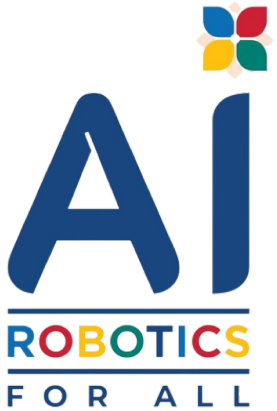
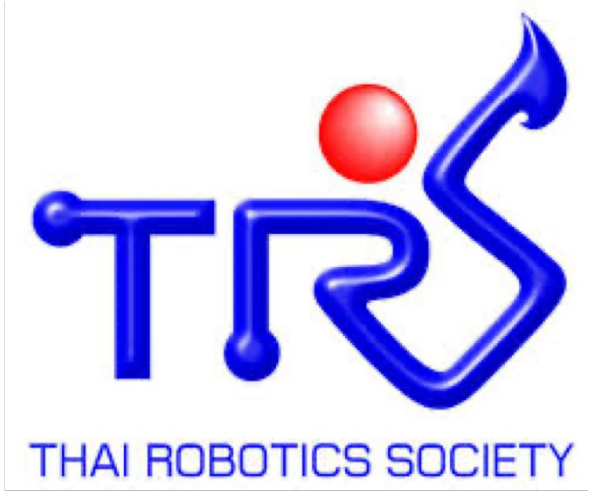

AI
ROBOTICS
FOR ALL


AiAT
ARTIFICIAL INTELLIGENCE ASSOCIATION OF THAILAND


KU
KASETSART
UNIVERSITY


THAI ROBOTICS SOCIETY


KU
robotcitizens.org

Requirement

- Python - หากใครเขียนไม่ได้ ให้ไปเรียนห้อง 0501 นะครับ
- ROS
- Navigation
- Robot manipulation

Slide

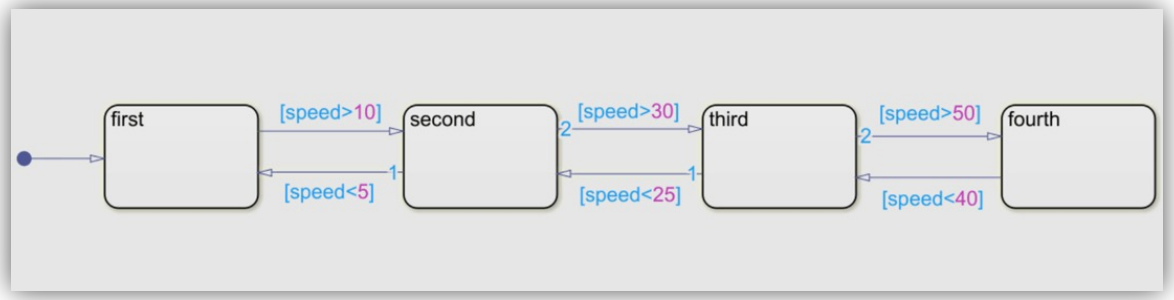
https://kasets.art/aeVUY1

Finite state machine

Noppanut Thongton

Finite state machine

A state machine (or finite state machine) is a representation of an event-driven, reactive system that transitions from one state to another if the condition that controls the change is met. State machines were conventionally used to describe computing systems, but they have been expanded to model complex logic in dynamic systems such as aircraft, automobiles, robots, and mobile phones.



Finite state machine

There are two main types of state diagrams:

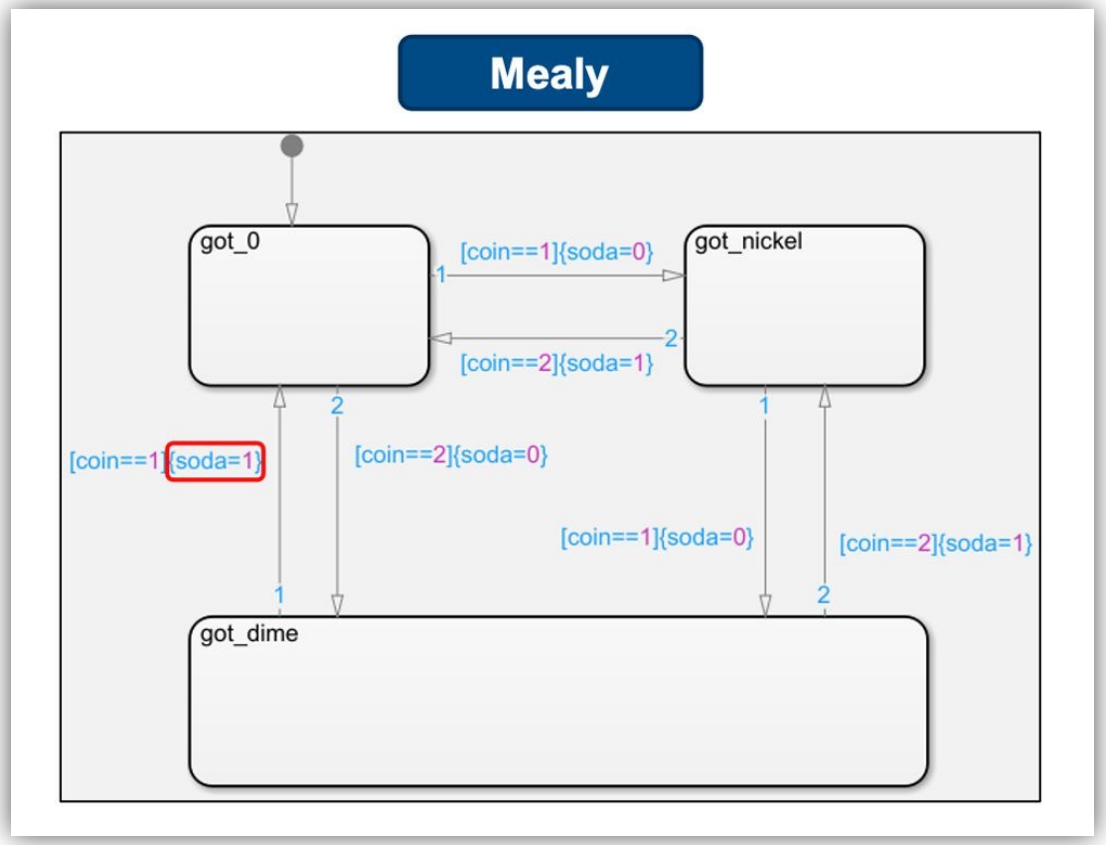
- **Mealy**

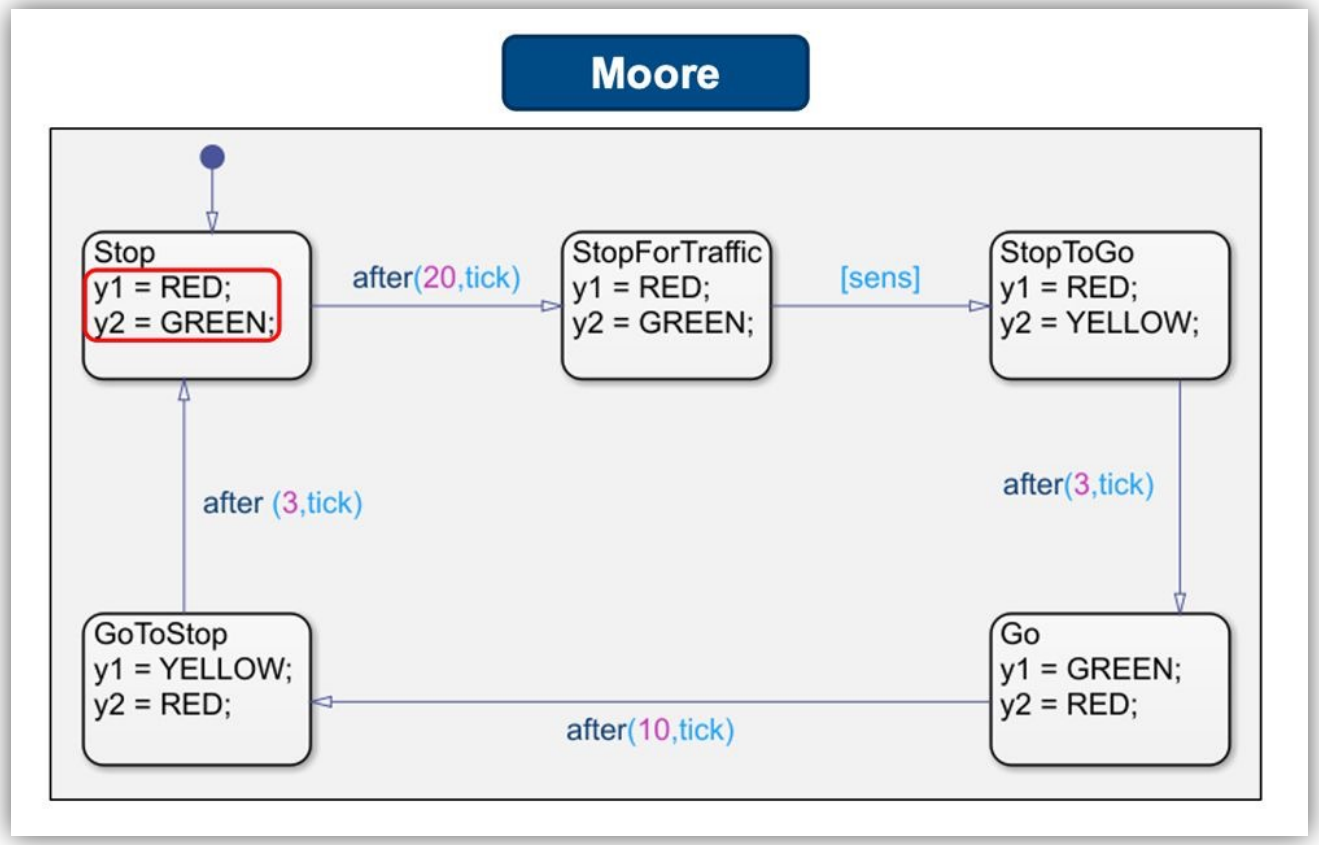
  State machine outputs depend not only on the states but also on inputs to the system, represented by defining machine outputs in the transitions.

- **Moore**

  State machine outputs depend only on the state of the system, represented by defining machine outputs on the states.

Finite state machine



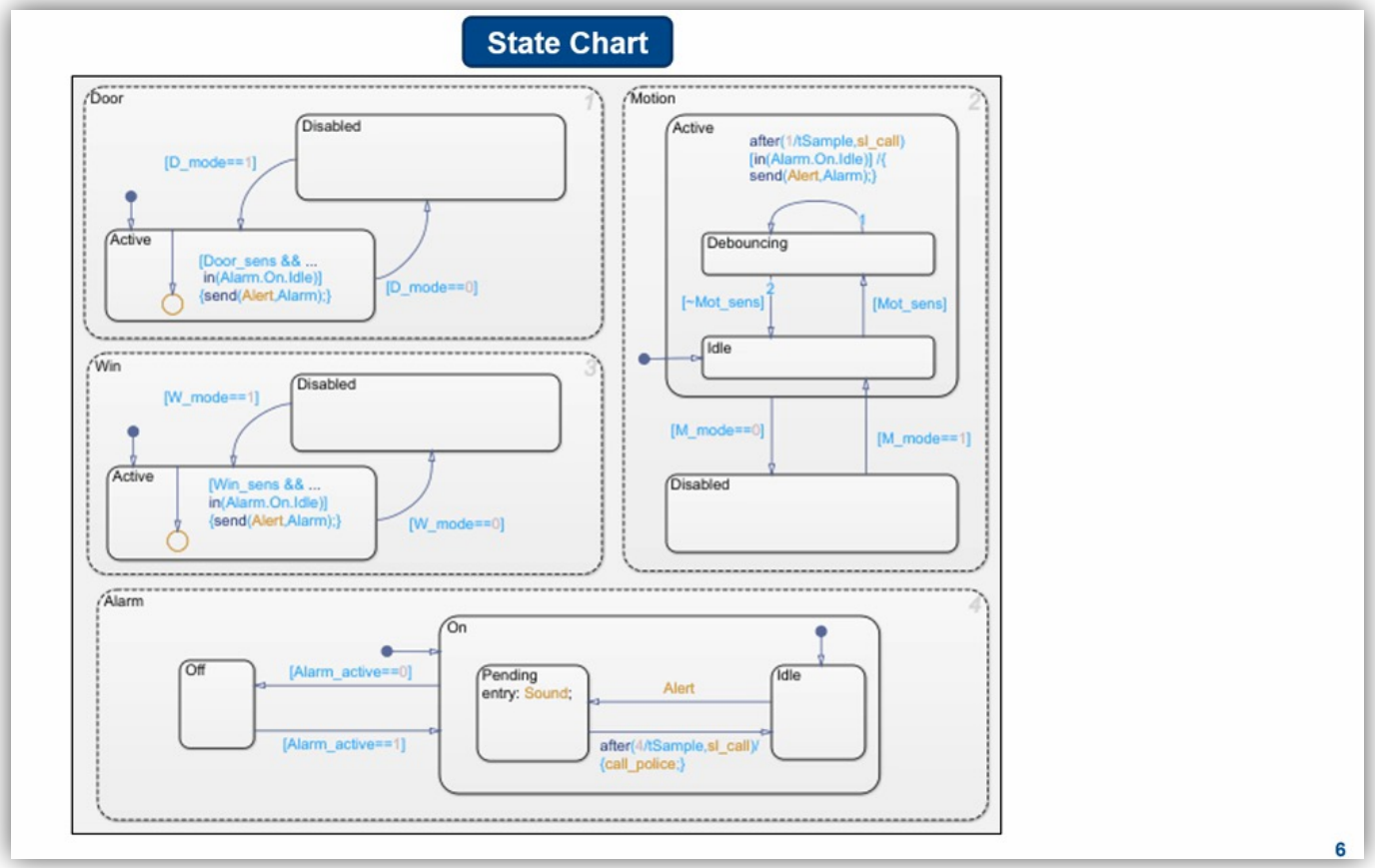


Finite state machine

To create state machines that emulate a complex software component, the basic building blocks of state diagrams are not enough. The following additional capabilities are required to capture intricate details of the system efficiently:

- **Hierarchy** – Introduces parent state(s) and further structures the design
- **Parallelism or Orthogonality** – Allows for a single diagram to include multiple states operating simultaneously
- **Event Broadcasting** – Allows for information exchange between two independent states or state machines

Finite state machine



SMACH

Library for implement finite state machine in ROS

SMACH

SMACH is a task-level architecture for rapidly creating **complex robot behavior**. At its core, SMACH is a ROS-independent Python library to build **hierarchical state machines**. SMACH is a new library that takes advantage of very old concepts in order to quickly create robust robot behavior with maintainable and modular code.

SMACH – Concept

Outcome Semantics

For all SMACH containers, the interface to contained states is defined via **state outcomes**. A state's potential outcomes are a property of the state instance, and must be declared before it is executed. If a SMACH plan is written by hand, all potential outcomes are declared on construction and the consistency of the state transitions can easily be checked without executing it.

State outcomes may cause different things to happen in different types of containers, but what happens after an outcome is emitted is irrelevant from perspective of the state. In this sense, outcomes can be considered "local" to a given state.

SMACH – Concept

Outcome Semantics

For example, a state might provide outcomes such as 'succeeded', 'aborted', or 'preempted' and as far as the state is concerned, this is the way that it interacts with the task-level state flow. In the context of a state machine, these outcomes would be associated with other states, thus forming a transition. In the context of another container, however, they might be treated differently. Outcomes simply acts as a common interface between states and containers.

SMACH – Concept

User Data

SMACH containers each have a flat database used to coordinate and pass data between different states. This becomes useful when states compute some result or return some sensor information. This allows such data to be held at the execution level, and made available to other tasks or procedures.

Similarly, to outcomes, the userdata keys that are set and retrieved in each state are described by the state's input keys and output keys. These are also a property of the state instance, and must be declared prior to execution. The declaration of all of these interactions prevents errors and aides in debugging when the SMACH plans become complex.

SMACH – Concept

Preemption

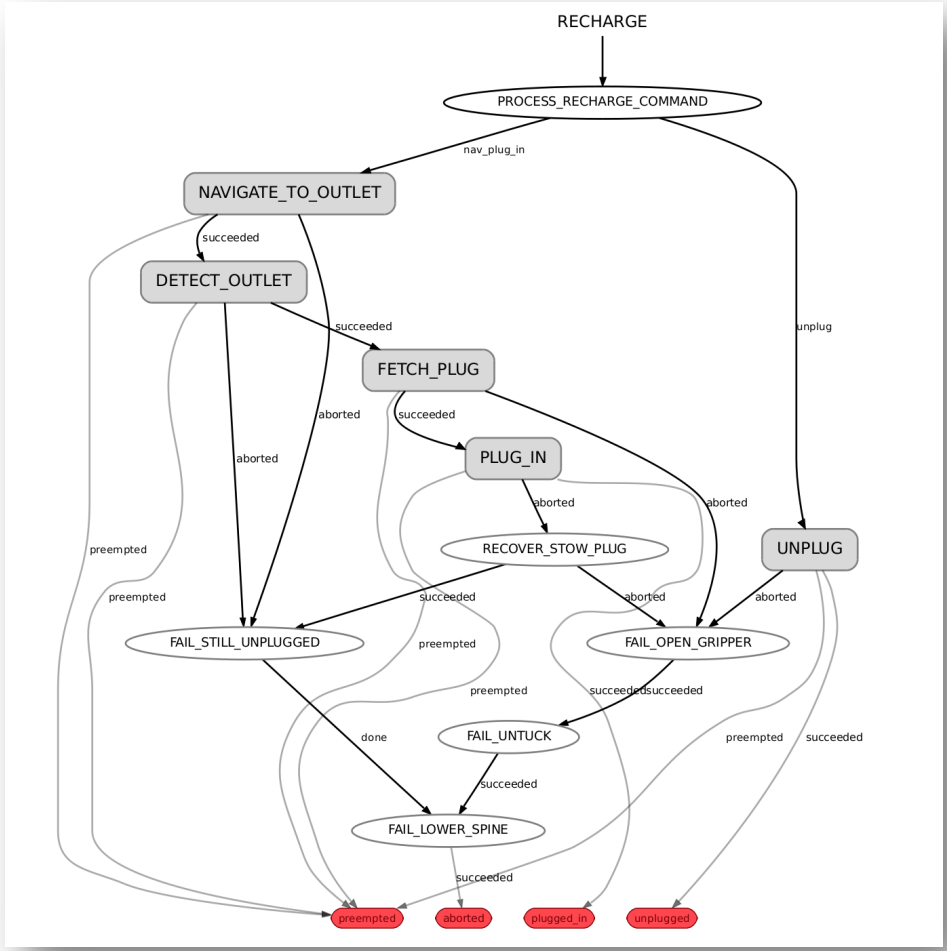
Preemption propagation is built into SMACH. The State base class includes an interface for coordinating preempt requests between containers and contained states. Each container type has its own well-defined behavior for responding to a preempt request. This allows the system to both cleanly respond to termination signals from an end user and be able to be canceled programmatically by a higher-level executive.

SMACH – Concept

Preemption

Preemption propagation is built into SMACH. The State base class includes an interface for coordinating preempt requests between containers and contained states. Each container type has its own well-defined behavior for responding to a preempt request. This allows the system to both cleanly respond to termination signals from an end user and be able to be canceled programmatically by a higher-level executive.

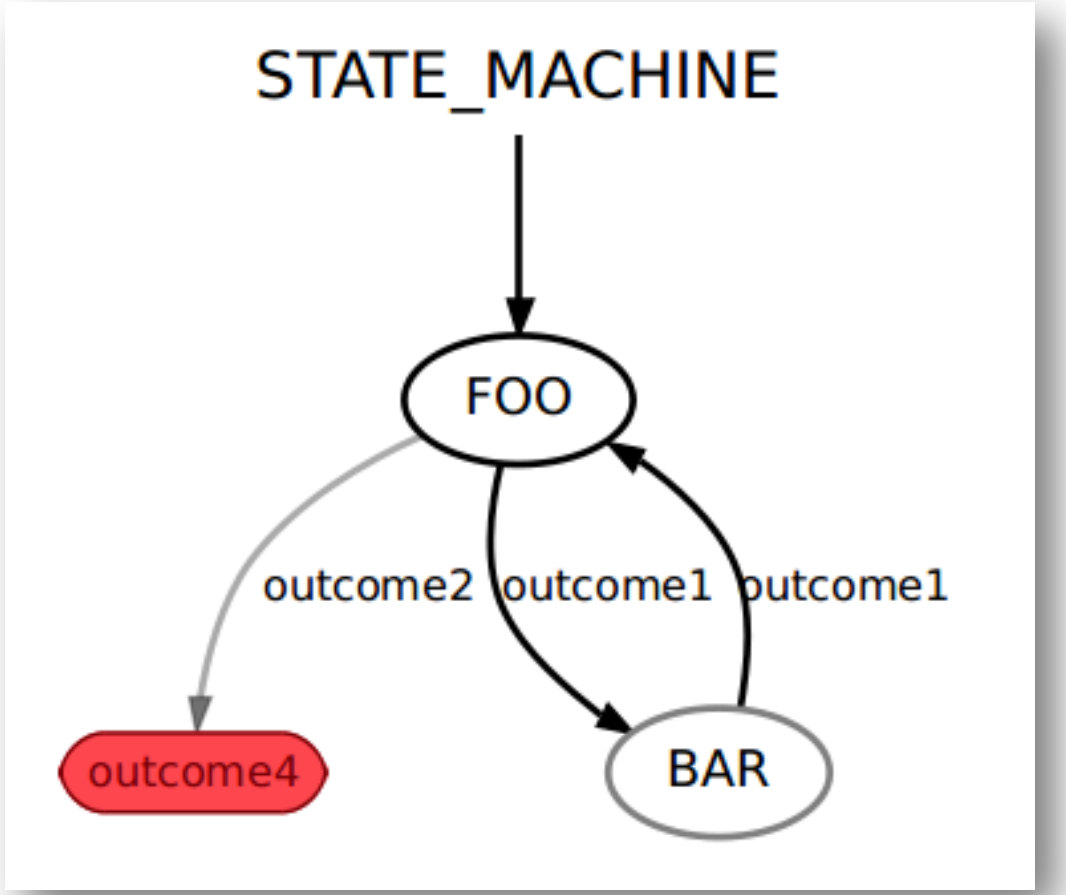
SMACH


RECHARGE
PROCESS_RECHARGE_COMMAND
nav_plug_in
NAVIGATE_TO_OUTLET
succeeded
DETECT_OUTLET
succeeded
FETCH_PLUG
unplug
aborted
succeeded
aborted
PLUG_IN
aborted
aborted
preempted
RECOVER_STOW_PLUG
UNPLUG
preempted
succeeded
aborted
aborted
FAIL_STILL_UNPLUGGED
preempted
FAIL_OPEN_GRIPPER
preempted
succeeded
succeeded
done
FAIL_UNTUCK
preempted
succeeded
succeeded
FAIL_LOWER_SPINE
succeeded
preempted
aborted
plugged_in
unplugged

SMACH – Installation

```
$ sudo apt install ros-noetic-smach ros-noetic-smach-ros
```

SMACH - Python



SMACH - Python

Create new package name “ your_smach ”

```
$ roscd
```

```
$ cd ../src
```

```
$ catkin_create_pkg your_smach rospy roscpp std_msgs smach
```

```
$ cd ..
```

```
$ catkin_make
```

SMACH - Python

Create new file name “smach-python-tutorial.py ”

```
$ rospack profile
```

```
$ roscd your_smach
```

```
$ mkdir script && cd script
```

```
$ gedit smach-python-tutorial.py
```

SMACH - Python

Code

SMACH - Python

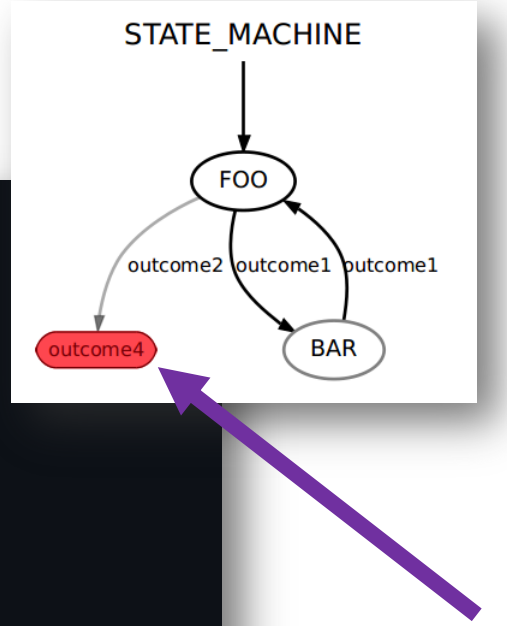
```
import rospy
import smach
import smach_ros
```

SMACH - Python

```python
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['outcome4'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR', 'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome1':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```

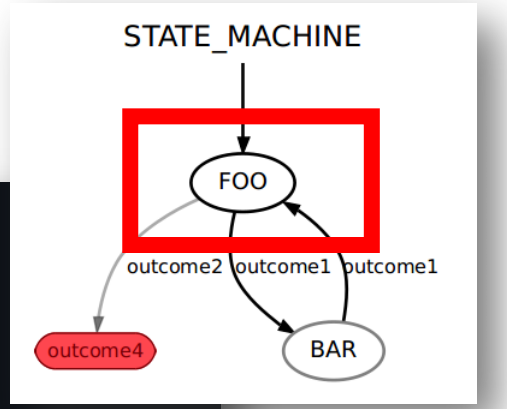
SMACH - Python

```python
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachin (outcomes=['outcome4'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR', 'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome1':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```



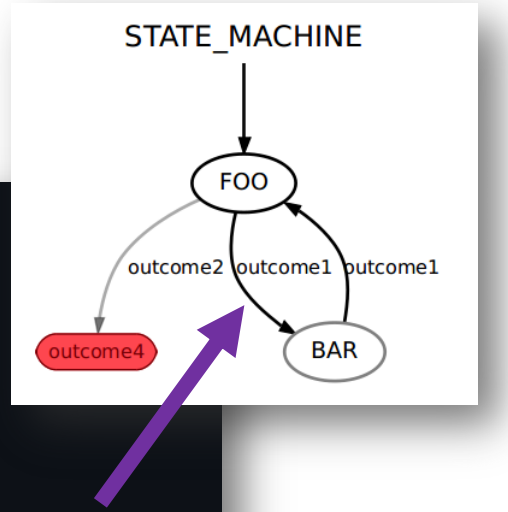
SMACH - Python

```python
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['outcome4'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR', 'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome1':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```



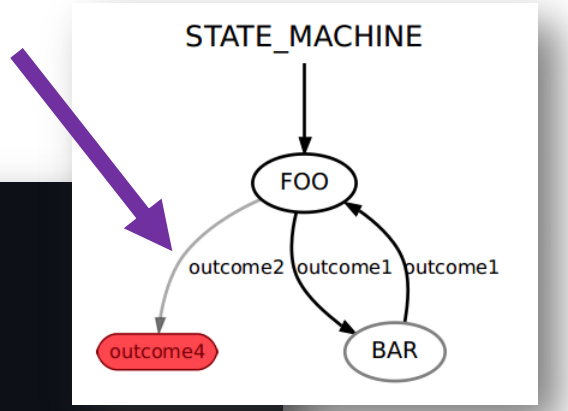
SMACH - Python

```python
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['outcome4'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR', 'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome1':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```



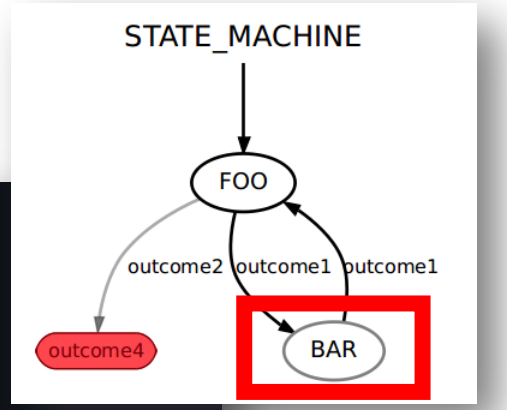
SMACH - Python

```python
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['outcome4'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR'  'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome1':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```



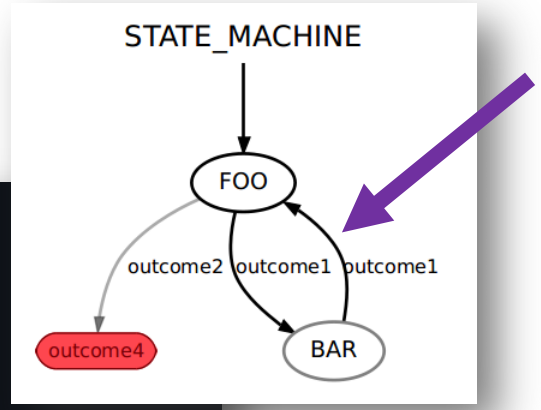
SMACH - Python

```python
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['outcome4'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR', 'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome1':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```



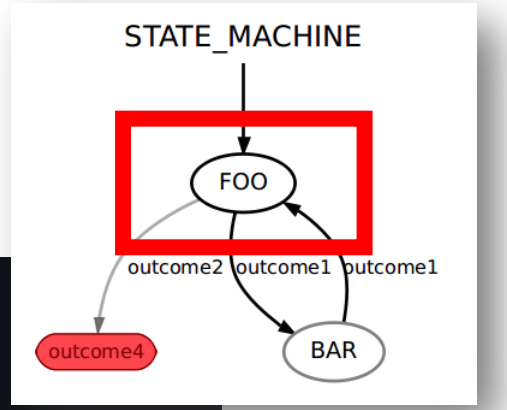
SMACH - Python

```python
def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['outcome4'])

    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('FOO', Foo(),
                               transitions={'outcome1':'BAR', 'outcome2':'outcome4'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome1':'FOO'})

    # Execute SMACH plan
    outcome = sm.execute()
```

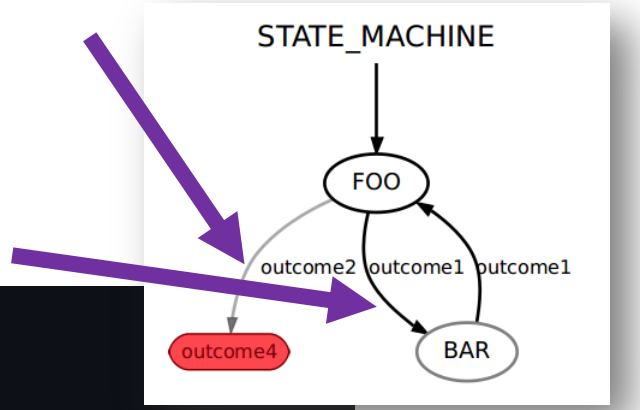


SMACH - Python

```python
class Foo(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome1','outcome2'])
        self.counter = 0

    def execute(self, userdata):
        rospy.loginfo('Executing state FOO')
        rospy.loginfo(f"self.counter = {self.counter}")
        if self.counter < 3:
            self.counter += 1
            return 'outcome1'
        else:
            return 'outcome2'
```



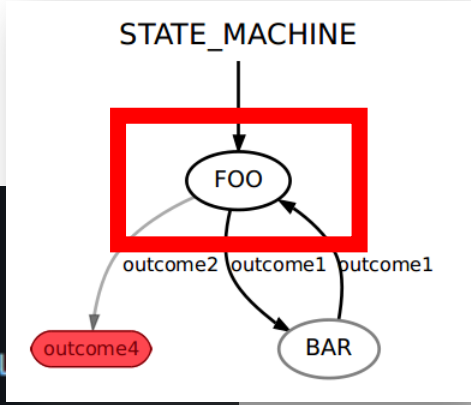
SMACH - Python



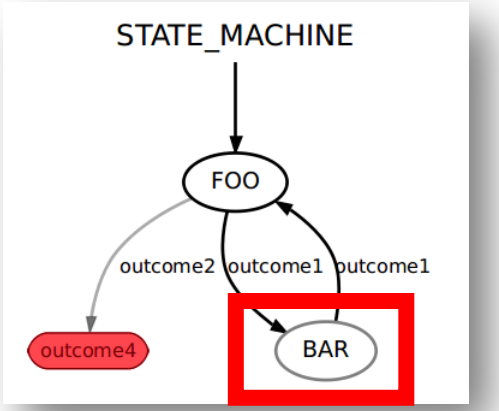

```python
class Foo(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome1','outcome2'])
        self.counter = 0

    def execute(self, userdata):
        rospy.loginfo('Executing state FOO')
        rospy.loginfo(f"self.counter = {self.counter}")
        if self.counter < 3:
            self.counter += 1
            return 'outcome1'
        else:
            return 'outcome2'
```

SMACH - Python

```python
class Foo(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome1','ou
        self.counter = 0

    def execute(self, userdata):
        rospy.loginfo('Executing state FOO')
        rospy.loginfo(f"self.counter = {self.counter}")
        if self.counter < 3:
            self.counter += 1
            return 'outcome1'
        else:
            return 'outcome2'
```

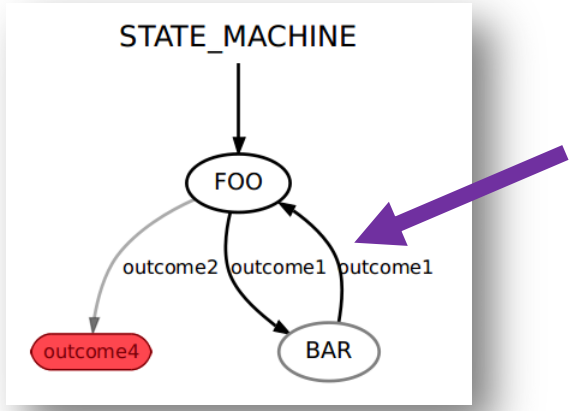


SMACH - Python




```python
class Bar(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome1'])

    def execute(self, userdata):
        rospy.loginfo('Executing state BAR')
        return 'outcome1'
```

SMACH - Python




```python
class Bar(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome1'])

    def execute(self, userdata):
        rospy.loginfo('Executing state BAR')
        return 'outcome1'
```

SMACH - Python

```
$ roscore
```

Open new terminal

```
$ python3 smach-python-tutorial.py
```

SMACH - Python

```
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'
[INFO] [1669785185.312016]: State machine starting in initial state 'FOO' with userdata:
       []
[INFO] [1669785185.312776]: Executing state FOO
[INFO] [1669785185.313364]: self.counter = 0
[INFO] [1669785185.313906]: State machine transitioning 'FOO':'outcome1'-->'BAR'
[INFO] [1669785185.314442]: Executing state BAR
[INFO] [1669785185.315104]: State machine transitioning 'BAR':'outcome1'-->'FOO'
[INFO] [1669785185.315635]: Executing state FOO
[INFO] [1669785185.316137]: self.counter = 1
[INFO] [1669785185.316667]: State machine transitioning 'FOO':'outcome1'-->'BAR'
[INFO] [1669785185.317170]: Executing state BAR
[INFO] [1669785185.317710]: State machine transitioning 'BAR':'outcome1'-->'FOO'
[INFO] [1669785185.318336]: Executing state FOO
[INFO] [1669785185.318873]: self.counter = 2
[INFO] [1669785185.319387]: State machine transitioning 'FOO':'outcome1'-->'BAR'
[INFO] [1669785185.319888]: Executing state BAR
[INFO] [1669785185.320366]: State machine transitioning 'BAR':'outcome1'-->'FOO'
[INFO] [1669785185.320889]: Executing state FOO
[INFO] [1669785185.321383]: self.counter = 3
[INFO] [1669785185.321889]: State machine terminating 'FOO':'outcome2':'outcome4'
```

SMACH - Python

Create custom service

GoToPosition.srv

```
string position_name
---
bool success
```

SMACH - Python

Try to create the smach for your robot with mockup state

```
$ roscd your_smach/script
```

```
$ gedit state-mockup.py
```

SMACH - Python

Code

SMACH - Python

```python
#!/usr/bin/env python3

import rospy
from std_srvs.srv import *
from your_smach.srv import GoToPosition, GoToPositionResponse

class StateMockUp(object):
    def __init__(self) -> None:
        rospy.init_node('state_mockup', anonymous=True)
        rospy.Service('robot/go_to_position', GoToPosition, self.gotopos_callback)
        rospy.Service('robot/do_sth', Empty, self.dosth_callback)

    def gotopos_callback(self, data):
        rospy.loginfo(f'Going to position name: {data.position_name}')
        res = GoToPositionResponse()
        res.success = True
        return res

    def dosth_callback(self, data):
        rospy.loginfo("Doing something")
        return EmptyResponse

if __name__ == "__main__":
    mockup = StateMockUp()
```

SMACH - Python

Try to create the smach for your robot with mockup state

```
$ roscd your_smach/script
```

```
$ gedit state.py
```

SMACH - Python

Code

SMACH - Python

```python
#!/usr/bin/env python3

import rospy
import smach
import smach_ros
from std_srvs.srv import *
from your_smach.srv import GoToPosition, GoToPositionRequest

class DoSth(smach.State):
    def __init__(self, outcomes=['success', 'fail']):
        super().__init__(outcomes)
        self.dosth_ser = rospy.ServiceProxy('robot/do_sth', Empty)

    def execute(self, ud):
        rospy.loginfo("Executing state DoSth")
        self.dosth_ser()
        return 'success'

class GoToPosition(smach.State):
    def __init__(self, outcomes=['success', 'fail']):
        super().__init__(outcomes)
        self.gotopos = rospy.ServiceProxy('robot/go_to_position', GoToPosition)

    def execute(self, ud):
        rospy.loginfo("Executing state GoToPosition")
        req = GoToPositionRequest()
        req.position_name = 'home'
        self.gotopos(req)
        return 'success'

class RobotState(object):
    def __init__(self) -> None:
        rospy.init_node('robot_state', anonymous=True)
        sm = smach.StateMachine(outcomes=['---finish---'])

        with sm:
            smach.StateMachine.add('DoSth1', DoSth(),
                                   transitions={'success':'Gotoposition', 'fail':'DoSth1'})
            smach.StateMachine.add('Gotoposition', GoToPosition(),
                                    transitions={'success':'DoSth2', 'fail':'Gotoposition'})
            smach.StateMachine.add('DoSth2', DoSth(),
                                   transitions={'success':'---finish---', 'fail':'DoSth2'})
        outcome = sm.execute()

if __name__ == "__main__":
    RobotState()
```

In-class Assignment

- **Define class by requirement below**
  - Class NavToPoint -> For navigation to saved position
  - Class MoveArmToPose -> For move arm to saved position
  - Class OpenArmGripper -> For open gripper
  - Class CloseArmGripper -> For close gripper
- **Command**
  - Save 2 positions: Living Room, Kitchen
  - Save 2 arm positions: Hold, Grasp
  - Robot start at living room and go to kitchen to grasp bottle from person and bring bottle back to living room