自主行動機器人導論 技術展示

姓名:葉冠宏 學號:108753208

成果影片:

https://youtu.be/iiAZXpUZFwM

簡介:

我總共用到三個程式的檔案，分別是 Test_h264_sub.py , Simple_tello.py, Fsm_pass_example.py.

Simple_tello.py 主要負責的是無人機動作模組的定義及發送，還有接收模組 State 的定義有哪些…等等。

Test_h264_sub.py 這個檔案主要負責的是無人機的影像判斷，以及在接收影像之後判斷要針對無人機所定義的 state 要發送怎樣的訊息。

Fsm_pass_example.py 中我使用的是 finite state machine 的框架。裡面有定義不同的 state 的動作詳細是該如何運作，以及定義在執行任務的時候，無人機的 state 要如何轉換至不同 state。

詳細說明:

1. Simple_tello.py:

```
class TelloState:

    def __init__(self):

        self.height = 0.0
        self.temperature_height_m = 0.0
        self.battery = 100.0
        self.is_flying = False
        self.fly_mode = 999
        self.target_x = -1
        self.target_y = -1
        self.canPass = -1
        ########################################
        self.hasPassedTwice=0
        self.cammiddle=1

        self.mission=0  #0: red , 1:blue , 2:findtarget
        ########################################3
```

無人機的 State 總共有這些，除了最基本的高度、溫度、電池、是否在飛、飛行模式之外。target_x, target_y 主要在儲存你的目標物的中心點座標位置，如果掃不到中心點，會回傳-1 的值。

canpass 在定義你是否可以通過你現在的目標物，那個顏色的框。初始值設為-1, 之後如果不行過框，其值為 0， 如果可以過框，其值為 1。

Haspassedtwice 在定義你已經過框了幾次。

Cammiddle 定義你的鏡頭是否有掃到目標物的顏色。其值為 1 代表有掃到，0 代表沒有。

Mission 在定義你現在所執行的任務是什麼。0 代表要過紅框，1 代表要過藍框，2 代表正在沿著綠色邊線試圖想掃到 apriltag，3 代表已經掃到 apriltag 後要沿著綠色邊線回來。

```
class Tello_drone():
    def __init__(self):
        self.state = TelloState()
        self.controler = TelloController()
        self._sensor()
    def _sensor(self):
        _ts_sub = rospy.Subscriber("/tello/status", TelloStatus, self._ts_cb, queue_size = 10)
        _tp_sub = rospy.Subscriber("/target_point", Float64MultiArray, self._tp_cb, queue_size = 10

    def _ts_cb(self, data):
        self.state.height = data.height_m
        self.state.temperature_height_m = data.temperature_height_m
        self.state.battery = data.battery_percentage
        self.state.is_flying = data.is_flying
        self.state.fly_mode = data.fly_mode
```

```
def _tp_cb(self, data):
    self.state.target_x = data.data[0]
    self.state.target_y = data.data[1]
    self.state.canPass = data.data[2]
    ########################################
    self.state.hasPassedTwice=data.data[3]
    self.state.cammiddle=data.data[4]
    self.state.mission=data.data[5]
```

Tello drone 這邊就在定義你如何接收所傳回的 data，並更改 Tellostate 所定義的 state。

```
class TelloController:

    def move(self, twist, limitTime):
        limitTime = limitTime * 1000
        startTime = int(round(time.time()*1000))
        rate = rospy.Rate(10)
        # print "MOVE~~~~"
        pub_move = rospy.Publisher("/tello/cmd_vel", Twist, queue_size = 10)

        while not rospy.is_shutdown():
          connections = pub_move.get_num_connections()
          if connections > 0:
            endTime = int(round(time.time()*1000))
            if endTime - startTime < limitTime:
              pub_move.publish(twist)

            else:
              #pub_move.publish(Twist())
              break
            rate.sleep()
```

```
def emergency(self):
    rate = rospy.Rate(10)

    puber = rospy.Publisher("/tello/emergency", Empty, queue_size=10)

    while not rospy.is_shutdown():
      cons = puber.get_num_connections()

      if cons > 0:
        puber.publish(Empty())
        rate.sleep()
        break
```

```
def takeoff(self):
    rate = rospy.Rate(10)

    puber = rospy.Publisher("/tello/takeoff", Empty, queue_size=10)

    while not rospy.is_shutdown():
      cons = puber.get_num_connections()

      if cons > 0:
        puber.publish(Empty())
        rate.sleep()
        break
```

```
def land(self):
    rate = rospy.Rate(10)

    puber = rospy.Publisher("/tello/land", Empty, queue_size=10)

    while not rospy.is_shutdown():
      cons = puber.get_num_connections()

      if cons > 0:
        puber.publish(Empty())
        rate.sleep()
        break
```

Tellocontroller 這個模組主要在定義無人機對於動作要如何對硬體發送訊息做出對應的反應。有移動(包含旋轉)、緊急降落、起飛、降落等 function。

2. Test_h264_sub.py

```
if __name__ == '__main__':
    try:
        main()
    except BaseException:
        traceback.print_exc()
    finally:
        stream.close()
        cv2.destroyAllWindows()
```

```
def main():

    fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')
    out = cv2.VideoWriter('test.avi', fourcc, 20.0, (1920, 720))

    rospy.init_node('h264_listener')
    rospy.Subscriber("/tello/image_raw/h264", CompressedImage, callback)
    point_pub = rospy.Publisher("/target_point", Float64MultiArray, queue_size = 10)
    container = av.open(stream)
    rospy.loginfo('main: opened')
    frame_skip = 300
    ################################3
    start_detect = True
    hasPassedTwice=0
    cammiddle=1
    mission=0

    width=960
    height=720
    detector = apriltag.Detector()

    mygoal=3
```

Main function 的前半部主要是對程式啟動後的各個參數做初始化。包括視窗 video 、無人機的接收端、發送端節點、frame 的頻率 frame_skip、鏡頭的寬度 960、高度 720，apriltag 的 api 等。

Start_detect 主要在定義是否還要再去針對目標物做是否要通過的判斷。

Haspassedtwice 是定義通過了框幾次。

Camiddle 是定義是否有掃到目標物的顏色。

Mission 是定義現在無人機所執行的任務。0 代表要過紅框，1 代表要過藍框，2 代表正在沿著綠色邊線試圖想掃到 apriltag，3 代表已經掃到 apriltag 後要沿著綠色邊線回來。

Mygoal 是在掃到 apriltag 之後先暫存你的 mission 是什麼，等待沿著綠色邊線回來之後，再去 assign 給 mission，決定要過紅框還是籃框。

```
for frame in container.decode(video=0):
    if 0 < frame_skip:
        frame_skip -= 1
        continue
    start_time = time.time()
    image = cv2.cvtColor(np.array(
        frame.to_image()), cv2.COLOR_RGB2BGR)

    hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    if mission==0:
     mask = find_Mask(hsv_img)
    elif mission==1:
     mask = find_Mask2(hsv_img)
    else:
     mask = find_Mask3(hsv_img)

    c_c, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    show_image = cv2.cvtColor(np.zeros(image.shape[:2], dtype=np.uint8), cv2.COLOR_GRAY2BGR)
```

這邊我們把無人機所掃到的影像每個 frame 先轉成 BGR channel 的 image，然後再轉成 HSV 的 channel，看現在 mission 無人機的任務是什麼來決定現在無人機是要掃哪個顏色的色塊。
Find_Mask 是掃紅色的色塊，find_mask2 是掃藍色的色塊，find_mask3 是掃綠色的色塊。

經過 find_mask function 返回後的 mask, 我們再經過 cv2.findcontours 找出色塊的邊界。

```
if mission==2:
 ##############################################################################3
 if len(c_c)==0:
  cammiddle=0
  rec_x=-1
  rec_y=-1
  point_pub.publish(Float64MultiArray(data = [rec_x, rec_y, 0,hasPassedTwice,cammiddle,mission]))
 else:
  cammiddle=1
  max_c = max(c_c, key = cv2.contourArea)
  rect = cv2.minAreaRect(max_c)
  r_w, r_h = rect[1]
  rec_x = rect[0][0]     #the rectangle center x
  rec_y = rect[0][1]     #the rectangle center y
  cv2.rectangle(show_image, (int(rec_x-0.5*r_w), int(rec_y-0.5*r_h)), (int(rec_x+0.5*r_w), int(re
  cv2.circle(show_image, (int(rec_x), int(rec_y)), 10, (0,50,175), -1)
  cv2.putText(show_image, str((r_w * r_h) / (960*720.)), (10,40), 5, 2, (255,255,0))

  search = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  results = detector.detect(search)
```

```
if len(results)==0:
  mission=2
else:
  goal = results[0].tag_id
  print(goal)

  if int(goal)==10:
    mission=3
    mygoal=1
  elif int(goal)==11:
    mission=3
    mygoal=0
  else:
    mission=2

#if r_w * r_h < width*height*0.001:
# approach=1

point_pub.publish(Float64MultiArray(data = [rec_x, rec_y, 0,hasPassedTwice,cammiddle,mission]))
```

Mission 2 是執行沿著綠色邊線至 Apriltag 的任務。

在這個任務中，如果 len(c_c)=0，即你掃 findcontour 中沒有掃到綠色的色塊，他會回傳 cammiddle=0, 中心點座標為-1 的值，以做出對應的反應。

如果有掃到綠色色塊，cammiddle 會設定為 1。我們去用 max 和 cv2.minAreaRect 找色塊邊界 c_c 的矩形範圍，然後找出其長 r_w, 高 r_h，中心座標 rec_x, rec_y。我們把 video 視窗中的影像加上所辨識的矩形，其中心點，還有矩形佔整個視窗的比例數字。

我們把上述資訊 publish 給無人機的節點，供動作端去判斷做上對應的反應。

針對 apriltag 方面，我們先把無人機影像轉成灰階，供 apriltag 的 api 去做辨識。如果沒有辨識到 apriltag 或是辨識的不是老師所列的 apriltag 的可能編號，則就會繼續 publish mission=2。否則如果有辨識到，我們就會把所辨識的編號所對應的任務先暫存到 mygoal，然後 mission 就會改為 3，進入沿著綠色邊線回去的階段。

```
elif mission==3:
  if len(c_c)==0:
    cammiddle=0
    rec_x=-1
    rec_y=-1
    point_pub.publish(Float64MultiArray(data = [rec_x, rec_y, 0,hasPassedTwice,cammiddle,mission]))

    mission=mygoal


  else:
    cammiddle=1
    max_c = max(c_c, key = cv2.contourArea)
    rect = cv2.minAreaRect(max_c)
    r_w, r_h = rect[1]
    rec_x = rect[0][0]     #the rectangle center x
    rec_y = rect[0][1]     #the rectangle center y
    cv2.rectangle(show_image, (int(rec_x-0.5*r_w), int(rec_y-0.5*r_h)), (int(rec_x+0.5*r_w), int(re
    cv2.circle(show_image, (int(rec_x), int(rec_y)), 10, (0,50,175), -1)
    cv2.putText(show_image, str((r_w * r_h) / (960*720.)), (10,40), 5, 2, (255,255,0))
```

如果 mission=3，代表無人機正在沿著綠色邊線回去。如果有掃到綠色色塊，同樣的，我們就會把矩形的長、寬、鏡頭是否在中央…等資訊回傳。

如果沒有掃到綠色色塊了，代表回程已經到盡頭了，於是我們把 mygoal 所暫存的任務代號 assign 給 mission 來決定我們要通過紅框還是籃框。

```
else:
 ##########################################################################################3
 if len(c_c)==0:
  cammiddle=0
  rec_x=-1
  rec_y=-1

  if hasPassedTwice==2:
   #point_pub.publish(Float64MultiArray(data = [rec_x, rec_y, 1,hasPassedTwice,cammiddle]))
   stream.close()
   cv2.destroyAllWindows()
   break
  else:
   start_detect=True
   if hasPassedTwice==1:
    if mygoal==3:
     mission=2

   point_pub.publish(Float64MultiArray(data = [rec_x, rec_y, 0,hasPassedTwice,cammiddle,mission])
```

```
else:
 cammiddle=1
 max_c = max(c_c, key = cv2.contourArea)
 rect = cv2.minAreaRect(max_c)
 r_w, r_h = rect[1]
 rec_x = rect[0][0]    #the rectangle center x
 rec_y = rect[0][1]    #the rectangle center y


 if mission==0:
  cv2.rectangle(show_image, (int(rec_x-0.5*r_w), int(rec_y-0.5*r_h)), (int(rec_x+0.5*r_w), int(re
 else:
  cv2.rectangle(show_image, (int(rec_x-0.5*r_w), int(rec_y-0.5*r_h)), (int(rec_x+0.5*r_w), int(re
 print("minAreaRect x: ", rec_x)
 print("minAreaRect y: ", rec_y)
 cv2.circle(show_image, (int(rec_x), int(rec_y)), 10, (0,50,175), -1)
 cv2.putText(show_image, str((r_w * r_h) / (960*720.)), (10,40), 5, 2, (255,255,0))
```

```
if start_detect:
 if r_w * r_h >= width*height*0.35:
  hasPassedTwice+=1
  point_pub.publish(Float64MultiArray(data = [rec_x, rec_y, 1,hasPassedTwice,cammiddle,mission]
  start_detect=False

 else:
  if r_w * r_h < width*height*0.01:
   cammiddle=0
   rec_x=-1
   rec_y=-1

  point_pub.publish(Float64MultiArray(data = [rec_x, rec_y, 0,hasPassedTwice,cammiddle,mission]
```

當進入到 mission 0 或 1 時，代表無人機要執行的是過紅框或是藍框的任務。

如果沒有掃到目標物的色塊，我們會進入第一個 if 迴圈。如果已經過框兩次，則程式會跳出 for loop ，關掉 video 視窗，整個任務達成。但如果是剛通過紅框，此時 haspassedtwice 通過框的次數為 1。於是 start_detect 辨識是否要判別可以過框的變數更改回初始的 True，然後 mission 改為 2，即追蹤綠色邊線至 apriltag。然後我們也是 publish 相關的資訊。

如果有掃到目標物的顏色，則我們也是把 cammiddle 鏡頭是否有掃到顏色、色塊的中心座標，以及已經通過幾次框、是否可以過框、什麼 mission 等資訊 publish 出去，做對應的反應。這邊，如果你掃到的色塊比例小於整個視窗的 0.01, 還是會判定為沒有掃到色塊，cammiddle=0，色塊中心座標改為-1。

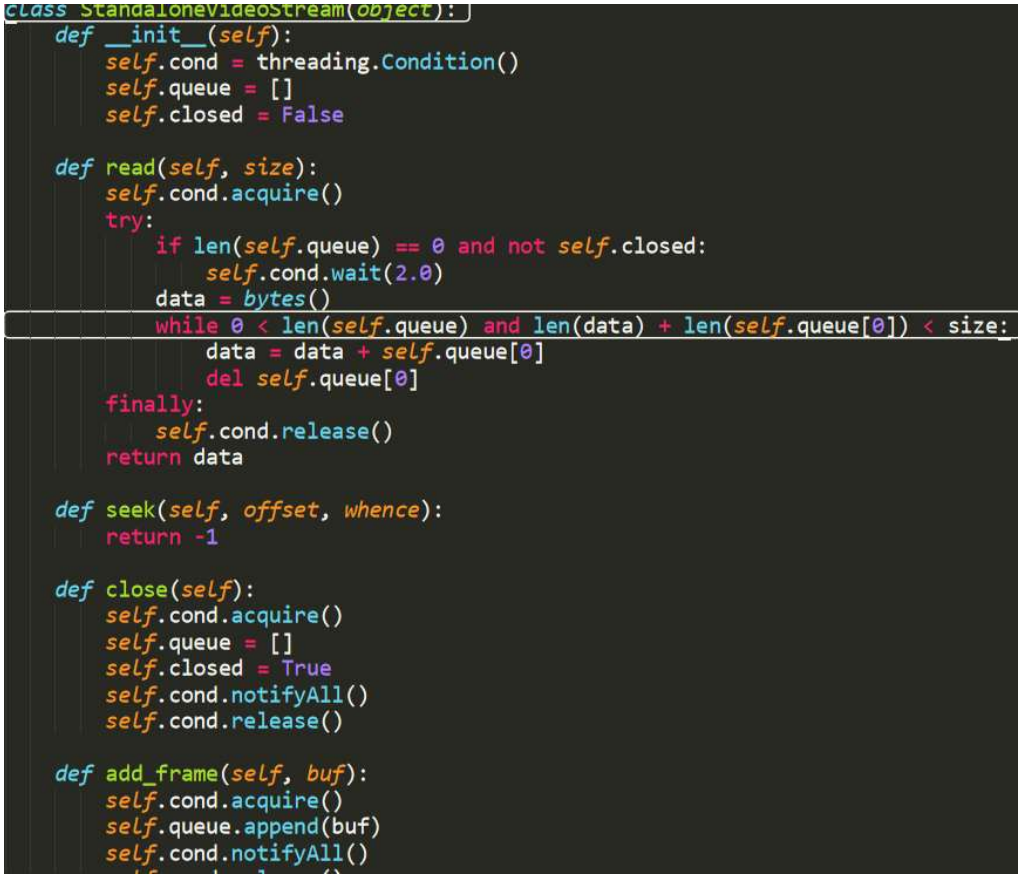
如果掃到的色塊不大於視窗比例的 0.35，則判定為不行過框，startdetect 維持 True，無人機的 canpass 的 state 還是 0。但是一旦大於 0.35，則 canpass 的無人機 state 改為 1，已過框次數加 1，startdetect 改為 false，直到已經過框。Start_detect=false 後，就不會再對無人機相關的 state 做竄改。

```
def find_Mask(img):    #red
    lr0 = np.array([0, 70, 0])
    ur0 = np.array([5, 255, 255])
    lr1 = np.array([175, 70, 0])
    ur1 = np.array([180, 255, 255])
    rm0 = cv2.inRange(img, lr0, ur0)
    rm1 = cv2.inRange(img, lr1, ur1)
    rm = cv2.bitwise_or(rm0, rm1)
    return rm


def find_Mask2(img): #blue
    lb0 = np.array([100, 70, 0])
    ub0 = np.array([130, 255, 255])
    bm = cv2.inRange(img, lb0, ub0)
    return bm

def find_Mask3(img): #green
    lg0 = np.array([45, 70, 0])
    ug0 = np.array([75, 255, 255])
    gm = cv2.inRange(img, lg0, ug0)
    return gm
```

Find_mask 是在抓紅色的色塊，因為在 hsv 中紅色的區塊位於 0~5 和 175~180 的頻率中，所以兩個區段都算是紅色。

Findmask2 則是辨別藍色，我們取 100~130 的區段。

Findmask3 是辨別綠色，我們取 45~75 的區段。



```
class StandaloneVideoStream(object):
    def __init__(self):
        self.cond = threading.Condition()
        self.queue = []
        self.closed = False

    def read(self, size):
        self.cond.acquire()
        try:
            if len(self.queue) == 0 and not self.closed:
                self.cond.wait(2.0)
            data = bytes()
            while 0 < len(self.queue) and len(data) + len(self.queue[0]) < size:
                data = data + self.queue[0]
                del self.queue[0]
        finally:
            self.cond.release()
        return data

    def seek(self, offset, whence):
        return -1

    def close(self):
        self.cond.acquire()
        self.queue = []
        self.closed = True
        self.cond.notifyAll()
        self.cond.release()

    def add_frame(self, buf):
        self.cond.acquire()
        self.queue.append(buf)
        self.cond.notifyAll()
```

這邊的 class 則是主要對視窗 video 作一些基本的初始化、object 的建立等影像所需的設定。

3. Fsm_pass_example.py.

```
class sMachine(StateMachine):

    ########################################
    # state
    hover = State('Hover', initial = True)
    correction = State('Correction')
    forward = State('Forward')

    ##passing rapidly
    addSp = State('AddSp')
    ########################################
    rotate=State('Rotate')
    ########################################
    trace=State('Trace')
    traceback=State('Traceback')
```

```
stop2do = addSp.to(hover) | traceback.to(hover)
refly = rotate.to(hover)
need2rotate = rotate.to(rotate) | hover.to(rotate) | correction.to(rotate) | forward.to(rotate)
################################################
wait4data = hover.to(hover)
start2correct = hover.to(correction)
need2correct = correction.to(correction) | forward.to(correction)
start2forawrd = hover.to(forward)
need2forawrd = correction.to(forward) | forward.to(forward)
need2addSp = forward.to(addSp) | correction.to(addSp)
#########################################################################
need2trace = hover.to(trace) | trace.to(trace)
need2traceback= trace.to(traceback) | traceback.to(traceback)
```

```
def on_enter_hover(self):
    msg = Twist()
    t1.controler.move(msg, 0.5)

def on_enter_correction(self):
    msg = Twist()
    dx = t1.state.target_x - 480
    dy = t1.state.target_y - fakey
    if dx != 0:
        msg.linear.x = dx / abs(dx) * 0.1
    if dy != 0:
        msg.linear.z = -dy / abs(dy) * 0.2
    t1.controler.move(msg, 0.5)

def on_enter_forward(self):
    msg = Twist()
    msg.linear.y = 0.2
    t1.controler.move(msg, 0.5)
```

```
def on_enter_addSp(self):
    msg = Twist()
    msg.linear.y = 0.4
    t1.controler.move(msg, 4)

    msg = Twist()
    msg.linear.y = 0.5
    t1.controler.move(msg, 4.3)


def on_enter_rotate(self):
    msg = Twist()
    msg.angular.z = 0.4
    t1.controler.move(msg, 0.6)
```

```
def on_enter_trace(self):
    msg = Twist()
    #traceH=rdH*2
    traceH=height/2
    dy = t1.state.target_y -(traceH+20)
    msg.linear.x = -0.1
    if dy != 0:
        msg.linear.z = -dy / abs(dy) * 0.1
    t1.controler.move(msg, 0.5)
########################################
def on_enter_traceback(self):
    msg = Twist()
    #traceH=rdH
    traceH=height/2
    dy = t1.state.target_y -traceH
    msg.linear.x = 0.1
    if dy != 0:
        msg.linear.z = -dy / abs(dy) * 0.1
    t1.controler.move(msg, 0.5)
```

我使用的是 finite state machine 的模型架構。我總共定義了 7 個 state。Hover, correction, forward, addsp, rotate, trace, traceback。也定義了這些 state 之間應該如何轉換的。

Hover 主要負責的是滯留在空中，通常是因為剛開始一個任務前所做的待機的角色。

Correction 主要負責無人機對於中心點的校正。我們利用鏡頭的中心座標，和 test_h264_sub.py 所回傳的色塊中心座標，其水平方向和垂直方向都做相減來做校正。對於水平方向，我們直接把相減值做標準化，然後乘以 0.1 的速度做移動，移動 0.5 秒。垂直方向則是，因為鏡頭的 y 座標是越往下越大，所以你要多乘以一個負號做校正，然後無人機的移動則是對 z 方向以其負的標準化值乘以 0.2 的速度，垂直移動 0.5 秒。最後這邊因為紅框或是籃框有邊的厚度，所以我們的鏡頭中心有設一個假的中心 fakey=170。

Forward 是負責前進，我們預設以 0.2 的速度前進 0.5 秒。

Addsp 是負責加速前進，我們先以 0.4 的速度前進 4 秒，再以 0.5 的速度前進 4.3 秒。

Rotate 則是負責順時鐘旋轉，我們每次旋轉 0.4 度，轉 0.6 秒。這邊不能一次旋轉太多，因為有可能會一下子錯過框的正中心位置轉過頭。

Trace 是負責沿著綠色的邊線。我們對於垂直方向的邏輯設定和 correction 一樣，但這邊的垂直中心是用鏡頭中心的一半再加上 20，這樣可以使無人機在飛時比中心點稍微飛高一點，不然最後無人機到達綠色邊線的盡頭時有可能會因為飛的稍低，掃不到 apriltag。垂直方向的速度是 0.1。水平方向我們都預設一直往左做 0.1 速度的前進，因為關卡的設計是那樣使我們可以這樣做，還有就是因為綠色的色塊在沿著邊線前進時會因為鏡頭的左右都有色塊而一直跳動，這對於演算法的設計增加一定的難度。我們前進 0.5 秒。

Traceback 也是一樣的原理，只是水平速度變成往右 0.1。

```
class MyModel(object):
    def __init__(self, state):
        self.state = state

    def run(self, fsm):
        while not rospy.is_shutdown():
            print(self.state)

            ############################
            if fsm.is_trace:
             if t1.state.mission==2:
              fsm.need2trace()
             else: #3
              fsm.need2traceback()
            ########

            elif fsm.is_traceback:
             if t1.state.mission==3:
              fsm.need2traceback()
             else:
              fsm.stop2do()
```

如果你是在 trace 這個 state, 當你下一個 state 的 mission 還是 2 時，你會繼續執行 trace, 反之代表你已經掃到 apriltag 了，會進入 traceback 的動作，沿著綠色邊線返回。

如果你是在 traceback 這個 state，到妳因為已經掃不到綠色的邊線後，你的 mission 才會改為 1 或 2，state 才藉由 stop2do 變成 hover 的 state，準備進行之後的旋轉及過框。否則，你就會持續 mission=3，持續沿著邊線返回。

```
elif fsm.is_hover:

    if t1.state.mission==2:
     fsm.need2trace()
    elif t1.state.target_x == -1 and t1.state.target_y == -1:
     if t1.state.hasPassedTwice==0:
      fsm.wait4data()
     else:
      fsm.need2rotate()

    else:
      dx = t1.state.target_x - 480
      dy = t1.state.target_y - fakey
      if t1.state.cammiddle == 1:
       if abs(dx) < 30 and abs(dy) < 30:
        fsm.start2forawrd()
       else:
        fsm.start2correct()

      else:
       fsm.need2rotate()
```

當進入 hover 的 state 之後，我們會先去判斷你現在的 mission 是不是 2，即沿著綠色邊線找 apriltag。如果是，代表的是你可能剛過第一個紅框，於是我們先轉到 trace 的 state。否則如果你有掃到色塊，你會進行前進、校正鏡頭，或是旋轉的動作。

我們把 t1.state.target_x ，即色塊的水平座標和鏡頭中心 480 去做相減，變成 dx。色塊垂直座標 T1.state.target_y 和鏡頭中心 fakey 做相減，變成 dy。如果這個 dx 和 dy 太大，即小於 30，代表無人機已經和中心相距不遠，於是我們用 forward 朝框邁進，但如果相距太遠，我們就進入 correction 的階段進行校正。

如果鏡頭掃不到色塊，或是即使掃到色塊，但佔比太小，我們就會讓無人機去旋轉 need2rotate，試圖對準目標。

```
elif fsm.is_correction:

    if t1.state.canPass == 1:
        fsm.need2addSp()
    else:
        dx = t1.state.target_x - 480
        dy = t1.state.target_y - fakey

        if t1.state.cammiddle == 1:
         if abs(dx) < 30 and abs(dy) < 30:
          fsm.need2forawrd()
         else:
          fsm.need2correct()
        else:
         fsm.need2rotate()
```

同樣的，在 correction 的 state，如果校正的不夠多，我們會繼續進行校正。如果夠多了，但比例還沒到 canpass 為 1 的地步，我們會 forward。如果接近目標到 canpass 的 state 變成 1，我們則會加速前進。如果掃不到色塊或是佔比太小，我們會旋轉 need2rotate。

```
elif fsm.is_forward:
    if t1.state.canPass == 1:
        fsm.need2addSp()
    else:
        dx = t1.state.target_x - 480
        dy = t1.state.target_y - fakey

        if t1.state.cammiddle == 1:
         if abs(dx) < 30 and abs(dy) < 30:
          fsm.need2forawrd()
         else:
          fsm.need2correct()

        else:
         fsm.need2rotate()

##############################################
elif fsm.is_addSp:
    if t1.state.hasPassedTwice == 2:
        break
    else:
        fsm.stop2do()
```

進入 Forward 的 state 後，其內的邏輯判斷，state 的轉換也和方才 correction 一樣。

進入 addsp 加速的 state 之後，如果已經過了兩次框，我們會 break 整個 while loop。使整個程式中止，完成降落的動作。如果還沒通過框兩次，代表你只通過框一次而已，我們會先藉由 stop2do 進入 hover 的 state，等待下次任務的執行。

```
elif fsm.is_rotate:
    if t1.state.cammiddle == 0:
        fsm.need2rotate()
    else:
        fsm.refly()
```

進入旋轉這個 state 之後，如果還是沒有掃到目標色塊，我們會繼續 rotate。否則如果掃到，我們會藉由 refly 進入 hover 的 state，重新進行之後的任務。

```
def main():


    while t1.state.is_flying == False:
        t1.controler.takeoff()

    while t1.state.fly_mode != 6:
        print("wait...")

    obj = MyModel(state='hover')
    fsm = sMachine(obj)
    obj.run(fsm)

    while t1.state.fly_mode != 6:
        print("wait...")

    while t1.state.is_flying == True:
        t1.controler.land()

if __name__ == '__main__':
    rospy.init_node('h264_pub', anonymous=True)
    main()
```

這邊是整個 main function。在無人機起飛之後，我們會去執行過框、沿著綠色邊線找 apriltag、沿著綠色邊線返回、過第二次框的任務，最後降落，完成任務。