EE346 lab7 report

王安哲 11912417 刘广骁 12011726

Environment

computer operating system: **ubuntu18.04**.

ROS version: **ROS melodic**

Hardware: **Turtlebot3 burger**

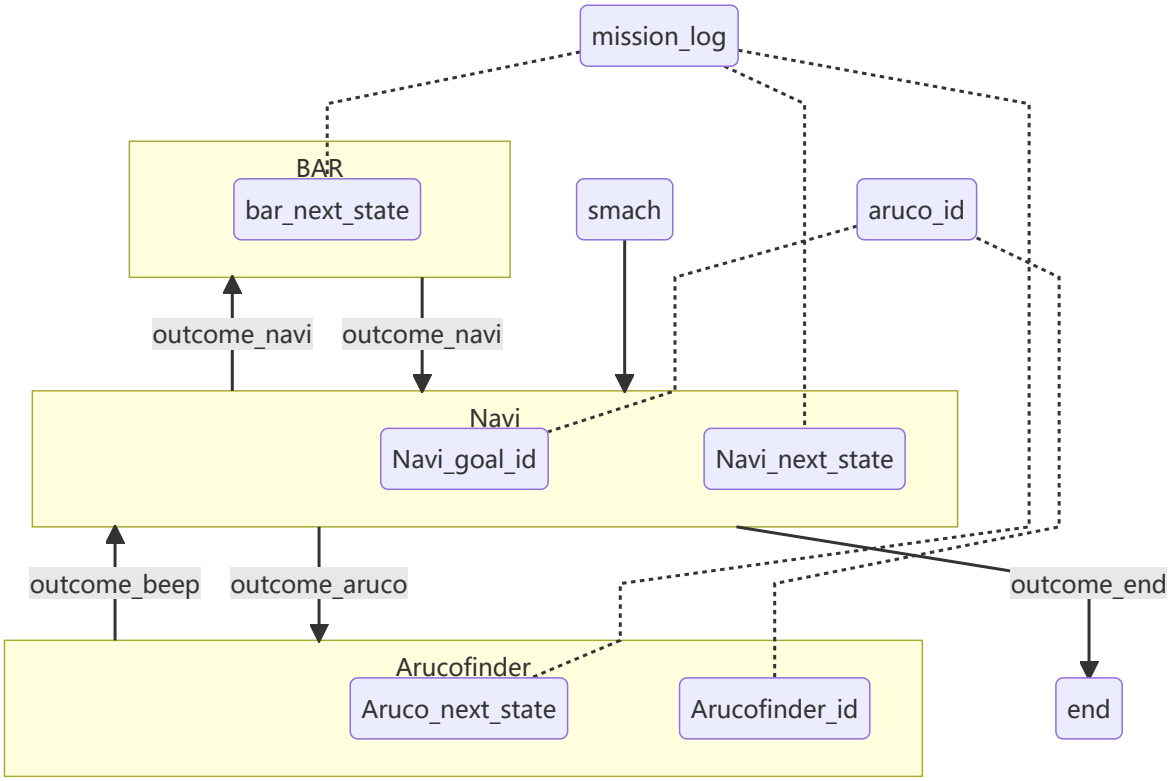
Design Objectives

Under the premise of getting the map of the current venue, we can maketurtlebot3 reach different points, and after that reach the specified location, get the aruco code information, park, and reach the specified point according to the aruco id.

Design Ideas

General idea is implemented through state machines.

To begin, we first navigate the turtlebot3 to defined points,the state machine is in Navi's state. After reaching the specified point, it enters the BAR state and then re-enters the Navi state to navigate to the next point. When all the points are reached in sequence, the turtlebot enters the Arucofinder state, rotates in place and acquires images from the camera, finds the aruco code, approaches it and stops, reads the id of the aruco, and navigates to the corresponding point according to the id.



Use

First we use ssh to connect the turtlebot3 (Raspberry Pi) with our PC, in the following, I will use rpi to refer to turtlebot.

Open **roscore** in PC.

```
roscore
```

Then launch the camera and drive system (include lidar) in **rpi**.

```
roslaunch raspinode camera2_320*240.launch
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Create arucofinder nodes in **PC**. These nodes correspond to cases with ids 1 to 4. In the next steps, we will subscribe to these nodes.

```
roslaunch catkin_ws/src/aruco-ros/launch/aruco1.launch
roslaunch catkin_ws/src/aruco-ros/launch/aruco2.launch
roslaunch catkin_ws/src/aruco-ros/launch/aruco3.launch
roslaunch catkin_ws/src/aruco-ros/launch/aruco4.launch
```

Then create the node for navigation in **PC**.

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
```

Finally, run our code **FSM_combined.py**

```
#!/usr/bin/env python
import roslib
import rospy
import smach
import smach_ros
import actionlib
# Brings in the .action file and messages used by the move base action
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from geometry_msgs.msg import PoseStamped
from geometry_msgs.msg import Twist
import sound_play
from sound_play.libsoundplay import SoundClient
from std_msgs.msg import String
import numpy as np
import sys
import actionlib
from actionlib_msgs.msg import *
from turtlebot3_msgs.msg import *


class Pose_py():
    def __init__(self):
        self.aruco_x = 0
        self.aruco_z = 0
    def posecallback(self, data):
        self.aruco_z=data.pose.position.z
        self.aruco_x=data.pose.position.x

class Bar(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome_navi'],
                             input_keys=['bar_next_state'])

    def execute(self, userdata):
        rospy.loginfo('Executing state BAR')
        if(userdata.bar_next_state == 0):
            rospy.loginfo('Next_state_Navi')
            return 'outcome_navi'


class Navi(smach.State):
    def __init__(self):
        self.waypoints = [
            [(3.69,1.056,0),(0.0,0.0,0.995,0.09199)],
            [(2.3358,1.472,0),(0,0,0.986,0.08)],
            [(0.62,2.833,0),(0.0,0.0,0.9,0.43)],  # 2
            [(1.765,-0.19,0),(0.0,0.0,0.-0.656,0.753)],
            [(-0.58, -0.86, 0), (0, 0, -0.1, 0.99)],
            [(-0.62, -0.95, 0), (0, 0, -0.1, 0.99)],  # 3
            [(3.15, -2.15, 0), (0, 0, -0.01, 0.9917)],
```

```
            [(3.22,-2.28,0),(0,0,-0.08,0.9917)],  # 4
            [(2.56, 1.4, 0), (0.0, 0.0, 0.08, 0.99)], # before enter the door
            [(4.6585,1.5379,0),(0,0,0.0998,-0.04356)], # 1
            [(4.2,1.42,0),(0,0,0.91,0.41)]
            ]
        smach.State.__init__(self,
                             outcomes=
['outcome_navi','outcome_aruco','outcome_end'],
                             input_keys=['Navi_goal_id'],
                             output_keys=['Navi_next_state'])
        self.cnt = 0

    def execute(self, userdata):
        if userdata.Navi_goal_id>0:
            print("goal is",userdata.Navi_goal_id)
            tmp=0
            if userdata.Navi_goal_id==1:
                tmp=-2
                result=self.movebase_client(self.waypoints[tmp])
            if userdata.Navi_goal_id==2:
                tmp=1
                for i in range(tmp+1):
                    result=self.movebase_client(self.waypoints[i])

            if userdata.Navi_goal_id==3:
                tmp=3
                result=self.movebase_client(self.waypoints[0])
                result=self.movebase_client(self.waypoints[1])
                result=self.movebase_client([(2.3358,1.472,0),(0,0,0.996,0.08)])
                result=self.movebase_client(self.waypoints[3])
                result=self.movebase_client(self.waypoints[4])
            if userdata.Navi_goal_id==4:
                tmp=4
                result=self.movebase_client(self.waypoints[0])
                print(1)
                # result=self.movebase_client(self.waypoints[1])
                result=self.movebase_client([(2.3358,1.472,0),(0,0,0.986,0.08)])
                result=self.movebase_client(self.waypoints[3])
                result=self.movebase_client(self.waypoints[6])
                result=self.movebase_client(self.waypoints[7])
            if result:
                rospy.loginfo("Successfully navigate to aruco maker id")
                return 'outcome_end'

        if self.cnt < len(self.waypoints):
            info_msg = "Start navigating to " + str(self.cnt) + "th point"
            rospy.loginfo(info_msg)
            result = self.movebase_client(self.waypoints[self.cnt])
            if result:
                rospy.loginfo("Successfully navigate to:"+ str(self.cnt)+"th
goals")
            self.cnt += 1
            userdata.Navi_next_state = 0
            return 'outcome_navi'
        else:
```

```
            rospy.loginfo('Navigation completed')
            userdata.Navi_next_state = 1
            return 'outcome_aruco'

    def movebase_client(self, posearray):
        # Create an action client called "move_base" with action definition file
"MoveBaseAction"
        client = actionlib.SimpleActionClient('move_base',MoveBaseAction)
        # Waits until the action server has started up and started listening for
goals.
        client.wait_for_server()
        # Creates a new goal with the MoveBaseGoal constructor
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = "map"
        goal.target_pose.header.stamp = rospy.Time.now()
        # Move 0.5 meters forward along the x axis of the "map" coordinate frame
        goal.target_pose.pose.position.x = posearray[0][0]
        goal.target_pose.pose.position.y = posearray[0][1]
        goal.target_pose.pose.position.z = posearray[0][2]
        # No rotation of the mobile base frame w.r.t. map frame
        goal.target_pose.pose.orientation.w = posearray[1][3]
        # Sends the goal to the action server.
        client.send_goal(goal)
        # Waits for the server to finish performing the action.
        wait = client.wait_for_result()
        # If the result doesn't arrive, assume the Server is not available
        if not wait:
            rospy.logerr("Action server not available!")
            rospy.signal_shutdown("Action server not available!")
        else:
        # Result of executing the action
            return client.get_result()


#define state ArucoFinder
class ArucoFinder(smach.State):
    def __init__(self):
        self.pp1 = Pose_py(1)
        self.pp2 = Pose_py(2)
        self.pp3 = Pose_py(3)
        self.pp4 = Pose_py(4)
        smach.State.__init__(self, outcomes=['outcome_beep'],
                             output_keys=['Arucofinder_id','Aruco_next_state'])
        # rospy.Subscriber("/aruco_single/pose",PoseStamped,pp.posecallback)
        self.pub = rospy.Publisher('/cmd_vel', Twist,queue_size=10)
        self.msg=Twist()
        # pp = Pose_py()
        self.rate = rospy.Rate(10)
        self.finishstop=False
        self.havefindid=False

    def execute(self, userdata):
        print("start finding aruco code")
        rospy.Subscriber("/aruco1/pose",PoseStamped,self.pp1.posecallback)
        rospy.Subscriber("/aruco2/pose",PoseStamped,self.pp2.posecallback)
```

```
        rospy.Subscriber("/aruco3/pose",PoseStamped,self.pp3.posecallback)
        rospy.Subscriber("/aruco4/pose",PoseStamped,self.pp4.posecallback)
        userdata.Arucofinder_id = 0
        userdata.Aruco_next_state = 1
        selected=0
        print(selected)
        aruco0=self.pp1
        while (not rospy.is_shutdown() and not self.finishstop):
            if (not self.havefindid):
                if (self.pp1.aruco_z!=0 and abs(self.pp1.aruco_x)<0.5):
                    userdata.Arucofinder_id=1
                    selected=1
                    print("id is 1")

                elif (self.pp2.aruco_z!=0 and abs(self.pp2.aruco_x)<0.5):
                    userdata.Arucofinder_id=2
                    aruco0=self.pp2
                    selected=2
                    print("id is 2")

                elif(self.pp3.aruco_z!=0 and abs(self.pp3.aruco_x)<0.5):
                    userdata.Arucofinder_id=3
                    aruco0=self.pp3
                    selected=3
                    print("id is 3")


                elif(self.pp4.aruco_z!=0 and abs(self.pp4.aruco_x)<0.5):
                    userdata.Arucofinder_id=4
                    aruco0=self.pp4
                    selected=4
                    print("id is 4")

                    soundhandle = SoundClient()
                    rospy.sleep(1)
                    num = int(1)
                    volume = 1.0
                    rospy.loginfo('Playing sound %i.' % num)

                    for i in range(4):
                        soundhandle.play(num, volume)
                        rospy.sleep(1)

                    rospy.sleep(1)


                if(not selected==0):
                    self.havefindid=True
                else:
                    self.msg.angular.z = -0.35
                    self.msg.linear.x = 0
                    self.pub.publish(self.msg)
                    self.rate.sleep()
                continue
```

```
            elif aruco0.aruco_z>0.43 and abs(aruco0.aruco_x)<0.3:
                ratio = (aruco0.aruco_z-0.20)
                self.msg.angular.z=-0.8*(aruco0.aruco_x)
                self.msg.linear.x=max(0.15*(ratio),0.02)
                self.pub.publish(self.msg)
                self.rate.sleep()
            else:
                self.msg.angular.z=0.0
                self.msg.linear.x=0.0
                self.pub.publish(self.msg)
                self.rate.sleep()

                self.msg.angular.z=0
                self.msg.linear.x=-0.03

                for i in range(5):
                    self.pub.publish(self.msg)
                    self.rate.sleep()
                self.finishstop=True
                break
        return 'outcome_beep'

if __name__ == '__main__':
    rospy.init_node('lab7')
    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['end']) #
    sm.userdata.mission_log = 0  # 0:navi 1:aruco
    sm.userdata.aruco_id = 0  #
    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('Navi',Navi(),
                               transitions={'outcome_navi':'BAR',
'outcome_aruco':'Arucofinder', 'outcome_end':'end'},
                               remapping=
{'Navi_goal_id':'aruco_id','Navi_next_state':'mission_log'})
        smach.StateMachine.add('Arucofinder', ArucoFinder(),
                               transitions={'outcome_beep': 'Navi'},
                               remapping=
{'Arucofinder_id':'aruco_id','Aruco_next_state':'mission_log'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome_navi':'Navi'},
                               remapping={'bar_next_state':'mission_log'})

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

Code explanation

Pose_py

We created the class Pose_py, which contains only 2 member variables. and defined a callback function. It is used to read the information from aruco.

```
class Pose_py():
    def __init__(self):
        self.aruco_x = 0
        self.aruco_z = 0
    def posecallback(self, data):
        self.aruco_z=data.pose.position.z
        self.aruco_x=data.pose.position.x
```

BAR

BAR This class initializes the state of the state machine. This state can be considered as an intermediate state between two points of navigation.

```
class Bar(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['outcome_navi'],
                             input_keys=['bar_next_state'])
    def execute(self, userdata):
        rospy.loginfo('Executing state BAR')
        if(userdata.bar_next_state == 0):
            rospy.loginfo('Next_state_Navi')
            return 'outcome_navi'
```

Navi

Firstly Navi initialization includes a sequence to record the information of different point poses and initializes the state of the state machine. cnt is here in order to record the serial number of the currently located point.

```
class Navi(smach.State):
    def __init__(self):
        self.waypoints = [
            [(3.69,1.056,0),(0.0,0.0,0.995,0.09199)],
            [(2.3358,1.472,0),(0,0,0.986,0.08)],
            [(0.62,2.833,0),(0.0,0.0,0.9,0.43)],  # 2
            [(1.765,-0.19,0),(0.0,0.0,0.-0.656,0.753)],
            [(-0.58, -0.86, 0), (0, 0, -0.1, 0.99)],
            [(-0.62, -0.95, 0), (0, 0, -0.1, 0.99)],  # 3
            [(3.15, -2.15, 0), (0, 0, -0.01, 0.9917)],
            [(3.22,-2.28,0),(0,0,-0.08,0.9917)],  # 4
            [(2.56, 1.4, 0), (0.0, 0.0, 0.08, 0.99)], # before enter the door
            [(4.6585,1.5379,0),(0,0,0.0998,-0.04356)], # 1
            [(4.2,1.42,0),(0,0,0.91,0.41)]
            ]
        smach.State.__init__(self,
                             outcomes=
['outcome_navi','outcome_aruco','outcome_end'],
```

```
                            input_keys=['Navi_goal_id'],
                            output_keys=['Navi_next_state'])
        self.cnt = 0
```

'movebase_client' is a function that, after passing in an array containing the point's pose, enables the turtlebot to reach the target point through the action mechanism.

```
    def movebase_client(self, posearray):
        # Create an action client called "move_base" with action definition file
"MoveBaseAction"
        client = actionlib.SimpleActionClient('move_base',MoveBaseAction)
        # Waits until the action server has started up and started listening for
goals.
        client.wait_for_server()
        # Creates a new goal with the MoveBaseGoal constructor
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = "map"
        goal.target_pose.header.stamp = rospy.Time.now()
        # Move 0.5 meters forward along the x axis of the "map" coordinate frame
        goal.target_pose.pose.position.x = posearray[0][0]
        goal.target_pose.pose.position.y = posearray[0][1]
        goal.target_pose.pose.position.z = posearray[0][2]
        # No rotation of the mobile base frame w.r.t. map frame
        goal.target_pose.pose.orientation.w = posearray[1][3]
        # Sends the goal to the action server.
        client.send_goal(goal)
        # Waits for the server to finish performing the action.
        wait = client.wait_for_result()
        # If the result doesn't arrive, assume the Server is not available
        if not wait:
            rospy.logerr("Action server not available!")
            rospy.signal_shutdown("Action server not available!")
        else:
        # Result of executing the action
            return client.get_result()
```

The execute function first determines whether the aruco id is read, and if it is, the specified point is reached according to the id, if not, it navigates to different points in order of finding.

```
    def execute(self, userdata):
        if userdata.Navi_goal_id>0:
            print("goal is",userdata.Navi_goal_id)
            tmp=0
            if userdata.Navi_goal_id==1:
                tmp=-2
                result=self.movebase_client(self.waypoints[tmp])
            if userdata.Navi_goal_id==2:
                tmp=1
                for i in range(tmp+1):
                    result=self.movebase_client(self.waypoints[i])

            if userdata.Navi_goal_id==3:
```

```
                tmp=3
                result=self.movebase_client(self.waypoints[0])
                result=self.movebase_client(self.waypoints[1])
                result=self.movebase_client([(2.3358,1.472,0),(0,0,0.996,0.08)])
                result=self.movebase_client(self.waypoints[3])
                result=self.movebase_client(self.waypoints[4])
            if userdata.Navi_goal_id==4:
                tmp=4
                result=self.movebase_client(self.waypoints[0])
                print(1)
                # result=self.movebase_client(self.waypoints[1])
                result=self.movebase_client([(2.3358,1.472,0),(0,0,0.986,0.08)])
                result=self.movebase_client(self.waypoints[3])
                result=self.movebase_client(self.waypoints[6])
                result=self.movebase_client(self.waypoints[7])
            if result:
                rospy.loginfo("Successfully navigate to aruco maker id")
                return 'outcome_end'

        if self.cnt < len(self.waypoints):
            info_msg = "Start navigating to " + str(self.cnt) + "th point"
            rospy.loginfo(info_msg)
            result = self.movebase_client(self.waypoints[self.cnt])
            if result:
                rospy.loginfo("Successfully navigate to:"+ str(self.cnt)+"th
goals")
            self.cnt += 1
            userdata.Navi_next_state = 0
            return 'outcome_navi'
        else:
            rospy.loginfo('Navigation completed')
            userdata.Navi_next_state = 1
            return 'outcome_aruco'
```

ArucoFinder

Arucofinder's init fuctiton instantiates the previously created class Pose_py, initializes the state of the state machine, and posts a message to '/cmd_vel' to enable parking.

```
class ArucoFinder(smach.State):
    def __init__(self):
        self.pp1 = Pose_py(1)
        self.pp2 = Pose_py(2)
        self.pp3 = Pose_py(3)
        self.pp4 = Pose_py(4)
        smach.State.__init__(self, outcomes=['outcome_beep'],
                             output_keys=['Arucofinder_id','Aruco_next_state'])
        # rospy.Subscriber("/aruco_single/pose",PoseStamped,pp.posecallback)
        self.pub = rospy.Publisher('/cmd_vel', Twist,queue_size=10)
        self.msg=Twist()
        # pp = Pose_py()
        self.rate = rospy.Rate(10)
        self.finishstop=False
        self.havefindid=False
```

This function allows the created node to subscribe to the previously created topic, and according to the value returned by the callback function to determine the id, in the specific operation, is to reach the specified position to start rotating to achieve, in some control methods to make turtlebot3 close to aurco and stop.

```
    def execute(self, userdata):
        print("start finding aruco code")
        rospy.Subscriber("/aruco1/pose",PoseStamped,self.pp1.posecallback)
        rospy.Subscriber("/aruco2/pose",PoseStamped,self.pp2.posecallback)
        rospy.Subscriber("/aruco3/pose",PoseStamped,self.pp3.posecallback)
        rospy.Subscriber("/aruco4/pose",PoseStamped,self.pp4.posecallback)
        userdata.Arucofinder_id = 0
        userdata.Aruco_next_state = 1
        selected=0
        print(selected)
        aruco0=self.pp1
        while (not rospy.is_shutdown() and not self.finishstop):
            if (not self.havefindid):
                if (self.pp1.aruco_z!=0 and abs(self.pp1.aruco_x)<0.5):
                    userdata.Arucofinder_id=1
                    selected=1
                    print("id is 1")

                elif (self.pp2.aruco_z!=0 and abs(self.pp2.aruco_x)<0.5):
                    userdata.Arucofinder_id=2
                    aruco0=self.pp2
                    selected=2
                    print("id is 2")

                elif(self.pp3.aruco_z!=0 and abs(self.pp3.aruco_x)<0.5):
                    userdata.Arucofinder_id=3
                    aruco0=self.pp3
                    selected=3
                    print("id is 3")


                elif(self.pp4.aruco_z!=0 and abs(self.pp4.aruco_x)<0.5):
                    userdata.Arucofinder_id=4
                    aruco0=self.pp4
                    selected=4
                    print("id is 4")

                    soundhandle = SoundClient()
                    rospy.sleep(1)
                    num = int(1)
                    volume = 1.0
                    rospy.loginfo('Playing sound %i.' % num)

                    for i in range(4):
                        soundhandle.play(num, volume)
                        rospy.sleep(1)

                    rospy.sleep(1)
```

```
                if(not selected==0):
                    self.havefindid=True
                else:
                    self.msg.angular.z = -0.35
                    self.msg.linear.x = 0
                    self.pub.publish(self.msg)
                    self.rate.sleep()
                continue

            elif aruco0.aruco_z>0.43 and abs(aruco0.aruco_x)<0.3:
                ratio = (aruco0.aruco_z-0.20)
                self.msg.angular.z=-0.8*(aruco0.aruco_x)
                self.msg.linear.x=max(0.15*(ratio),0.02)
                self.pub.publish(self.msg)
                self.rate.sleep()
            else:
                self.msg.angular.z=0.0
                self.msg.linear.x=0.0
                self.pub.publish(self.msg)
                self.rate.sleep()

                self.msg.angular.z=0
                self.msg.linear.x=-0.03

                for i in range(5):
                    self.pub.publish(self.msg)
                    self.rate.sleep()
                self.finishstop=True
                break
        return 'outcome_beep'
```

main

```
if __name__ == '__main__':
    rospy.init_node('lab7')
    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['end']) #
    sm.userdata.mission_log = 0  # 0:navi 1:aruco
    sm.userdata.aruco_id = 0  #
    # Open the container
    with sm:
        # Add states to the container
        smach.StateMachine.add('Navi',Navi(),
                               transitions={'outcome_navi':'BAR',
'outcome_aruco':'Arucofinder', 'outcome_end':'end'},
                               remapping=
{'Navi_goal_id':'aruco_id','Navi_next_state':'mission_log'})
        smach.StateMachine.add('Arucofinder', ArucoFinder(),
                               transitions={'outcome_beep': 'Navi'},
                               remapping=
{'Arucofinder_id':'aruco_id','Aruco_next_state':'mission_log'})
        smach.StateMachine.add('BAR', Bar(),
                               transitions={'outcome_navi':'Navi'},
```

```
                                remapping={'bar_next_state':'mission_log'})
    # Execute SMACH plan
    outcome = sm.execute()
```