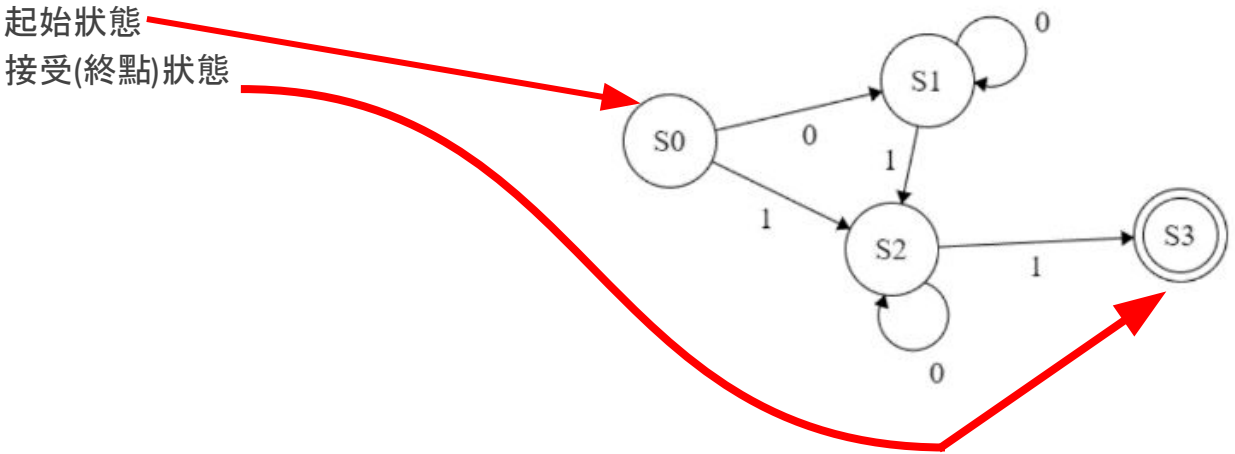
無人機智慧系統開發與實作
System Development and Implementation of Drone Intelligence

Finite State Machine (FSM) 介紹

有限狀態機 Finite State Machine

又稱有限狀態自動機。

表示有限個狀態以及在這些狀態之間的轉移和動作等行為的數學模型。



Example

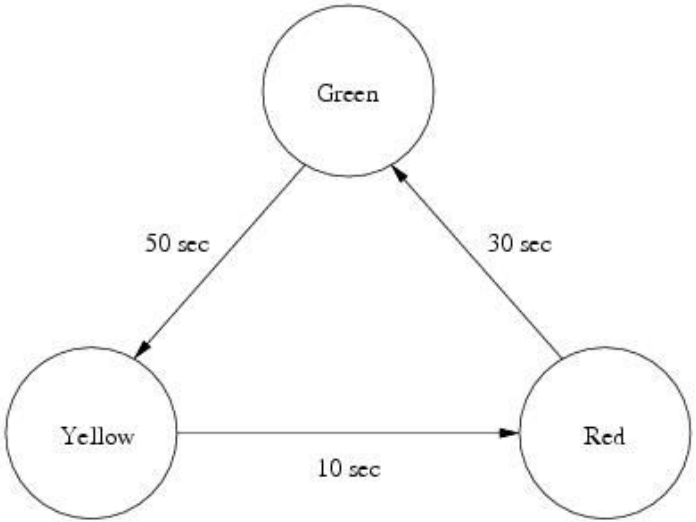
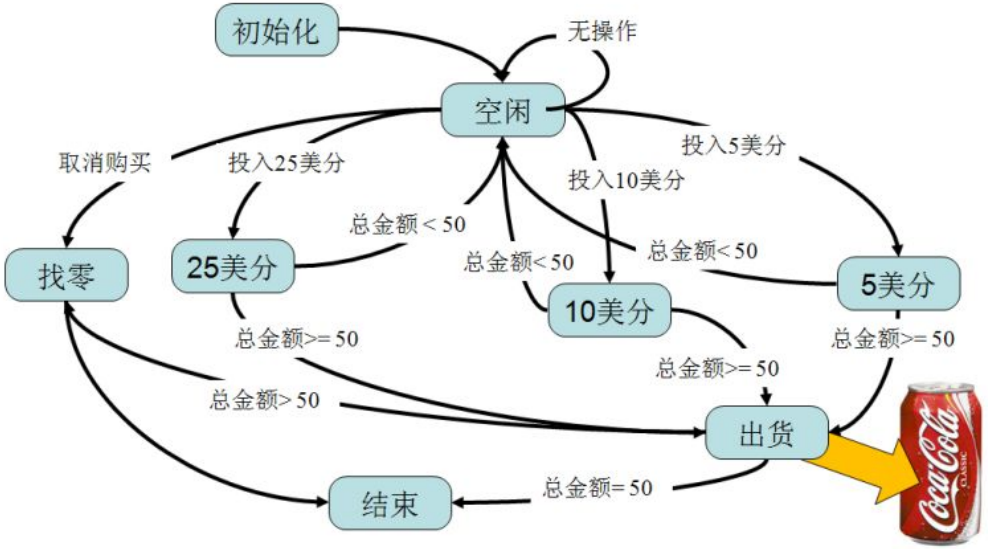


Figure 1: traffic light

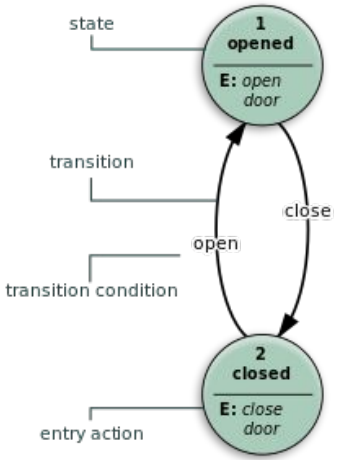


可乐自动贩售机模型

FSM 分類

接受/辨識器: 產生二元輸出(Yes or No)來表示是否被機器接受。
當所有輸入被處理後，如果當前狀態是接受狀態，則輸入被接受，否則被摒。
Input: 字元，(不使用動作)
Ex. 正規語言

變換器: 基於輸入和/或狀態產生輸出。應用於控制。
Ex. 電梯門控制

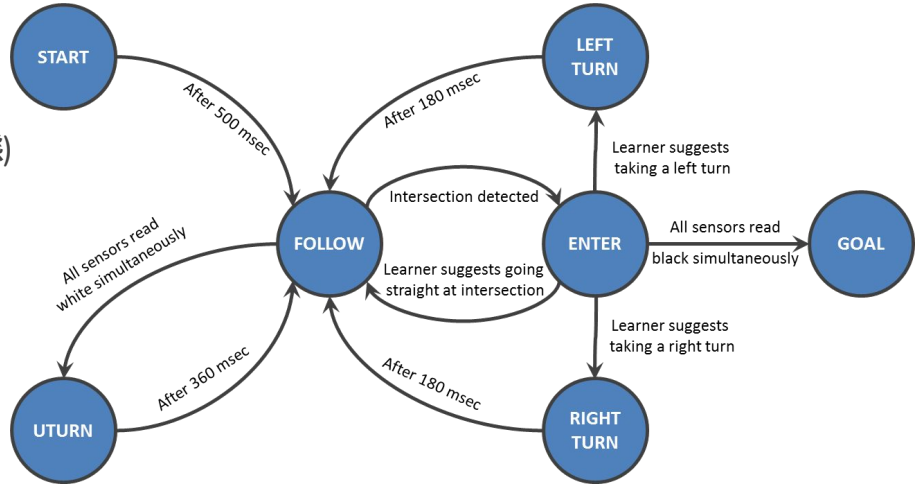


FSM-based programming

兩個特徵:

1. 程式執行時可以清楚劃分成數個有限狀態機的步驟
2. 不同的步驟程式區段只能透過一組清楚標示的變數(亦即狀態)交換資訊。

=> 一程式在任兩個不同時間下，只有狀態數值不同，其餘都一樣。



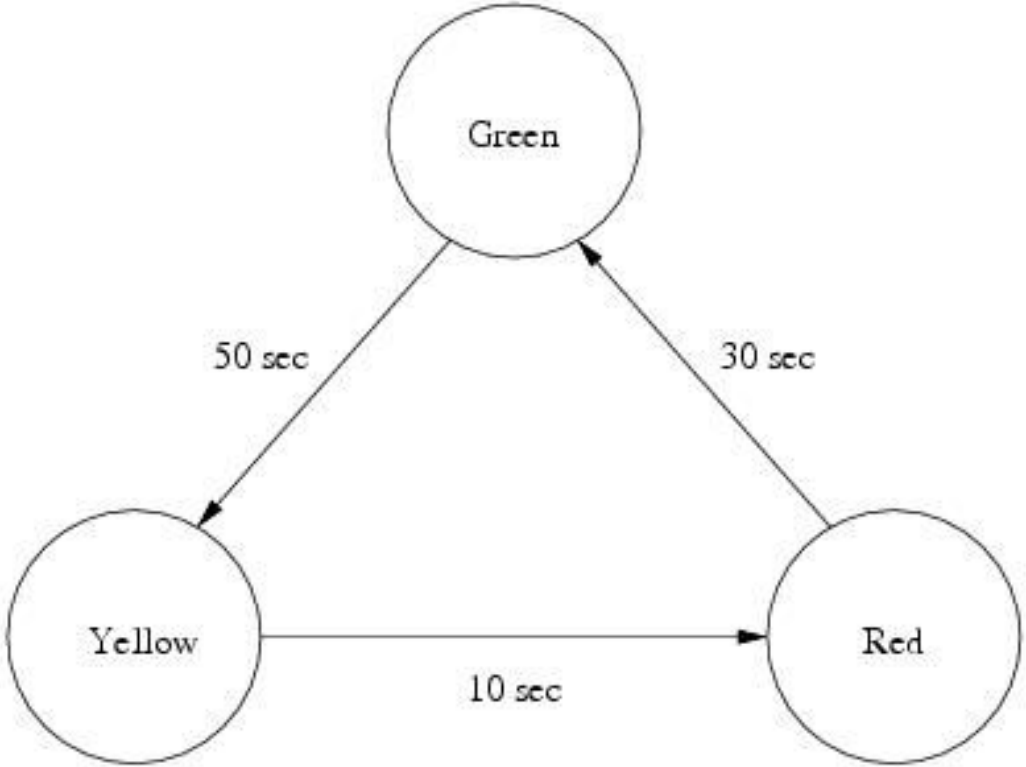
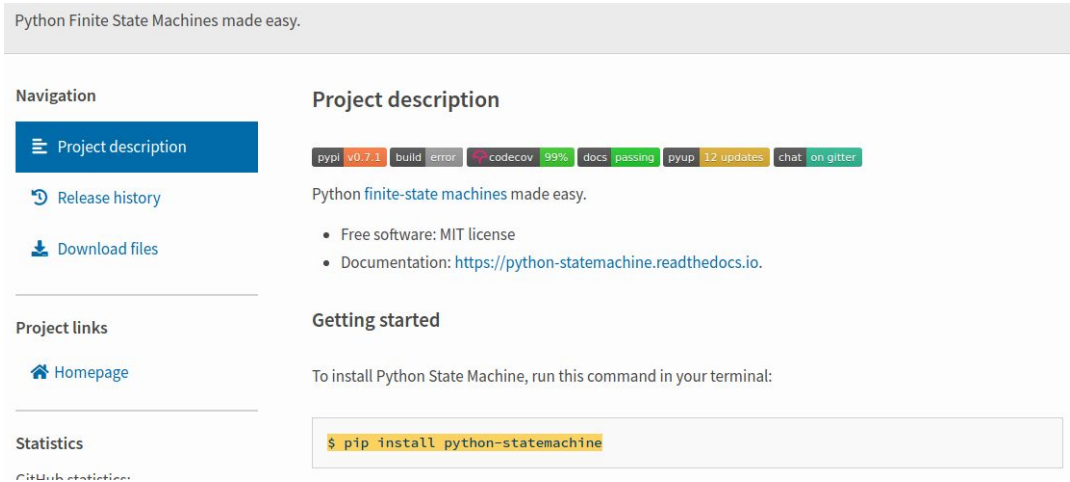


Figure 1: traffic light

https://pypi.org/project/python-statemachine/

Python statemachine

```
$ pip install python-statemachine
```



https://pypi.org/project/python-statemachine/

FSM - code

```
from statemachine import StateMachine, State

class TrafficLightMachine(StateMachine):
    green = State('Green', initial = True)
    yellow = State('Yellow')
    red = State('Red')

    slowdown = green.to(yellow)
    stop = yellow.to(red)
    go = red.to(green)

# first
traffic_light = TrafficLightMachine()
print("now state: ", traffic_light.current_state)

print("is green: ", traffic_light.is_green)
print("is yellow: ", traffic_light.is_yellow)
print("is red: ", traffic_light.is_red)

print( [s.identifier for s in traffic_light.states] )
print( [t.identifier for t in traffic_light.transitions] )
```

```
E:\code_tello>python fsm_test.py
('now state: ', State('Green', identifier='green', value='green', initial=True))
('is green: ', True)
('is yellow: ', False)
('is red: ', False)
['green', 'red', 'yellow']
['go', 'slowdown', 'stop']
```

FSM - code

在綠燈下如果做了錯誤的transitions?

`traffic_light.stop()`

```
Traceback (most recent call last):
  File "fsm_test.py", line 23, in <module>
    traffic_light.stop()
  File "c:\python27\lib\site-packages\statemachine\statemachine.py", line 56, in __call__
    return self.func(*args, **kwargs)
  File "c:\python27\lib\site-packages\statemachine\statemachine.py", line 84, in transition_callback
    return self._run(machine, *args, **kwargs)
  File "c:\python27\lib\site-packages\statemachine\statemachine.py", line 108, in _run
    self._verify_can_run(machine)
  File "c:\python27\lib\site-packages\statemachine\statemachine.py", line 104, in _verify_can_run
    raise TransitionNotAllowed(self, machine.current_state)
statemachine.exceptions.TransitionNotAllowed: Can't stop when in Green.
```

FSM - code

在綠燈下如果做slowdown?

```
('now state: ', State('Yellow', identifier='yellow', value='yellow', initial=False))
('is green: ', False)
('is yellow: ', True)
('is red: ', False)
```

FSM - code

Callback

```
class TrafficLightMachine(StateMachine):
    green = State('Green', initial = True)
    yellow = State('Yellow')
    red = State('Red')

    slowdown = green.to(yellow)
    stop = yellow.to(red)
    go = red.to(green)

    def on_slowdown(self):
        print('YELLOW!!')

    def on_stop(self):
        print('RED!')

    def on_go(self):
        print('GREEN!')
```

```
YELLOW!!
('now state: ', State('Yellow', identifier='yellow', value='yellow', initial=False))
('is green: ', False)
('is yellow: ', True)
('is red: ', False)
RED!
('now state: ', State('Red', identifier='red', value='red', initial=False))
('is green: ', False)
('is yellow: ', False)
('is red: ', True)
GREEN!
('now state: ', State('Green', identifier='green', value='green', initial=True))
('is green: ', True)
('is yellow: ', False)
('is red: ', False)
```

FSM - code

```python
from statemachine import StateMachine, State
import time

class TrafficLightMachine(StateMachine):
    green = State('Green', initial = True)
    yellow = State('Yellow')
    red = State('Red')

    cycle = green.to(yellow) | yellow.to(red) | red.to(green)

    def on_enter_green(self):
        #self.stop()
        time.sleep(50)

    def on_enter_yellow(self):
        #self.go()
        time.sleep(10)

    def on_enter_red(self):
        #self.slowdown()
        time.sleep(30)

    def show(self):
        print("now state: ", self.current_state)
        print("is green: ", self.is_green)
        print("is yellow: ", self.is_yellow)
        print("is red: ", self.is_red)
```

```python
traffic_light = TrafficLightMachine()

print( [s.identifier for s in traffic_light.states] )
print( [t.identifier for t in traffic_light.transitions] )

while True:
    traffic_light.cycle()
    traffic_light.show()
```

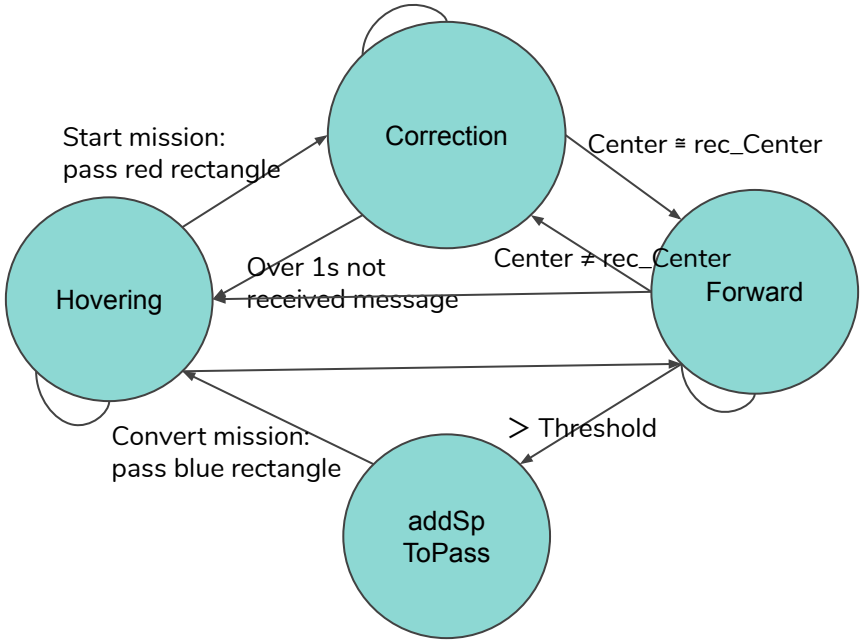
```
['green', 'red', 'yellow']
['cycle']
('now state: ', State('Yellow', identifier='yellow', value='yellow', initial=False))
('is green: ', False)
('is yellow: ', True)
('is red: ', False)
('now state: ', State('Red', identifier='red', value='red', initial=False))
('is green: ', False)
('is yellow: ', False)
('is red: ', True)
('now state: ', State('Green', identifier='green', value='green', initial=True))
('is green: ', True)
('is yellow: ', False)
('is red: ', False)
('now state: ', State('Yellow', identifier='yellow', value='yellow', initial=False))
('is green: ', False)
('is yellow: ', True)
('is red: ', False)
('now state: ', State('Red', identifier='red', value='red', initial=False))
('is green: ', False)
('is yellow: ', False)
('is red: ', True)
('now state: ', State('Green', identifier='green', value='green', initial=True))
('is green: ', True)
('is yellow: ', False)
('is red: ', False)
```

UAV Pass rectangle scheme

Pachauri, A., More, V., Gaidhani, P., & Gupta, N. (2016). Autonomous Ingress of a UAV through a window using Monocular Vision. *arXiv preprint arXiv:1607.07006*.

1. UAV最初只執行偏航運動，以便搜索目標窗口。一旦正確檢測到目標窗口，它將保持該高度並停止偏航運動。
2. 計算其當前位置與所需位置之間的相對角度（其正向將垂直於窗口平面，並且將與窗口中心對齊）。
3. 開始在特定方向上移動。執行Y平面中的運動（左/右運動）以及偏航運動以便繼續檢測整個窗口正常。
4. 根據窗口中心的高度連續調整其高度。
5. 一旦到達所需位置，開始向窗口移動（在X平面中移動）並將迭代地執行前面的步驟。
6. 每當它無法檢測到窗口時，默認情況下，它會開始遠離窗口，以便將其置於幀中並再次開始檢測

FSM in ROS


Correction
Start mission:
pass red rectangle
Center ≅ rec_Center
Hovering
Over 1s not
received message
Center ≠ rec_Center
Forward
Convert mission:
pass blue rectangle
> Threshold
addSp
ToPass

FSM in ROS

```
class sMachine(StateMachine):
    hover = State('Hover', initial = True)
    correction = State('Correction')
    forward = State('Forward')
    addSp = State('AddSp')

    to_hover = hover.to(hover) | addSp.to(hover) | correction.to(hover) | forward.to(hover)
    to_correction = hover.to(correction) | correction.to(correction) | forward.to(correction)
    to_forward = hover.to(forward) | correction.to(forward) | forward.to(forward)
    to_addSp = forward.to(addSp) | correction.to(addSp)
```

FSM in ROS

```
class MyModel(object):
    def __init__(self, state):
        self.state = state
        self.target = (-1,-1,1)
        self.center = (480, 320)
        self.check = False
        self.canLand = False
        self.rec_time = 0
        self.self_pub = rospy.Subscriber('/selfDefined', test, self.cback)
        self.cmd_pub = rospy.Publisher('/tello/cmd_vel', Twist, queue_size = 10)
        self.land_sub = rospy.Subscriber('/tello/status', TelloStatus, self.ts_callback)
        self.land_pub = rospy.Publisher('/tello/land', Empty, queue_size = 1)
        self.rate = rospy.Rate(10)


    def cback(self, data):
        self.rec_time = rospy.get_time()
        self.target = data.l1

    def ts_callback(self, data):
        if data.fly_mode == 12:
            self.canLand = True

    def run(self, fsm):
        #print(type(self))
        while not rospy.is_shutdown():
            print(self.state)
            if fsm.is_hover:
                self.cmd_pub.publish(Twist())
                self.rate.sleep()
                if (self.target[0] == -1 and self.target[1] == -1) or (rospy.get_time() - self.rec_time > 0.6):
                    fsm.to_hover()
                elif abs(self.target[0] - self.center[0]) >= 60 or abs(self.target[1] - self.center[1]) >= 60:
                    fsm.to_correction()
                elif abs(self.target[0] - self.center[0]) < 60 and abs(self.target[1] - self.center[1]) < 60:
                    fsm.to_forward()
```

```
if __name__ == '__main__':
    rospy.init_node('h264_pub', anonymous=True)
    obj = MyModel(state='hover')
    fsm = sMachine(obj)
    obj.run(fsm)
```