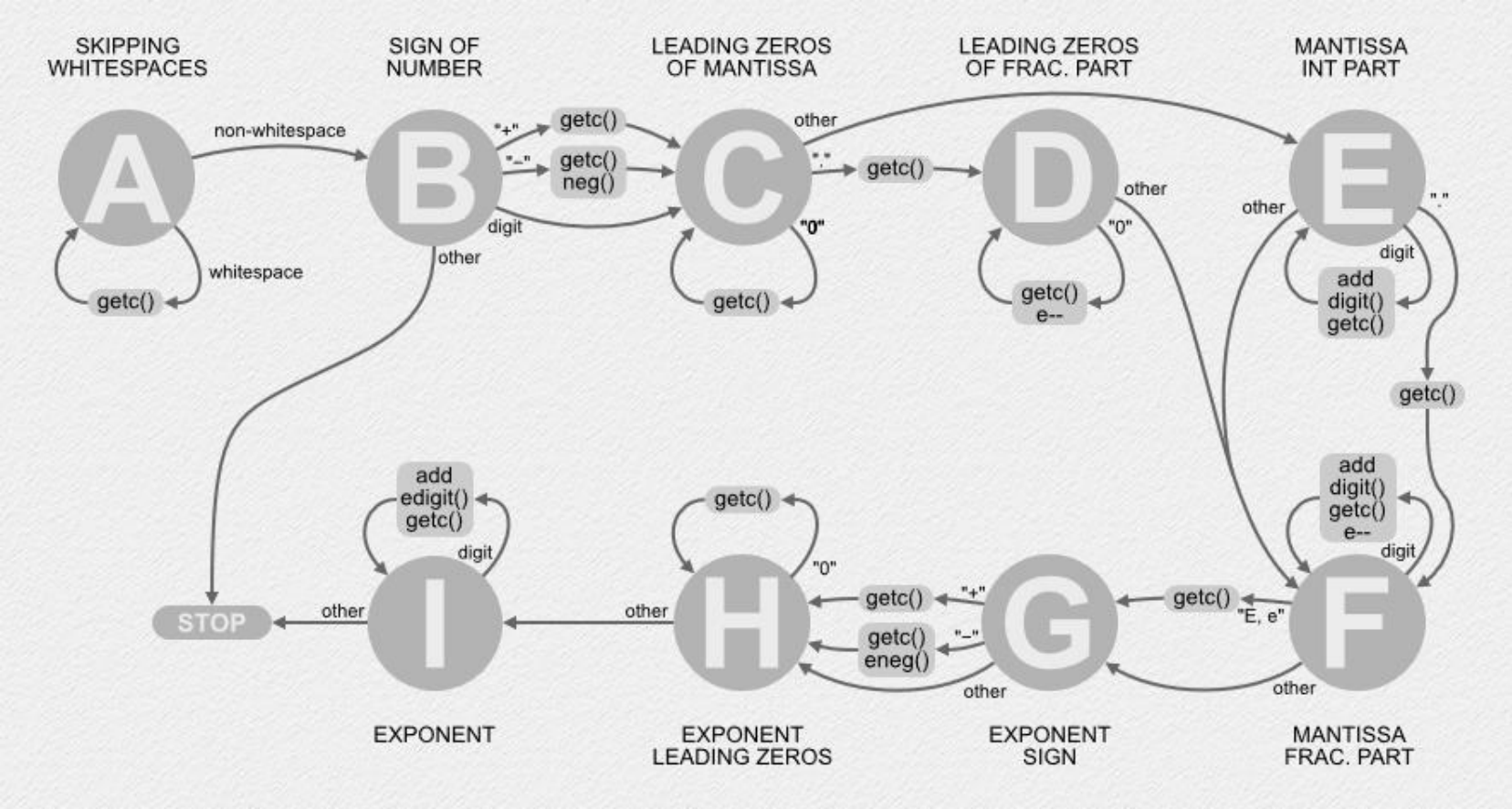
If the seconds character is (OMG, there must be a million combinations!)

if (nChar==2) and ARGHHHHHHHHH!

Floating Point Number Flowchart



So how do you turn this into code?



FSM Code Template

```
Routine FSM
        State = 0; (And other initializations)
        Do forever
                do common stuff;
                switch (state):
                case 0:
                        do stuff; Set State or exit loop if needed; break;
                case 1:
                        do stuff; Set State or exit loop if needed; break;
                case …
                        do stuff; Set State or exit loop if needed; break;
                default:
                        do stuff; Set State or exit loop if needed; break;
                end switch
        End forever
End Routine
```

Crude implementation of floating point parser

```
Routine TextToFloat(string)
        State = 0; sign = 1; whole= 0; fraction=0
        Do forever
                c = getNextCharFromString()
                switch (state):
                case 0: Do Stuff; break;        // Skipping leading whitespace
                case 1: Do Stuff; break;        // Sign of number
                case 2: Do Stuff; break;        // Leading zeros of mantissa
                case 3: Do Stuff; break;        // Fractional leading zeros
                case 4: Do Stuff; break;        // Integer part
                case 5: Do Stuff; break;        // Fractional part
                case 6: Do Stuff; break;        // Exponent sign
                case 7: Do Stuff; break;        // Exponent leading zeros
                case 8: Do Stuff; break;        // Exponent
                case 9: Do Stuff; break;        // Stop
                default: ASSERT(); break;       // Should never get here!
                End Switch
        End forever
End Routine
```

Skipping leading whitespace

```
case 0:
        if (c == <space>) or (c == <tab>)
                State = 0;  // No change
        else
                pushCharBackToString(c);
                State = 1;  // Sign of number
        endif
        break
```

Sign of number (Digit, +, -, EOS, or other)

```
case 1:
        if (c == '-')
                sign = -1;
                State = 2;  // Leading zeros of mantissa
        else if (c == '+')
                sign = 1;
                State = 2; // Leading zeros of mantissa
        else if (c >= '0' and c <= 9)
                pushCharBackToString(c);
                State = 2; // Leading zeros of mantissa
        else if (c == EndOfString)
                State = 9;  // Exit
        else
                State = 9;  // Exit
        endif
        break;
```

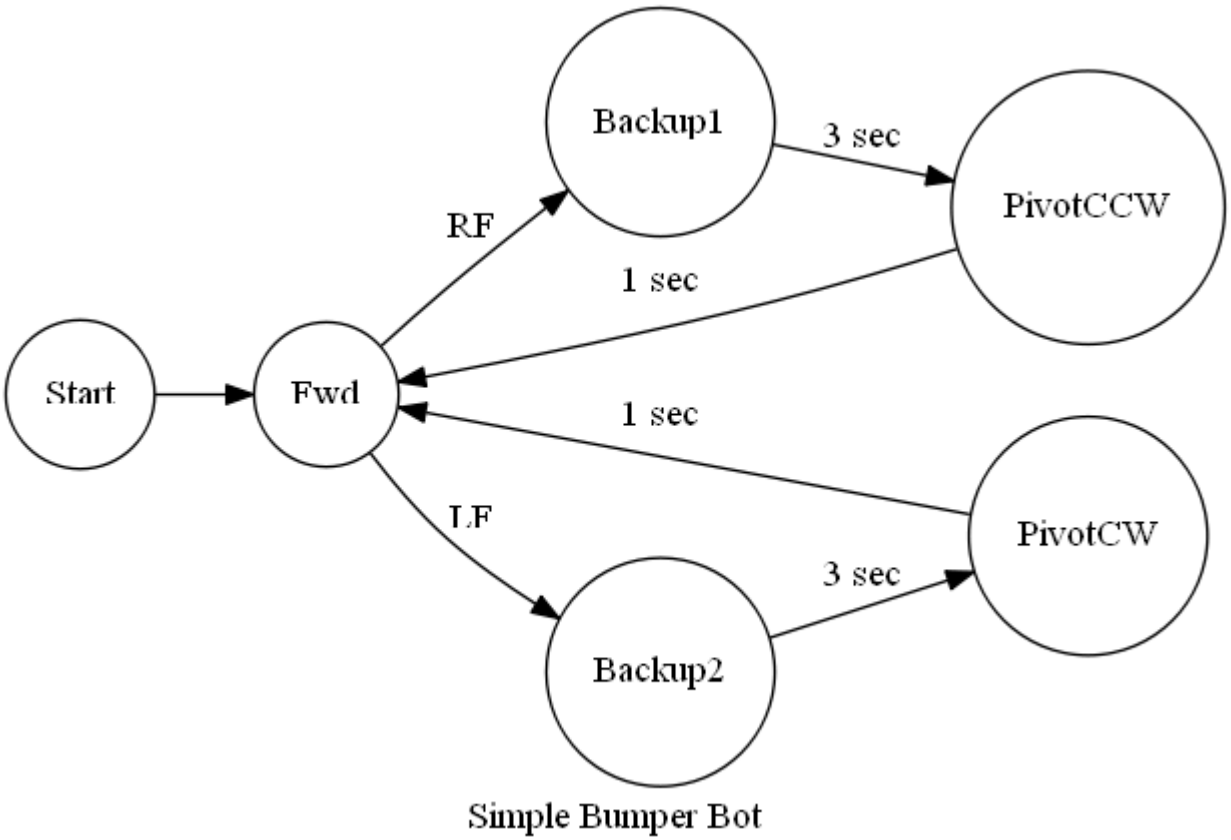
Sign of number (Digit, +, -, Null, or other)

```
case 1:
        if (c == '-')
                sign = -1;
                State = 2;  // Leading zeros of mantissa
        else if (c == '+')
                sign = 1;
                State = 2; // Leading zeros of mantissa
        else if (c >= '0' and c <= 9)
                pushCharBackToString(c);
                State = 2; // Leading zeros of mantissa
        else if (c == Null)
                State = 9;  // Exit
        else  // other
                State = 9;  // Exit
        endif
        break;
```

A More Appropriate Example

- Simple Bumper bot.
  - Has two motor tank drive (RM, LM)
  - Has bumpers on front corners (FrontRight, FrontLeft)
  - Goes forward till it hits something then
    - Moves backwards a bit (time based)
    - Turns away from the object for a bit (time based)
    - Then goes forward again.

Simple State Diagram



Short Detour

To create these “Bubble” diagrams, you can obviously use paper and pencil, PowerPoint, or something like Visio.

But a real simple way is to use the DOT program from the Graphviz package.

DOT allows the user to generate bubble diagrams from a text file.

The previous diagram was generated auto-magically from this text file:

```
digraph FSM {
 rankdir   = LR;
 fontsize  = 10;
 size      = "8.5,11.0";
 overlap   = false;
 fontsize  = 14;
 label     = "Simple Bumper Bot";

 node [shape=circle]; Start;
 node [shape=circle]; Fwd;
 node [shape=circle]; Backup1;
 node [shape=circle]; PivotCCW;
 node [shape=circle]; Backup2;
 node [shape=circle]; PivotCW;

 Start      -> Fwd      [ label = "" ];
 Fwd        -> Backup2  [ label = "LF" ];
 Backup2    -> PivotCW  [ label = "3 sec" ];
 PivotCW    -> Fwd      [ label = "1 sec" ];
 Fwd        -> Backup1  [ label = "RF" ];
 Backup1    -> PivotCCW [ label = "3 sec" ];
 PivotCCW   -> Fwd      [ label = "1 sec" ];

}
```

Start with the Template

```
Routine FSM
        State = 0; (And other initializations)
        Do forever
                do common stuff;
                switch (state):
                case 0:
                        do stuff; Set State or exit loop if needed; break;
                case 1:
                        do stuff; Set State or exit loop if needed; break;
                case …
                        do stuff; Set State or exit loop if needed; break;
                default:
                        do stuff; Set State or exit loop if needed; break;
                end switch
        End forever
End Routine
```

Adjust for Arduino

```
Init(void) {
        State = 0; (And other initializations)
}
Loop(void) {
        for (;;) {
                do common stuff;
                switch (state): {
                case 0:
                        do stuff; Set State or exit loop if needed; break;
                case 1:
                        do stuff; Set State or exit loop if needed; break;
                case …
                        do stuff; Set State or exit loop if needed; break;
                default:
                        do stuff; Set State or exit loop if needed; break;
                }
        }
}
```

Define your States

```
#define START      0
#define FWD        1
#define BACKUP1    2
#define PIVOT_CCW  3
#define BACKUP2    4
#define PIVOT_CW   5
```

List all the conditions that change states

- Right front bumper pressed
- Left front bumper pressed
- Time delay

Update Template

```
Void init(void) {
        state = START;
}
Void loop(void) {
        int state = 0;
        for (;;) {
                do common stuff;
                switch (state):
                case START:
                        do stuff; Set State or exit loop if needed; break;
                case FWD:
                        do stuff; Set State or exit loop if needed; break;
                case BACKUP1:
                        do stuff; Set State or exit loop if needed; break;
                case PIVOT_CCW:
                        do stuff; Set State or exit loop if needed; break;
                case BACKUP2:
                        do stuff; Set State or exit loop if needed; break;
                case PIVOT_CW:
                        do stuff; Set State or exit loop if needed; break;
                default:
                        do stuff; Set State or exit loop if needed; break;
                }
        }
}
```

Implement START

```
Case START:
        setMotor(RIGHT, DIR_FWD);
        setMotor(LEFT, DIR_FWD);
        state = FWD;  // Switch to FWD unconditionally.
        break;
```

Implement FWD

```
Case FWD:
        rf = getRightFrontBumper();
        lf = getLeftFrontBumper();
        if (rf == true) {
                state = BACKUP1;
        } else if (lf == true) {
                state = BACKUP2;
        } else {
                // Keep on trucking
        }
        break;
```

Implement BACKUP1

```
case BACKUP1:
        setMotor(RIGHT, DIR_REV);
        setMotor(LEFT, DIR_REV);
        delay(3*1000);
        state = PIVOT_CCW;
        break;
```

Implement PIVOT_CCW

```
case PIVOT_CCW:
        setMotor(RIGHT, DIR_FWD);
        setMotor(LEFT, DIR_REV);
        delay(1*1000);
        state = FWD;
        break;
```

Implement BACKUP2

```
case BACKUP2:
        setMotor(RIGHT, DIR_REV);
        setMotor(LEFT, DIR_REV);
        delay(3*1000);
        state = PIVOT_CW;
        break;
```

Implement PIVOT_CW

```
case PIVOT_CW:
        setMotor(RIGHT, DIR_REV);
        setMotor(LEFT, DIR_FWD);
        delay(1*1000);
        state = FWD;
        break;
```

Demo

Problem with ‘delay’

During delays, the robot can’t do anything else.

It can’t update a display, take measurements, respond to voice, …

(It will still respond to interrupts)

One solution is to use an RTOS. (Remember those? ☺)

- As the delay is happening other tasks can run.

Another solution is to use target times.

- Compute a time to change states
- Compare this time each iteration to the current time.
- Switch states when the current time >= to the target time.

Sample using target times

```
case PIVOT_CW:
    setMotor(RIGHT, DIR_REV);
    setMotor(LEFT, DIR_FWD);
    delay(1*1000);
    state = FWD;
    break;
```

Must be careful with initialization of these variables!

```
case PIVOT_CW:
    if (firstPass) {
        firstPass = false;
        targetTime = millis() + 1000;
        setMotor(RIGHT, DIR_REV);
        setMotor(LEFT, DIR_FWD);
    } else if (millis() > targetTime) {
        state = FWD;
        firstPass = true;
    } else {
    }
    break;
```

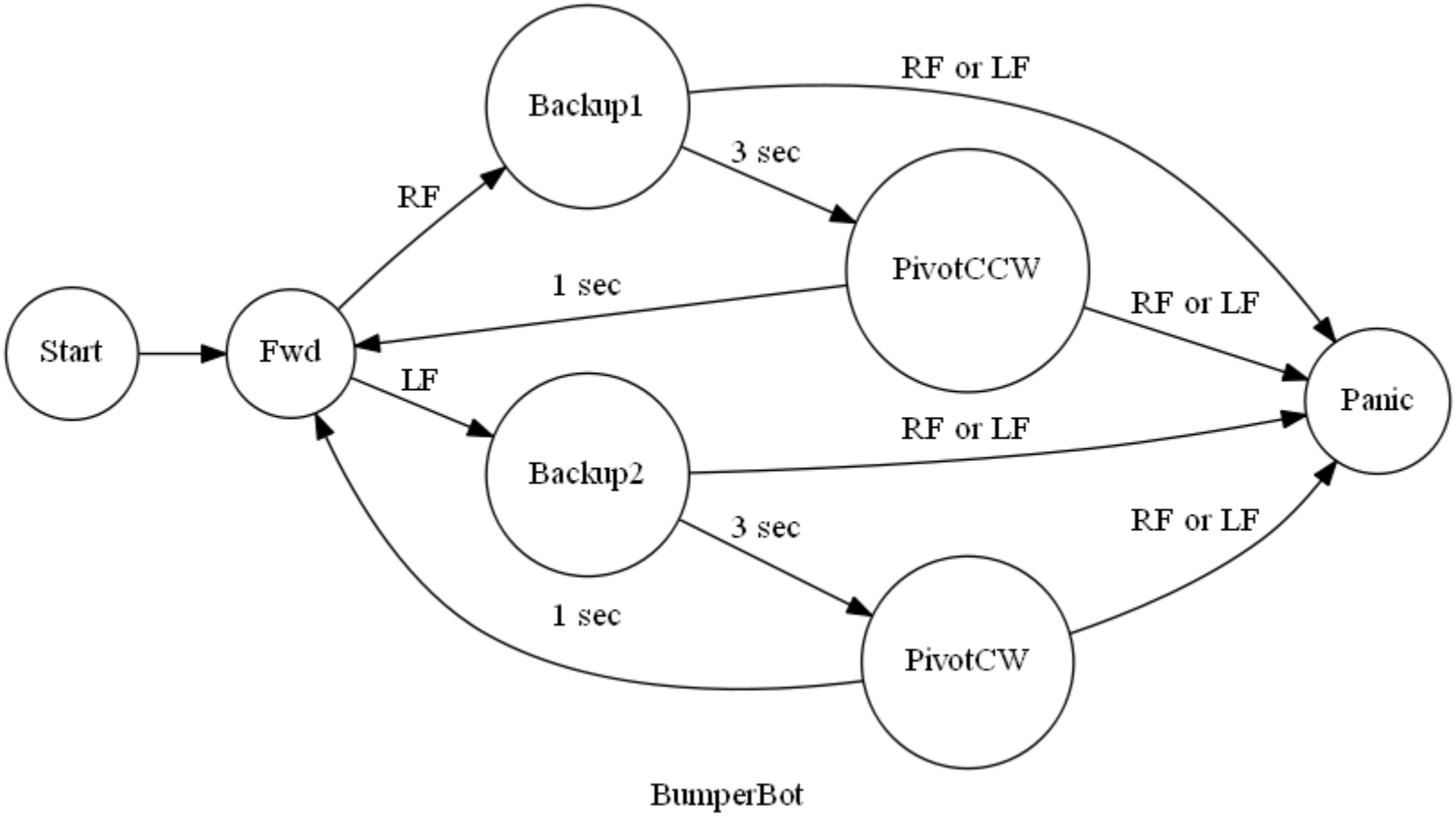
Bumper Problems

- What happens if the bumper does not get released backing up?
- What happens if the robot hits something going backwards?
- What happens if the bumper gets pressed during pivots?
- What happens is both bumpers get hit at the same time?

A More Sophisticated Example

- Complex Bumper bot.
  - Has two motor tank drive (RM, LM)
  - Has bumpers on front corners (FrontRight, FrontLeft)
  - Goes forward till it hits something then
    - Moves backwards a bit
    - Turns away from the object for a bit
    - Then goes forward again.
  - If it hits something turning
    - Full stop,
    - Wait for rescue.
  - If it hits something going backwards
    - Full stop
    - Wait for rescue

Equivalent State Machine



Examples of when to use an FSM

- Point of sale payment system
  - Total OK?, Swipe CC, Wait for auth, Please sign, have clerk check sig, …
- Gas pump
  - Swipe card, Debit?, Select fuel, Pump (show ads), submit CC, …
- Decoding binary messages (GPS binary formats)
  - Read sync word, read header, validate header, read body, validate body, process body.
- Convert ASCII to floating point values
  - Horner's rule
- Parsing languages/scripts
- Sequencing
  - Apply power, wait, send command, wait for reply, process reply, turn off power.

Documentation

Use the DOT program from Graphviz and a simple script.

Put the dot commands inside your code.

Run a script to strip out the dot commands into a new file.

Run dot on the file to generate a PDF, BMP, PNG, …

Use that in your documentation.

State machine definition at top of file

```
/* File header …….. */

/* WEIRD_TOKEN digraph FSM { */
/* WEIRD_TOKEN  rankdir   = LR; */
/* WEIRD_TOKEN  fontsize  = 10; */
/* WEIRD_TOKEN  size      = "8.5,11.0"; */
/* WEIRD_TOKEN  overlap   = false; */
/* WEIRD_TOKEN  fontsize  = 14; */
/* WEIRD_TOKEN  label     = "Simple Bumper Bot"; */
/* WEIRD_TOKEN  node [shape=circle]; Start; */
/* WEIRD_TOKEN  node [shape=circle]; Fwd; */
/* WEIRD_TOKEN  node [shape=circle]; Backup1; */
/* WEIRD_TOKEN  node [shape=circle]; PivotCCW; */
/* WEIRD_TOKEN  node [shape=circle]; Backup2; */
/* WEIRD_TOKEN  node [shape=circle]; PivotCW; */
```

State machine definitions in code

```
Case FWD:
        rf = getRightFrontBumper();
        lf = getLeftFrontBumper();
        if (rf == true) {
                state = BACKUP1;
                /* WEIRD_TOKEN  Fwd -> Backup1   [ label = “RF" ];  */
        } else if (lf == true) {
                state = BACKUP2;
                /* WEIRD_TOKEN  Fwd -> Backup2   [ label = “LF" ];  */
        } else {
                // Keep on trucking
        }
        break;
```

State machine definition at bottom of file

.

.

.

.

/* WEIRD_TOKEN }*/

/* Bottom of file XYZ.c */

Script to extract DOT commands

Read each line

- If line has “WEIRD_TOKEN” in it
- Strip off leading whitespace
- Strip off ‘/* WEIRD_TOKEN’
- Strip off ‘*/’ from end
- Write results to file

This is like a 15 line program in Python.

Practical advise

A large FSM is ugly to program and maintain

You have a switch for each case.

Each case has at least a few lines of code.

Pretty soon you have the subroutine from hell.

It is almost impossible to comprehend.

Solution 1: Subroutines

```
Void loop(void) {
        for (;;) {
                FSM_Top();
                switch (state):
                case START:         FSM_Start();         break;
                case FWD:           FSM_Fwd();           break;
                case BACKUP1:       FSM_Backup1();       break;
                case PIVOT_CCW:     FSM_Pivot_CCW(); break;
                case BACKUP2:       FSM_Backup2();       break;
                case PIVOT_CW:      FSM_Pivot_CW();      break;
                default:            FSM_default();       break;
                }
        }
```

Solution 1: Subroutines

- Drawback:
  - Because all your code is in separate subroutines you have issues with data sharing.
  - Solutions:
    - Expose the variables as globals.
    - Write set/get routines for the data. (Recommended)
      - void setState(int state);
      - int getState(void);

Solution 2: Table Driven

To implement this, you need to be very comfortable with structures, arrays, typedefs, and callback routines (function pointers).

If you are then create a table of callback routines for each state.

Solution 2: Table Driven

```
TableEntry_t table[] = {
        FSM_Start,
        FSM_Fwd,
        FSM_Backup1,
        FSM_Pivot_CCW,
        FSM_Backup2,
        FSM_Pivot_CW,
        FSM_default
};
```

```
void loop(void)
{
        entry = table[state];
        if (entry.cb != NULL) {
                entry.cb();
        }
}
```

Questions?