**Software system** สามารถแบ่งออกเป็นได้หลาย module โดยmodules สามารถแบ่งกลุ่มได้ออกเป็น 3 layer คือ

* Decision layer (planning module) เป็นชั้น planning เป็นชั้นของstate โดยชั้นนี้จะไปเรียกรับค่าจาก perception layer และสั่งงานหุ่นยนต์ผ่าน control layer
* Control layer (control module)จะเป็นตัวสั่งการหุ่นยนต์
* Perception layer (perception module) จะประกอบไปด้วย environment understanding จะมีข้อมูลเกี่ยวกับหุ่นยนต์ เช่น speech recognition, localization จาก sensor ต่างๆ เช่น laser range finder(hokuyo), microphone, kinect, etc ซึ่งแต่ละ module จะสื่อสารกันผ่านทาง Robot Operating System(ROS)

**-Decision layer**

แบ่งออกเป็น

* task แบ่งงานออกเป็นกลุ่มใหญ่ๆ เช่น restaurant task, GPSR, etc ภายในtask ใหญ่จะมีบางfuction ที่มีการทำงานซ้ำๆกัน จึงมีการเขียนเป็น subtask ย่อยลงมา เพื่อง่ายต่อการเรียกใช้งาน

template

from include.abstract\_task import AbstractTask

class template(AbstractTask):

def \_\_init\_\_(self, planning\_module):

AbstractTask,\_\_init\_\_(self, planning\_module)

self.subtask = None

def perform(self,perception\_data):

if self.state is ‘init’:

# Do something

self.subtaskBook.get\_subtask(self,’Say’).say(‘I\’m ready to start’)

self.subtask = self.subtaskBook.get\_subtask(self, ‘MovePassDoor’)

self.change\_state(‘move\_pass\_door’)

elif self.state is ‘move\_pass\_door’:

if self.subtask.state is ‘finish’: #เป็นการเช็คว่า subtask ที่สั่งทำงานจบรึยังเพื่อกันให้ไม่ทำงานอื่นต่อทั้งที่งานก่อนหน้ายังทำงานจบ

self.change\_state(‘wait\_command’)

elif self.state is ‘wait\_command’:

if perception\_data.device is ‘VOICE’ and ‘start’ in perception\_data.input:

#ข้อมูลที่ได้จาก perception layer มีจำนวนมาก แต่เราต้องการเช็คข้อมูลเสียง และตรวจว่ามีคำว่า ‘start’ ในข้อมูลที่หุ่นได้ยินรึเปล่า

self.subtask = self.subtaskBook.get\_subtask(self, ‘Say’)

self.subtask.say(‘I\’m ready to start’)

#เป็นรูปแบบการเรียกใช้ subtask

self.change\_state(‘finish’)

#เมื่อจบ task ให้เปลี่ยน stateของtask เป็น finish

การทำงานของ task จะมีการไหลไปทีละ stateๆโดยไม่จำเป็นต้องไหลแบบบนลงล่าง ดังนั้นเมื่อสั่งให้subtask ใดๆทำงานให้เปลี่ยน state ของ task เพื่อให้flow ไหล เเละไม่เกิดลูปหรือการสั่งsubtask ทำงานรัวๆ

เมื่อสร้างไฟล์ taskจะต้อง add task เข้า task\_book และต้องสร้าง launch file เพื่อให้ ROS สร้างเรียกใช้งานได้

# taskไม่สามารถเรียกใช้งาน task อื่นได้ และไม่สามารถเรียกใช้งาน skill ได้

# การสั่งงาน subtask เดียวกันติดกันต้องใส่ delay เพื่อกันไม่ให้ค่าสถานะของsubtask เดิมถูกส่งกลายเป็นสถานะของ subtask ใหม่

* subtask การเขียนคล้ายการเขียน task

template

from include.abstract\_subtask import AbstractSubtask

class template(AbstractSubtask):

def \_\_init\_\_(self, planning\_module):

AbstractSubtask.\_\_init\_\_(Self, planning\_module)

self.skill = self.current\_skill

self.subtask = self.current\_subtask

def perform(self, perception\_data):

if self.state is ‘init’:

#check if skill is succeed

if self.skill.state is ‘succeed’:

self.change\_state(‘finish’)

elif self.skill.state is ‘aborted’:

self.change\_state(‘error’)

เมื่อสร้าง subtask file ใหม่ต้องใส่subtask เข้า subtaskbook ด้วยเพื่อให้ subtask อื่นหรือ task สามารถเรียกไปใช้งานได้

# subtask สามารถเรียกใช้งาน subtask เเละ skill อื่นได้

* skill ภายใน subtask บางครั้งจำเป็นต้องทำงานย่อยๆซ้ำๆ จึงเขียน skill ขึ้นมาเพื่อให้ง่ายต่อการเรียกใช้ และเป็นตัวสั่งงาน control module ให้ไปสั่งหุ่นอีกที

template

from include.abstract\_skill import AbstractSkill

class Template(AbstractSkill)

def \_\_init\_\_(self, control\_module):

AbstractSkill,\_\_init\_\_(self, control\_module)

gripper = control\_module.right\_gripper

def perform(self, perception\_data):

if self.state is ‘init’:

self.change\_State(‘succeeded’)

เมื่อสร้าง skill ใหม่ต้องเพิ่มข้อมูลเข้าใน skill book ด้วยเพื่อให้ subtask เรียกใช้ได้