

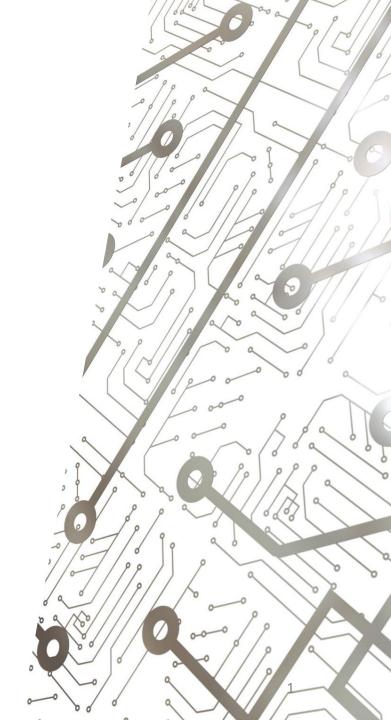




Software Lecture 6:

State Machines with

ROS



Contents

In this lecture, we shall cover:

- Overview for finite state machines and SMACH
- Creating a simple SMACH state machine in Python
- Passing data between states
- Calling ROS Services/Actions from SMACH state machines

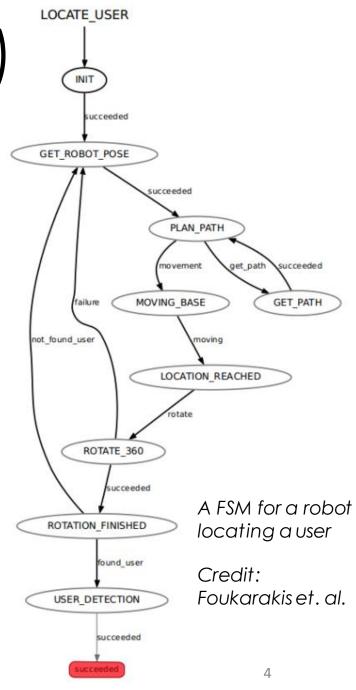
Before We Begin

- Again, I strongly suggest bookmarking the following link: https://github.com/OxRAMSociety/RobotArm
- All example code can be found in:
 Tutorials/Software Tutorials (2022)/Example Scripts
 Of course, code for this lecture will be in the Lecture 6 folder, and so on for future lectures

• I will highlight some code in these presentations, but you should refer to the Example Scripts for the entire code

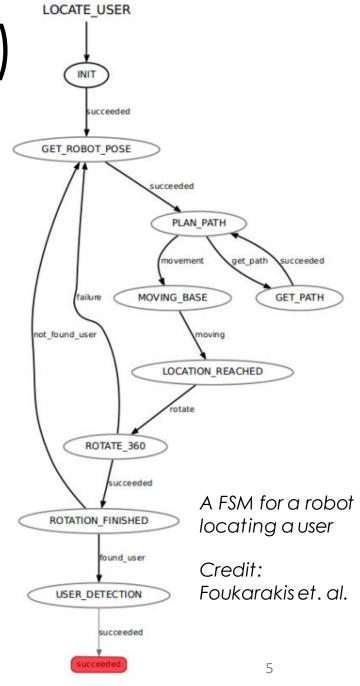
State Machines – Overview (1)

- Finite state machines (FSMs) are abstract machines that can be in exactly one of a finite set of states at a given time
- FSMs can transition between states depending on the outcomes from the current state
- When they are initialised, they start in the start state, and eventually finish in one of the possible end states



State Machines – Overview (2)

- The FSM algorithm works as follows:
 - With the inputs provided to the current state, carry out the behaviour defined for that state (e.g. move the arm to a target position)
 - 2. Observe the results from the behaviour (did the arm reach the target?)
 - 3. Use these results to choose the outcome of the state (from a discrete set of outcomes)
 - 4. Transition to the next state according to the outcome (each outcome has a corresponding transition)

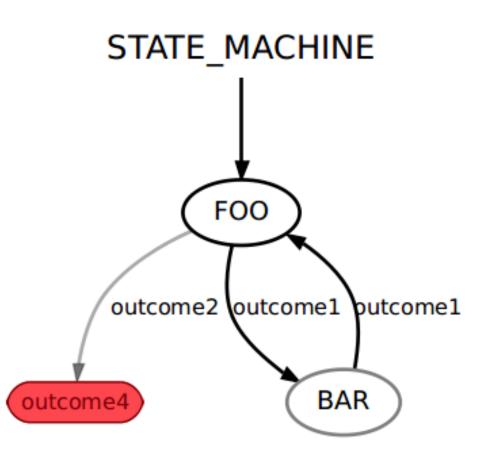


SMACH

- SMACH is a Python library used to create FSMs
- Very useful for rapidly creating complex robot behaviour
- While being ROS-independent at its core, SMACH has a package for ROS that allows its FSMs to integrate seamlessly into a ROS network
- We will be using this library to create behaviours for our robot arm

Creating a Simple FSM (1)

- Our first FSM shall transition between two states, FOO and BAR, a few tim before finishing on an end state
- In tutorial_script's scripts folder, create the file Example_SM_Simple.
- Find the script in our GitHub and co it into your file
- Let's have a look at the code...



A very similar example FSM Credit: ROS Wiki

Creating a Simple FSM (2)

```
# Define a class for state Foo
class Foo(smach.State):
    def init (self): # This method runs when the state is first initialised
        # Initialise the state and define its possible outcomes:
        smach.State. init (self, outcomes=['outcome1','outcome2'])
        self.counter = 0
    def execute(self, userdata): # This method runs when the state machine transitions to this state
        rospy.loginfo('Executing state F00')
        if self.counter < 3:
            self.counter += 1
            return 'outcome1' # The outcome returned by the method will determine the next state
        else:
            return 'outcome2'
```

- Here we define a class for the FOO state type, with an initialisation method as well as the method execute that runs when the FSM transitions to the state
 - In the code, we also define a class for BAR

Creating a Simple FSM (3)

```
name == ' main ':
rospy.init node('example sm simple')
# Create a SMACH state machine, defining the possible outcomes
sm = smach.StateMachine(outcomes=['end'])
# Open the container
with sm:
   # Add states to the container
   # Provide the name of the state, the class from which it is derived,
   # and the next state for each possible outcome
   smach.StateMachine.add('F00', Foo(),
                          transitions={'outcome1':'BAR', 'outcome2':'end'})
    smach.StateMachine.add('BAR', Bar(),
    transitions={'outcome1':'F00'})
# Execute SMACH plan
outcome = sm.execute()
```

 Here we initialise the FSM, add some states to it and define the transitions for the outcomes, and finally execute the FSM

Creating a Simple FSM (4)

 As always, give your script permissions, then run roscore in one terminal and your script in another. You should see this:

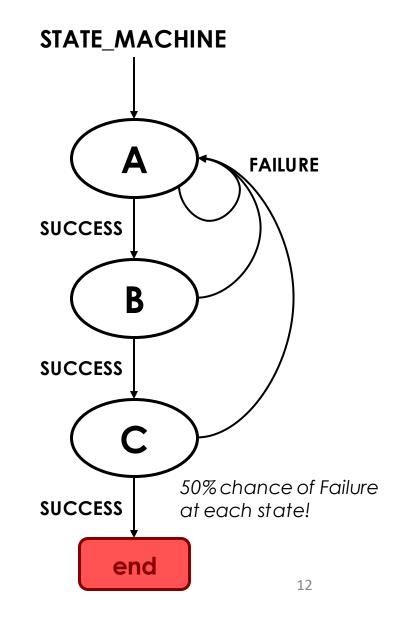
```
jacques@jacques-VirtualBox:~$ rosrun tutorial scripts Example SM Simple.py
[INFO] [1669550311.148662]: State machine starting in initial state 'FOO' with userdata:
INFO] [1669550311.149684]: Executing state F00
INFO] [1669550311.150522]: State machine transitioning 'FOO': 'outcome1'-->'BAR'
INFO] [1669550311.151384]: Executing state BAR
INFO] [1669550311.152321]: State machine transitioning 'BAR':'outcome1'-->'F00'
[INF0] [1669550311.153429]: Executing state F00
[INFO] [1669550311.156788]: State machine transitioning 'FOO':'outcome1'-->'BAR'
INFO] [1669550311.157808]: Executing state BAR
[INFO] [1669550311.161178]: State machine transitioning 'BAR':'outcome1'-->'F00'
INFO] [1669550311.161796]: Executing state FOO
[INFO] [1669550311.162506]: State machine transitioning 'FOO':'outcome1'-->'BAR'
[INF0] [1669550311.163123]: Executing state BAR
[INFO] [1669550311.163787]: State machine transitioning 'BAR':'outcome1'-->'F00'
[INF0] [1669550311.165545]: Executing state F00
[INFO] [1669550311.166535]: State machine terminating 'FOO':'outcome2':'end'
jacques@jacques-VirtualBox:~$
```

Passing Data Between States (1)

- In SMACH, the FSM can be given userdata, a set of variables global to the FSM and used to pass data between states
- You can even access the final userdata directly from the FSM once execution is complete
- For a state to read and write to a variable in userdata, it must have it defined as an input key and output key respectively

Passing Data Between States (2)

- We shall demonstrate this by implementing the FSM shown and logging the number of times it transitions to each state
- In tutorial_script's scripts folder, create the file Example_SM_Advanced.py
- Find the script in our GitHub and copy it into your file
- Let's have a look at the code...



Passing Data between States (3)

```
class stateA(smach.State):
   def init (self):
        smach.State. init (self, outcomes=['SUCCESS', 'FAILURE'], output keys=['A count'])
       # This state can output data to our custom 'A count' field for the state machine's global userdata
       # To let the state receive input data from a field, use 'input keys=['...']'
       self.counter = 0 # This variable is stored internally within the state,
                           # and will be remembered when the state is executed again
   def execute(self, userdata):
        rospy.loginfo("Executing State A...")
        r.sleep() # Sleep for one second
        self.counter += 1 # Increment the counter
       userdata.A count = self.counter # This demonstrates how to acess the global userdata
       # Determine next state (here, random with 50% chance of success)
        result = random.choice(['SUCCESS', 'FAILURE'])
        return result
```

- Here is our class definition for state A; note how we defined the variable A_count in the list of output keys
 - We update this variable every time the FSM transitions to state A

Creating a Simple FSM (4)

```
name == ' main ':
rospy.init node('example sm advanced')
r = rospy.Rate(2)
sm = smach.StateMachine(outcomes=['end'])
with sm:
    smach.StateMachine.add('A', stateA(), transitions={'FAILURE':'A','SUCCESS':'B'})
    smach.StateMachine.add('B', stateB(), transitions={'FAILURE':'A','SUCCESS':'C'})
    smach.StateMachine.add('C', stateC(), transitions={'FAILURE':'A','SUCCESS':'end'})
# To exit the State Machine, Tasks A, B, and C must each be successfully completed in order
# Each Task has a 50 percent chance of success
# If any Task fails, re-start from Task A
# initialise our userdata
sm.userdata.A count = 0
sm.userdata.B count = 0
sm.userdata.C count = 0
outcome = sm.execute()
# We can access these
rospy.loginfo("State A Attempted %d Times", sm.userdata.A count)
rospy.loginfo("State B Attempted %d Times", sm.userdata.B count)
rospy.loginfo("State C Attempted %d Times", sm.userdata.C count)
```

Here we set up and execute the FSM as before, but also initialise our userdata before execution and read the logged data values after execution

Passing Data Between States (5)

 Give your script permissions, then run roscore in one terminal and your script in another. You should see something like this:

```
jacques@jacques-VirtualBox:~$ rosrun tutorial scripts Example SM Advanced.py
[INFO] [1669550357.617107]: State machine starting in initial state 'A' with userdata:
        ['A count', 'B count', 'C count']
[INFO] [1669550357.618414]: Executing State A...
[INFO] [1669550358.118699]: State machine transitioning 'A':'FAILURE'-->'A'
[INFO] [1669550358.122125]: Executing State A...
[INFO] [1669550358.618136]: State machine transitioning 'A': 'FAILURE'--> 'A'
[INFO] [1669550358.620118]: Executing State A...
[INFO] [1669550359.118857]: State machine transitioning 'A':'SUCCESS'-->'B'
[INFO] [1669550359.120489]: Executing State B...
[INFO] [1669550359.619824]: State machine transitioning 'B':'SUCCESS'-->'C'
[INFO] [1669550359.623382]: Executing State C...
[INFO] [1669550360.118600]: State machine terminating 'C':'SUCCESS':'end'
[INFO] [1669550360.120230]: State A Attempted 3 Times
[INFO] [1669550360.122563]: State B Attempted 1 Times
[INFO] [1669550360.123611]: State C Attempted 1 Times
jacques@jacques-VirtualBox:~$
```

Since each success is random, you will likely get different results each time you run the script!

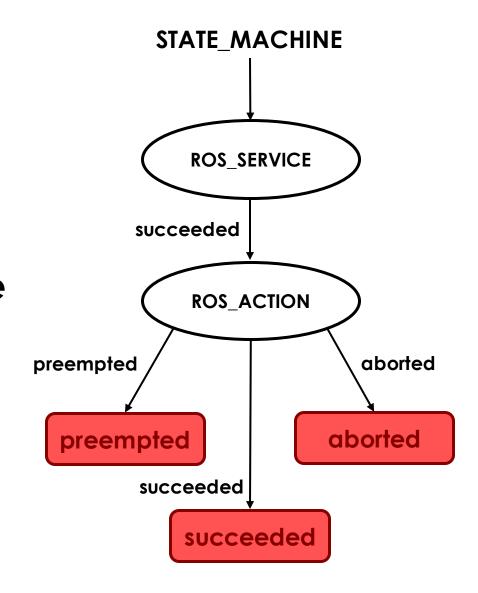
```
[INFO] [1669550536.849852]: State A Attempted 9 Times
[INFO] [1669550536.853074]: State B Attempted 3 Times
[INFO] [1669550536.855435]: State C Attempted 2 Times
```

SMACH and ROS (1)

- As mentioned previously, SMACH has an extension for ROS that allows us to integrate our FSMs into the rest of the ROS network, e.g. for calling Services and Actions
- We could take the naïve approach of setting up Service and Action Clients in the same way as Lectures 4/5 within the execute_cb method...
- ...but SMACH_ROS provides a more sophisticated approach, with specialised Service and Action Client states

SMACH and ROS (2)

- We shall demonstrate this by implementing a FSM that calls our ExampleService and uses the response as a goal for ExampleAction
- In tutorial_script's scripts folder, create the file Example_SM_ROS.py
- Find the script in our GitHub and copy it into your file
- Let's have a look at the code...



SMACH and ROS (3)

```
Example ROS Service Client State:
def request cb(userdata, request): # This is called upon transitioning to the service client state
    request = ExampleServiceRequest()
    request.input number = random.randint(1,3)
    return request # This is used as the service request
def response cb(userdata, response): # This is called upon receiving a response from the server
    userdata.service response = response.output number
    return 'succeeded' # This is used as the state outcome
# Initialise the state
smach.StateMachine.add('ROS SERVICE',
                        smach ros.ServiceState('ExampleService',
                        ExampleService,
                        request cb=request cb,
                        response cb=response cb,
                        output keys=['service response']),
                        transitions={'succeeded' : 'ROS ACTION'})
```

 Here is our implementation for the Service Client; note how we only need to define callbacks for the request and response

SMACH and ROS (4)

```
# Example ROS Action Client State:
def goal cb(userdata, goal): # This is called upon transitioning to the action client state
    goal = ExampleGoal()
    goal.seconds requested = userdata.service response
    return goal # This is used as the action goal
def result cb(userdata, status, result): # This is called upon receiving a result from the server
    if status == actionlib.GoalStatus.SUCCEEDED: # Check goal status
        return 'succeeded' # This is used as the state outcome
    elif status == actionlib.GoalStatus.PREEMPTED:
        return 'preempted'
    else:
        return 'aborted'
# Initialise the state
smach.StateMachine.add('ROS ACTION',
                        smach ros.SimpleActionState('example as',
                        ExampleAction,
                        goal cb=goal cb,
                        result cb=result cb,
                        input keys=['service response']),
                        transitions={'succeeded' : 'succeeded',
                        'preempted' : 'preempted',
                        'aborted' : 'aborted'})
```

 Here is our implementation for the Action Client; again, we only need to define callbacks for the goal and result

Passing Data Between States (5)

 Give your script permissions, then run roscore in one terminal, run the Example Service and Action Servers, and finally your script:

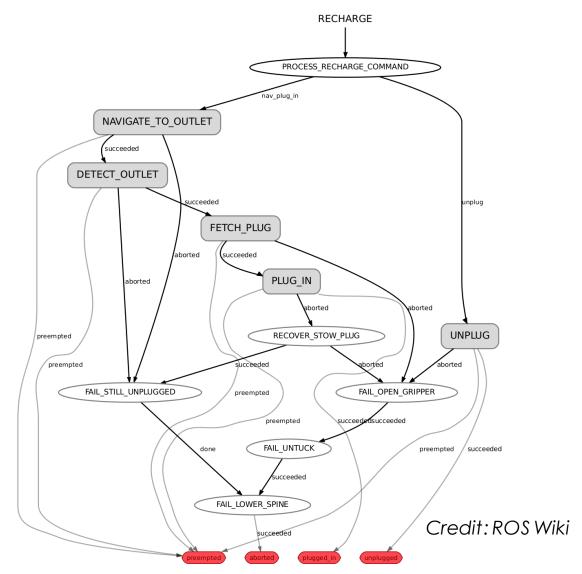
```
ON':'succeeded':'succeeded
                                                                  jacques@jacques-VirtualBox:~$ rosrun tutorial scripts Example S
PARAMETERS
  /rosdistro: noetic
                                                                  M ROS.py
                                                                  [INFO] [1669548378.177058]: State machine starting in initial s
 * /rosversion: 1.15.14
                                                                  tate 'ROS SERVICE' with userdata:
NODES
                                                                          ['service response']
                                                                  [INFO] [1669548378.192204]: State machine transitioning 'ROS SE
                                                                 RVICE':'succeeded'-->'ROS ACTION'
auto-starting new master
process[master]: started with pid [3001]
                                                                   WARN] [1669548378.193041]: Still waiting for action server 'e>
ROS MASTER URI=http://jacques-VirtualBox:11311/
                                                                  ample as' to start... is it running?
                                                                  [INFO] [1669548378.269675]: Connected to action server 'example
setting /run id to 68008502-6e44-11ed-9bc4-a9e2193494ef
process[rosout-1]: started with pid [3012]
                                                                  [INFO] [1669548382.283890]: State machine terminating 'ROS ACTI
started core service [/rosout]
                                                                 ON':'succeeded':'succeeded'
                                                                  jacques@jacques-VirtualBox:~$ 🗌
                       jacques@jacques-VirtualBox: ~ 62x15
 acques@jacques-VirtualBox:~$ rosrun tutorial scripts Example
                                                                 quested: 1) - Performing Action...
                                                                 [INFO] [1669547795.969307]: SUCCESS - Action Complete after 1 s
Service Server.py
[INFO] [1669547620.554779]: Order Received: 2
                                                                  econds
[INFO] [1669547620.556063]: Processed Order: 4
                                                                  [INFO] [1669547805.480103]: Action Request Received (seconds re
[INFO] [1669547794.963844]: Order Received: 1
                                                                 quested: 9) - Performing Action...
                                                                 [INFO] [1669547814.481959]: SUCCESS - Action Complete after 9 s
[INF0] [1669547794.965120]: Processed Order: 1
                                                                  econds
[INFO] [1669547805.445374]: Order Received: 3
[INFO] [1669547805.446533]: Processed Order: 9
                                                                  [INFO] [1669548378.271372]: Action Request Received (seconds re
                                                                 quested: 4) - Performing Action...
[INFO] [1669548378.189768]: Order Received: 2
                                                                 [INFO] [1669548382.280699]: SUCCESS - Action Complete after 4 s
[INFO] [1669548378.191037]: Processed Order: 4
                                                                  econds
```

Closing Thoughts: Fault-Tolerant FSMs (1)

- When you're creating a FSM for your robot, you need to make sure you've covered every possible outcome, even the undesirable ones
 - What if your robot fails to move to a target?
 - What if your object detection software can't find the object you're looking for?
- Your FSM must have fallbacks in place for when things inevitably don't go to plan – this is fault-tolerant FSM design
 - This process will likely take up the most time when you design FSMs!

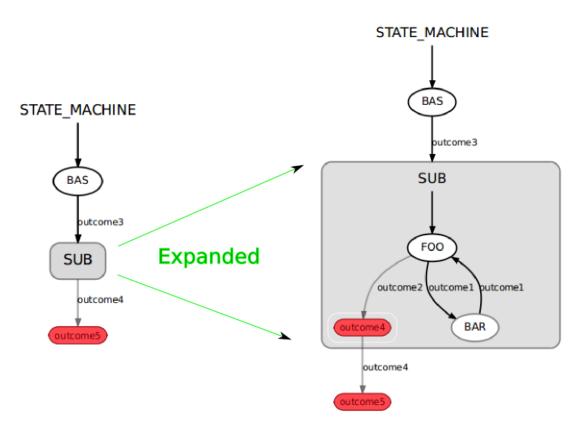
Closing Thoughts: Fault-Tolerant FSMs (2)

- This is an example FSM used as the behaviour for a robot charging itself at a standard outlet
- The states each call a ROS
 Action, and their outcomes
 depend on the result
- Observe how the FSM is built to handle task failure robustly



Closing Thoughts: Hierarchical FSMs

- You can nest a FSM within another FSM; the result is called a Hierarchical FSM
- The nested FSM essentially acts as a state within the parent FSM
- This allows you to develop FSMs for simple behaviours and use these as building blocks for more complex ones
- SMACH provides full functionality for this!



An example Hierarchical FSM Credit: ROS Wiki

Closing Thoughts: SMACH Tutorials

- SMACH and SMACH_ROS have extensive documentation and plenty of tutorials on the ROS Wiki; I suggest checking this out if you want to learn more about using FSMs!
- Link: http://wiki.ros.org/smach/Tutorials

Summary

We covered:

- Overview for finite state machines and SMACH
- Creating a simple SMACH state machine in Python
- Passing data between states
- Calling ROS Services/Actions from SMACH state machines

Homework: Revise the contents of the lecture, go over the code and make sure you can follow what's going on







Thank You!

This was the final lecture, I really hope you enjoyed this course!

Any Questions? Contact jacques.cloete@trinity.ox.ac.uk