Introduction to Stateflow (for MATLAB)

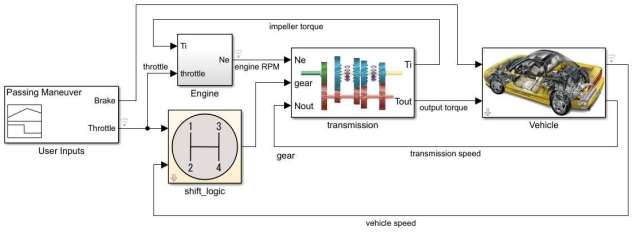
Alex Tarchini
(alex@mathworks.com)

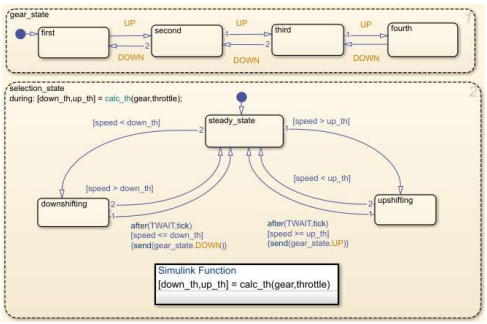
Why use State Machines

A state transition diagrams can serve as a high-level starting point for a complex software design process.

This can prove a far more effective development strategy than diving straight into writing thousands of lines of code

Simulink and Stateflow are a powerful combination





Simulink is used to respond to continuous changes

Stateflow is used to respond to discrete changes

3

Simulink and Stateflow are a powerful combination

Systems need to respond to both continuous and discrete changes

Continuous	Suspension dynamics
Discrete	Gear changes

Continuous	Propulsion system
Discrete	Liftoff stages

Continuous	Rotational movement
Discrete	Operational modes

4

What we'll cover during this session

- State Machines in a nutshell
- Why use State machines
- When to use Stateflow for designing logic
- Harel charts
  - Hierarchy
  - Parallelism
  - Broadcasting
- Examples
- Questions

State Machines in a nutshell

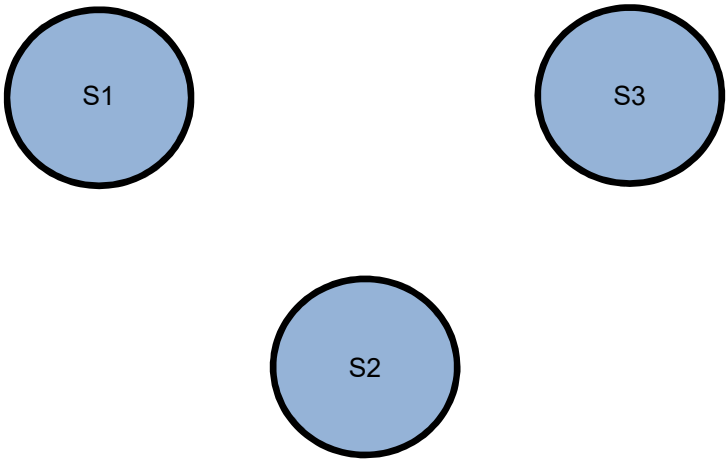
A finite state machine is a **method** of modeling a system comprised of a **limited** number of modes: depending on which mode it's in, the machine will behave in one manner or another.

And while finite state machines can be used to describe items both natural and manmade, for historical reasons the term most commonly refers to **computing systems**.

Now there are many ways of expressing finite state machines, though a **graphical approach** is often the one taken.

6

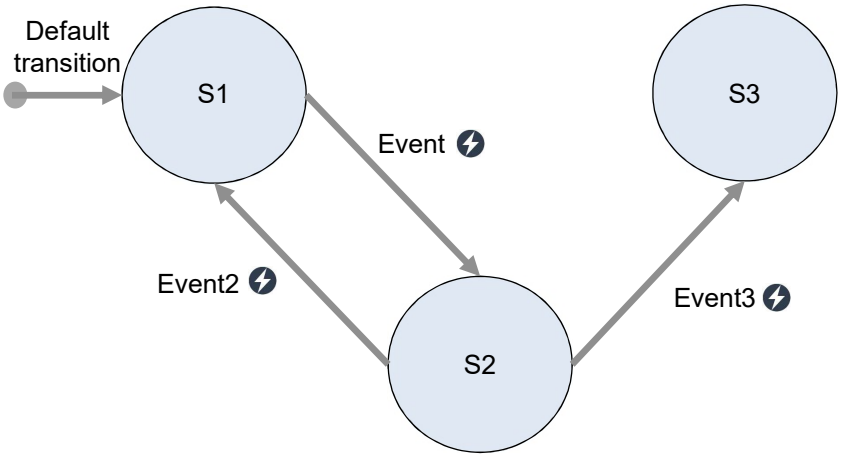
Basic elements



States

7

Basic elements



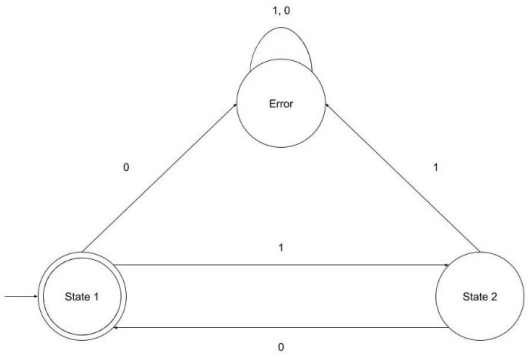
Transitions

8

Let's create a Finite State Machine that parses a sequence that:

- starts with 1
- every 1 must be followed by 0
- every 0 must be followed by 1

For example, 1010 is a valid input sequence but 110 and 10100 are not.

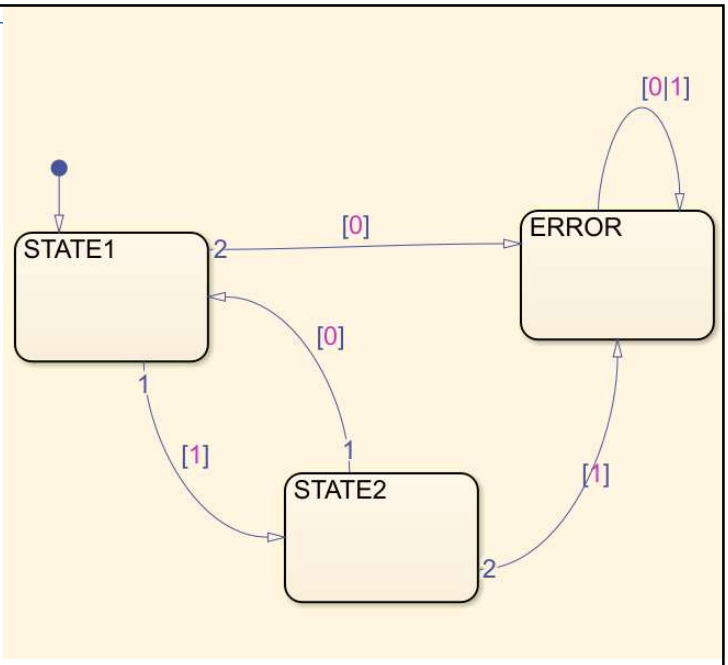


9

```
def match(self, current_state, state_input):
    """Determines if the state and the input satisfies this transition relation"""
    return self.current_state == current_state and self.state_input == state_input

class FSM:
  """A basic model of computation"""

    def __init__(
            self,
            states=[],
            alphabet=[],
            accepting_states=[],
            initial_state=''):
        self.states = states
        self.alphabet = alphabet
        self.accepting_states = accepting_states
        self.initial_state = initial_state
        self.valid_transitions = False

    def add_transitions(self, transitions=[]):
        """Before we use a list of transitions, verify they only apply to our states"""
        for transition in transitions:
            if transition.current_state not in self.states:
                print(
                    'Invalid transition. {} is not a valid state'.format(
                        transition.current_state))
                return
            if transition.next_state not in self.states:
                print('Invalid transition. {} is not a valid state'.format)
                return
        self.transitions = transitions
        self.valid_transitions = True

    def __accept(self, current_state, state_input):
        """Looks to see if the input for the given state matches a transition rule"""
        # if the input is valid in our alphabet
        if state_input in self.alphabet:
            for rule in self.transitions:
                if rule.match(current_state, state_input):
                    return rule.next_state
            print('No transition for state and input')
            return None
        return None

    def accepts(self, sequence):
        """Processes an input stream to determine if the FSM will accept it"""
        # Check if we have transitions
        if not self.valid_transitions:
            print('Cannot process sequence without valid transitions')

        print('Starting at {}'.format(self.initial_state))
        # When an empty sequence provided, we simply check if the initial state
        # is an accepted one
        if len(sequence) == 0:
            return self.initial_state in self.accepting_states

        # Let's process the initial state
        current_state = self.__accept(self.initial_state, sequence[0])
        if current_state is None:
            return False

        # Continue with the rest of the sequence
        for state_input in sequence[1:]:
            print('Current state is {}'.format(current_state))
            current_state = self.__accept(current_state, state_input)
            if current_state is None:
                return False

        print('Ending state is {}'.format(current_state))
        # Check if the state is an accepting state, or is an accepting state
        return current_state in self.accepting_states
```

That state machine in a traditional programming language

https://stackabuse.com/theory-of-computation-finite-state-machines/



The same state machine in Stateflow

10

Why not just use MATLAB to design logic?

For simple logic, just use MATLAB

```
if u >= t
    y = u;
else
    y = 0;
end
```

But for logic that is more complex, Stateflow is the best tool to use

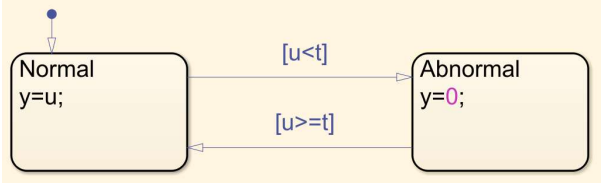
- By the way, even simple logic can often get complex

Comparing MATLAB and Stateflow for Designing Logic

Task: Create simple logic

Identify places in data where values are above a threshold t

```
for i = 1:length(inData)
    if(inData(i) >= t)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```

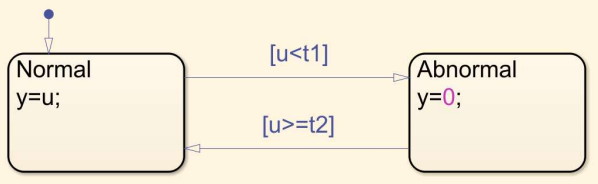


Comparing MATLAB and Stateflow for Designing Logic

Task: Add hysteresis

Identify places in data where values are below t1 and above t2

```
inNormalRegion = true;
for i = 1:length(inData)
    if(inNormalRegion && (inData(i)<t1))
        inNormalRegion = false;
    elseif(~inNormalRegion && (inData(i)>=t2))
        inNormalRegion = true;
    end
    if(inNormalRegion)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```



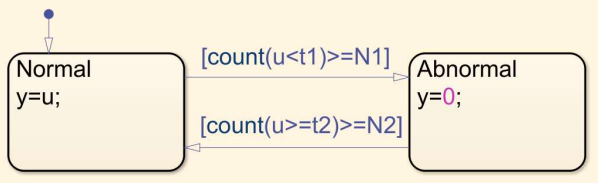
13

Comparing MATLAB and Stateflow for Designing Logic

Task: Add debouncing

Identify places in data where values are below t1 for at least N1 samples and above t2 for at least N2 samples

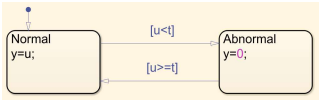
```
inNormalRegion = true;
counter = 0;
for i=1:length(inData)
    if(inNormalRegion)
        if(inData(i)<t1)
            counter = counter+1;
            if(counter>=N1)
                inNormalRegion = false;
            end
        else
            counter = 0;
        end
     else
        if(inData(i)>=t2)
            counter = counter+1;
            if(counter>=N2)
                inNormalRegion = true;
            end
        else
            counter = 0;
```



14

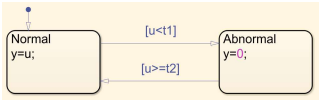
MATLAB code expands, while Stateflow chart remains compact

simple logic



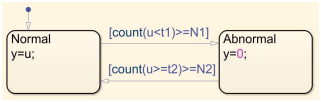

```
for i = 1:length(inData)
    if(inData(i) >= t)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```

+ hysteresis




```
inNormalRegion = true;
for i = 1:length(inData)
    if(inNormalRegion && (inData(i)<t1))
        inNormalRegion = false;
    elseif(~inNormalRegion && (inData(i)>=t2))
        inNormalRegion = true;
    end
    if(inNormalRegion)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```

+ hysteresis + debouncing



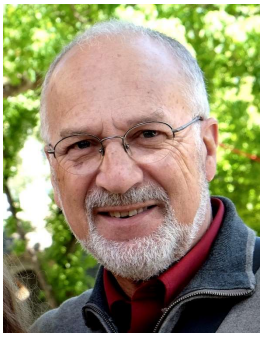

```
inNormalRegion = true;
counter = 0;
for i=1:length(inData)
    if(inNormalRegion)
        if(inData(i)<t1)
            counter = counter+1;
            if(counter>=N1)
                inNormalRegion = false;
            end
        else
            counter = 0;
        end
     else
        if(inData(i)>=t2)
            counter = counter+1;
            if(counter>=N2)
                inNormalRegion = true;
            end
        else
            counter = 0;
        end
     end
     if(inNormalRegion)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```

15

In the 1980s, Dr. David Harel took a fresh look at finite state machines and realized that state transition diagrams were an excellent way of expressing the logic of avionic systems.

In order to get everything he wanted, he had to expand their semantic with three new concepts:

- Parallelism
- Hierarchy
- Event broadcast

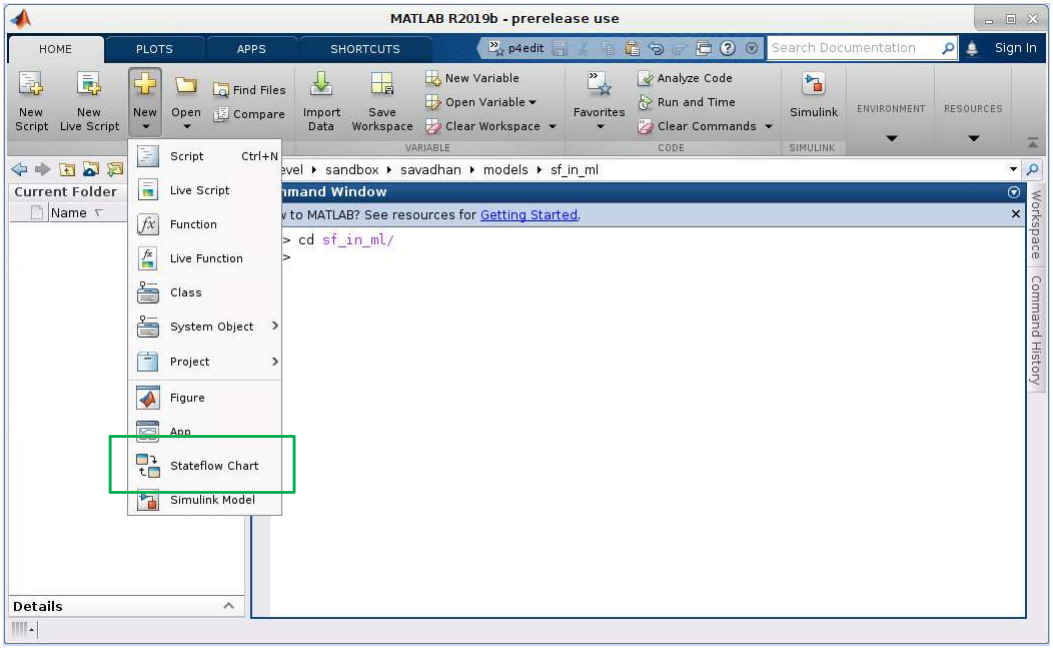


These enhanced diagrams were called *statecharts*

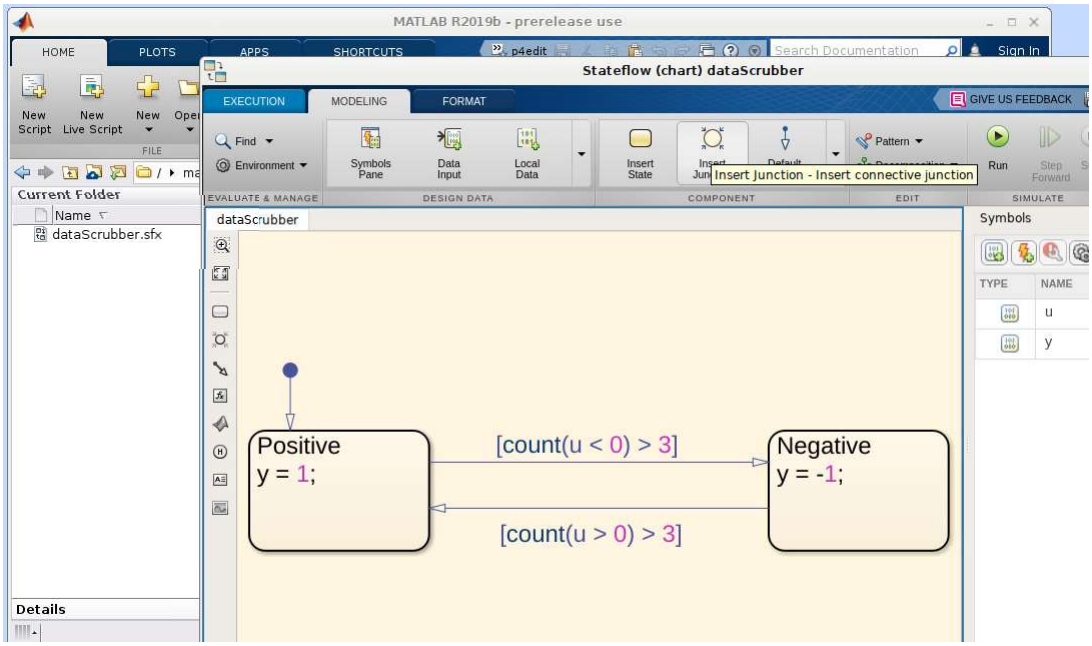
16

>> sldemo_fuelsys

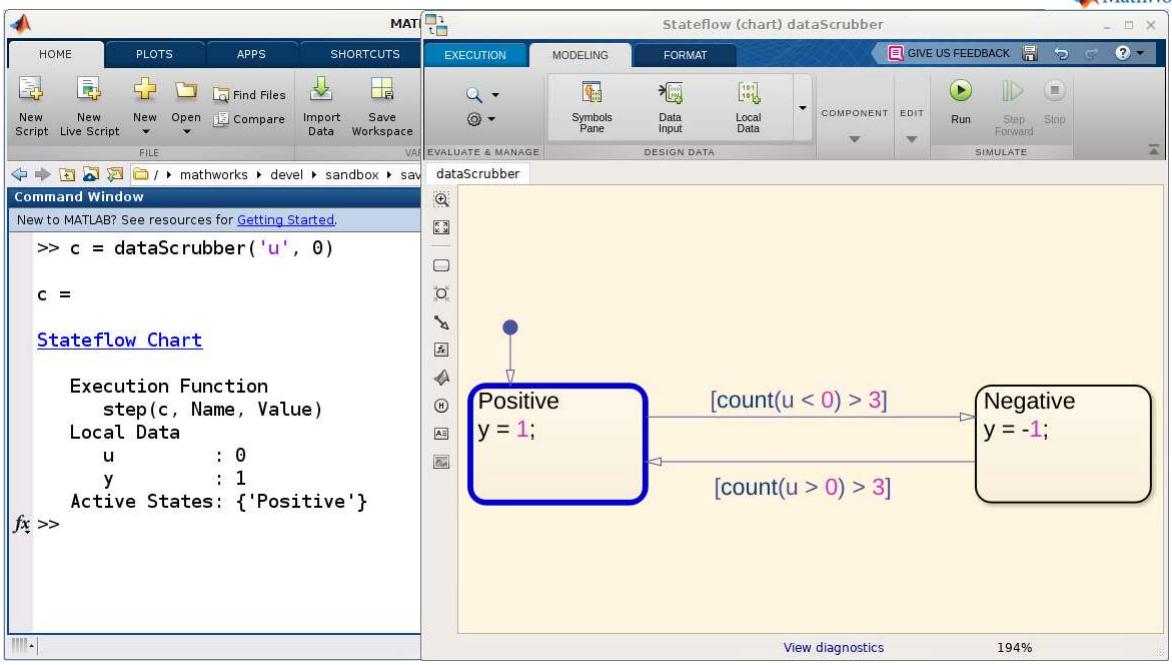
17



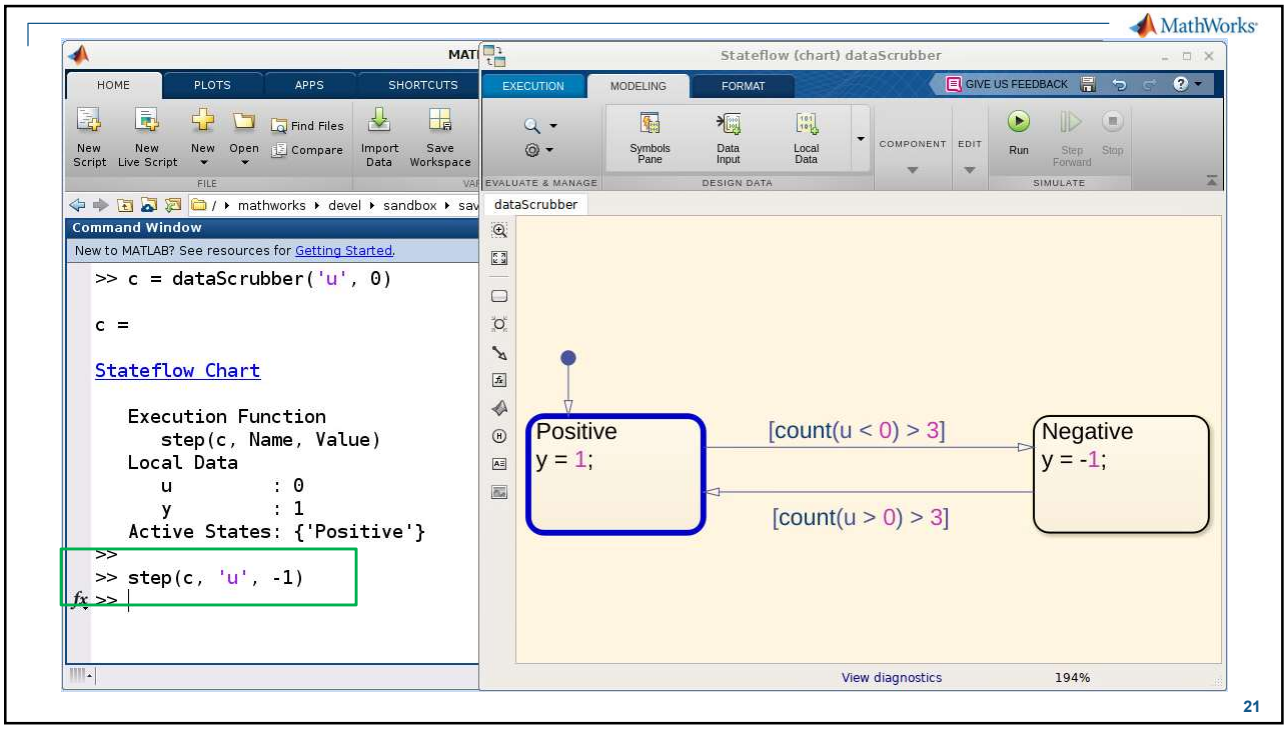
18

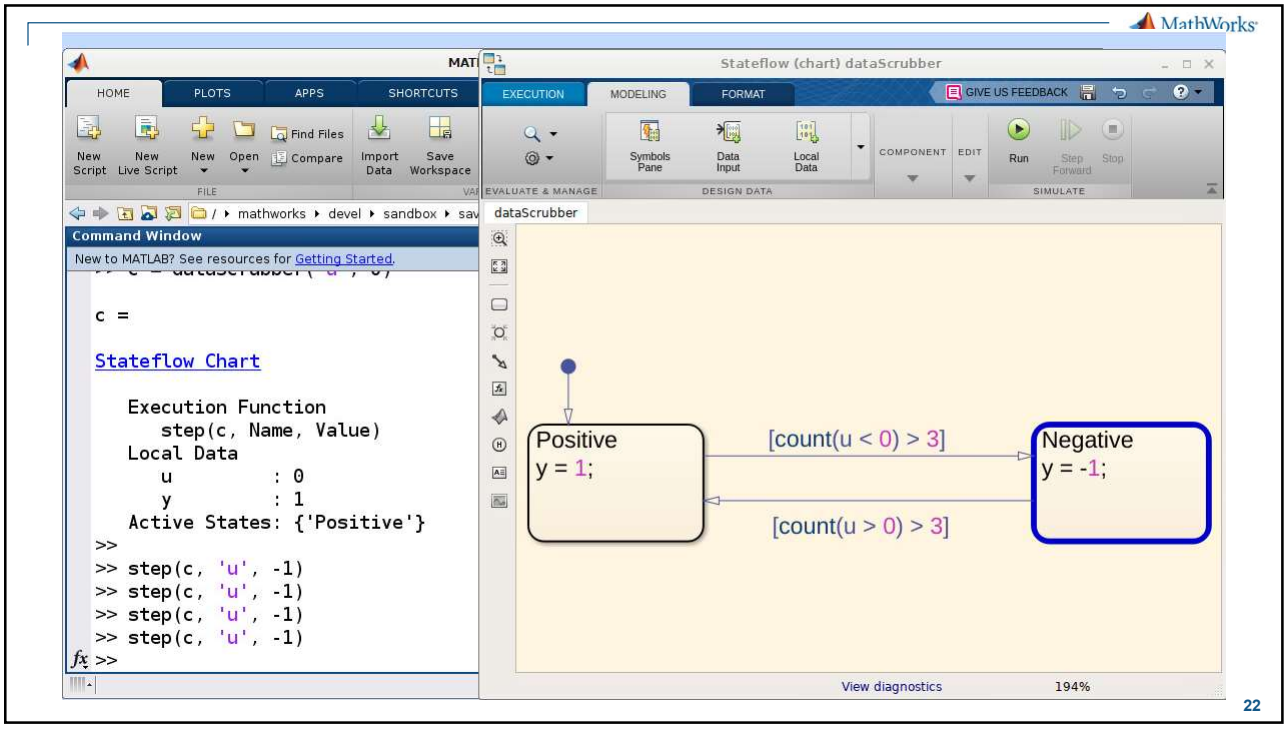

MATLAB R2019b - prerelease use
Stateflow (chart) dataScrubber
Insert Junction - Insert connective junction
dataScrubber
Positive
y = 1;
[count(u < 0) > 3]
[count(u > 0) > 3]
Negative
y = -1;
Symbols
TYPE
NAME
u
y

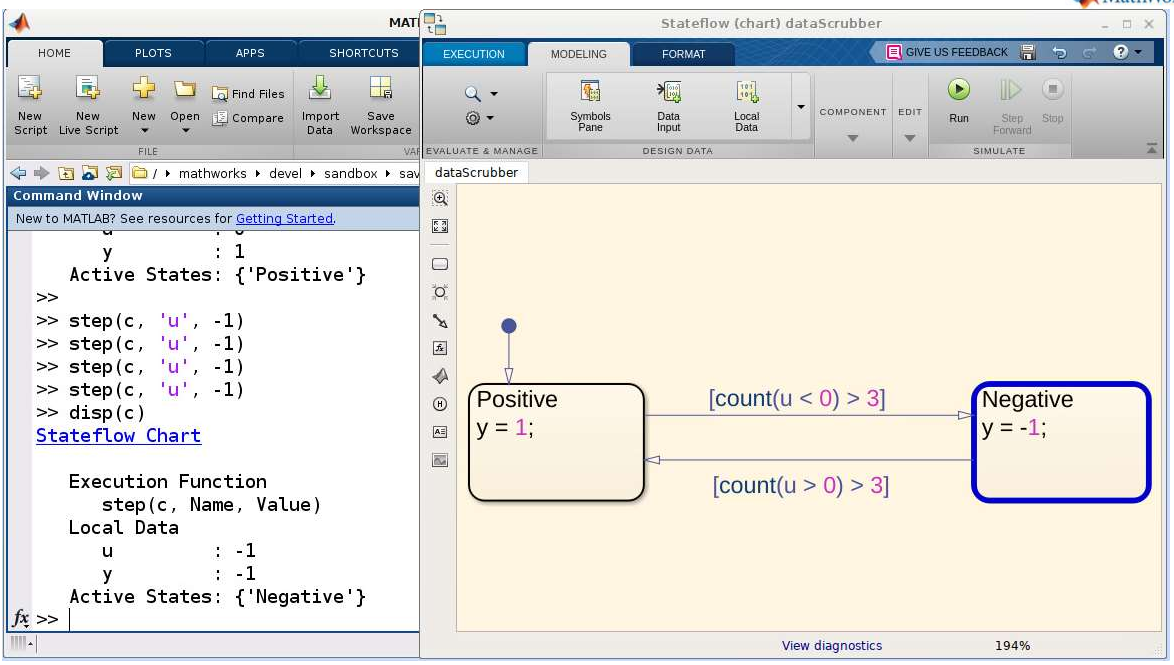
19


Stateflow (chart) dataScrubber
Command Window
>> c = dataScrubber('u', 0)
c =
Stateflow Chart
Execution Function
step(c, Name, Value)
Local Data
u : 0
y : 1
Active States: {'Positive'}
>>
dataScrubber
Positive
y = 1;
[count(u < 0) > 3]
[count(u > 0) > 3]
Negative
y = -1;
View diagnostics
194%

20

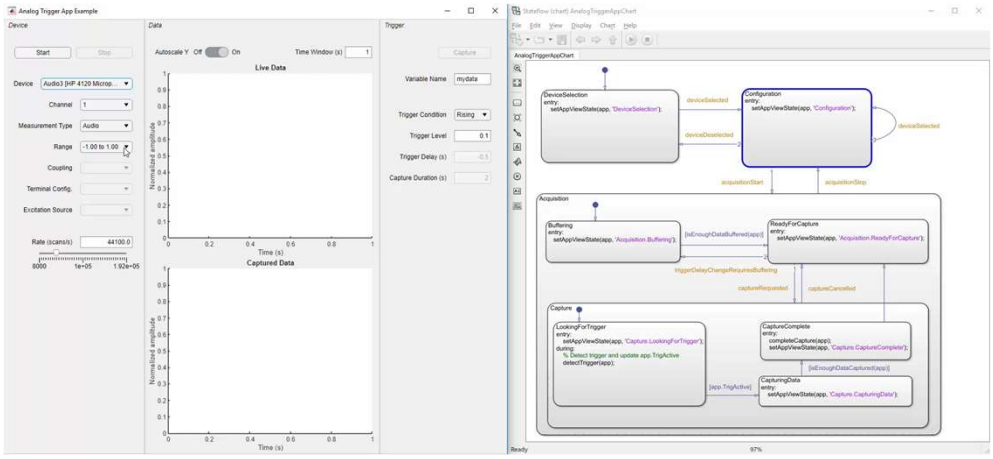

MathWorks
Stateflow (chart) dataScrubber
HOME
PLOTS
APPS
SHORTCUTS
EXECUTION
MODELING
FORMAT
GIVE US FEEDBACK
New Script
New Live Script
New
Open
Find Files
Compare
Import Data
Save Workspace
Symbols Pane
Data Input
Local Data
COMPONENT
EDIT
Run
Step Forward
Stop
FILE
EVALUATE & MANAGE
DESIGN DATA
SIMULATE
mathworks ▸ devel ▸ sandbox ▸ sav
dataScrubber
Command Window
New to MATLAB? See resources for Getting Started.
>> c = dataScrubber('u', 0)
c =
Stateflow Chart
Execution Function
step(c, Name, Value)
Local Data
u : 0
y : 1
Active States: {'Positive'}
>>
>> step(c, 'u', -1)
>>
Positive
y = 1;
[count(u < 0) > 3]
[count(u > 0) > 3]
Negative
y = -1;
View diagnostics
194%
21


MathWorks
Stateflow (chart) dataScrubber
HOME
PLOTS
APPS
SHORTCUTS
EXECUTION
MODELING
FORMAT
GIVE US FEEDBACK
New Script
New Live Script
New
Open
Find Files
Compare
Import Data
Save Workspace
Symbols Pane
Data Input
Local Data
COMPONENT
EDIT
Run
Step Forward
Stop
FILE
EVALUATE & MANAGE
DESIGN DATA
SIMULATE
mathworks ▸ devel ▸ sandbox ▸ sav
dataScrubber
Command Window
New to MATLAB? See resources for Getting Started.
c =
Stateflow Chart
Execution Function
step(c, Name, Value)
Local Data
u : 0
y : 1
Active States: {'Positive'}
>>
>> step(c, 'u', -1)
>> step(c, 'u', -1)
>> step(c, 'u', -1)
>> step(c, 'u', -1)
>>
Positive
y = 1;
[count(u < 0) > 3]
[count(u > 0) > 3]
Negative
y = -1;
View diagnostics
194%
22



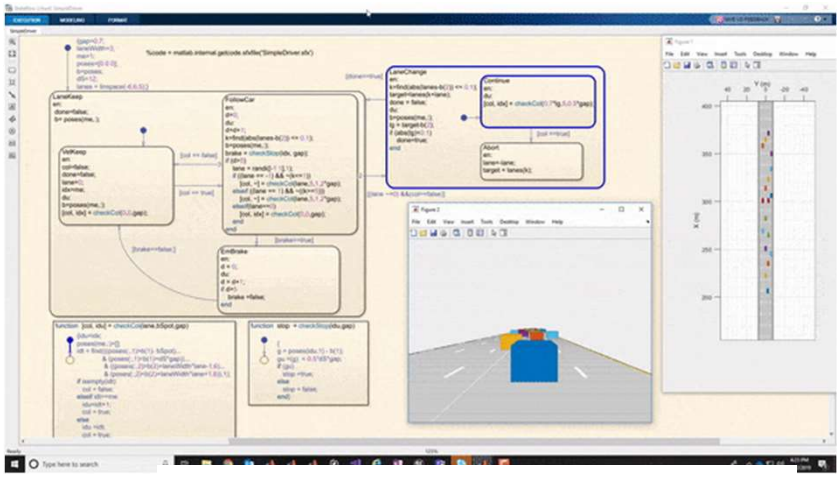
Uses of Stateflow in MATLAB

- Test and Measurement
  - Detecting a data stream trigger



Some other examples

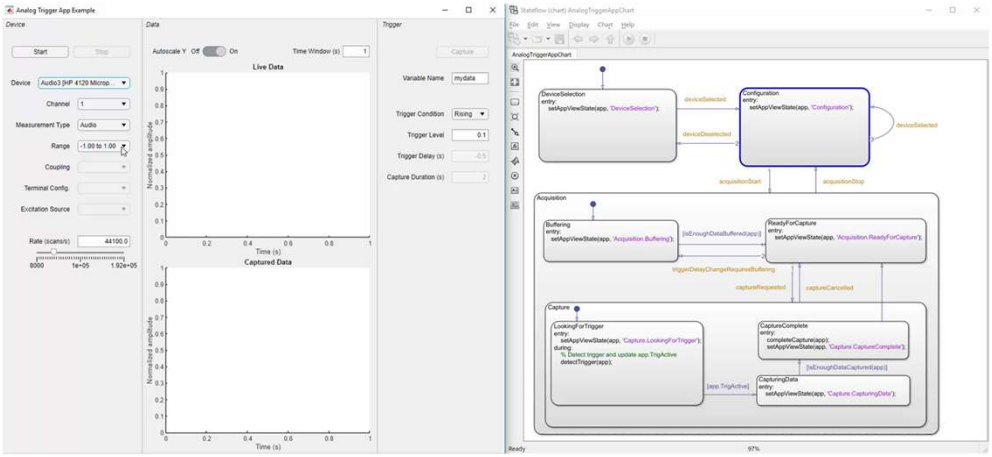
- Robotics and autonomous systems engineers
  - Design path planning, path following, and obstacle avoidance algorithms



25

Who gets most value in using state machines

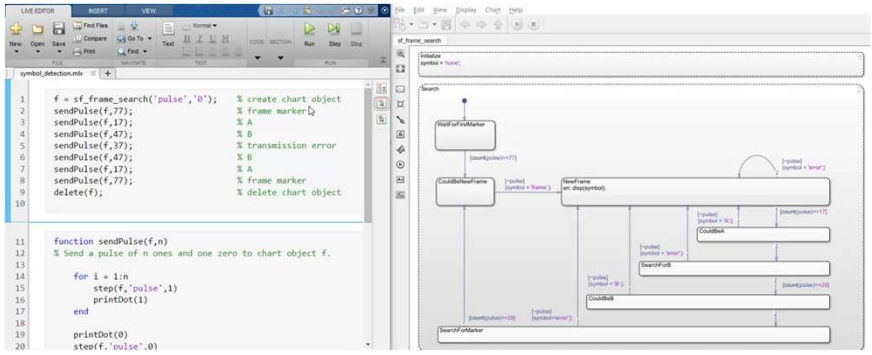
- UI developers
  - Control UI's developed in App Designer and GUIDE



26

Who gets most value in using state machines

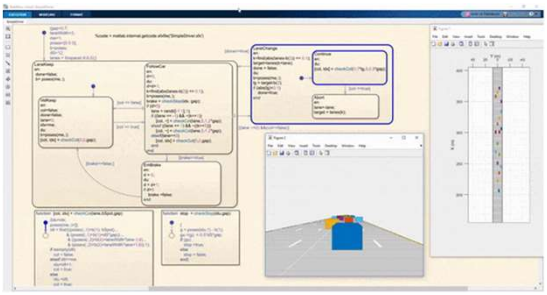
- UI developers
  - Control UI's developed in App Designer
- Communications engineers
  - Analyze streams of communication data and model communication protocols



27

Who gets most value in using state machines

- UI developers
  - Control UI's developed in App Designer and GUIDE
- Communications engineers
  - Analyze streams of communication data and model communication protocols
- Robotics and autonomous systems engineers
  - Design path planning, path following, and obstacle avoidance algorithms



28

Who gets most value in using state machines

- UI developers
  - Control UI's developed in App Designer and GUIDE
- Communications engineers
  - Analyze streams of communication data and model communication protocols
- Robotics and autonomous systems engineers
  - Design path planning, path following, and obstacle avoidance algorithms
- ADAS engineers
  - Analyze sensor data and apply ground truth labeling algorithms
- Radar engineers
  - Analyze streams of data coming from radar systems
- Test systems engineers
  - Monitor I/O of systems and analyze various modes of operation

29

Many examples included in shipping documentation

Getting started tutorials

Create Stateflow Charts for Execution as MATLAB Objects
Save standalone Stateflow charts outside of Simulink® models.

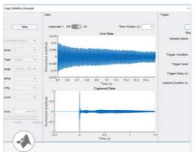
Execute and Unit Test Stateflow Chart Objects
Run Stateflow charts in MATLAB or through the Stateflow editor.

Debug a Standalone Stateflow Chart
Interrupt execution to step through each action in a Stateflow chart.

Execute Stateflow Chart Objects Through Scripts and Models
Create a MATLAB script or a Simulink model that invokes a standalone Stateflow chart.

Application examples
- Connecting to App Designer
- Communications example
- Comp finance example
- Data acquisition example

Featured Examples



Analog Trigger App by Using Stateflow Charts

stateflow execution in matlab

30

Question?

Thanks!

alex@mathworks.com