第五章：物體辨識與 ROS Package 應用

蘇詠善

本章節將帶領學員活用 Open Source Computer Vision Library (OpenCV) 實踐物體辨識，並把 OpenCV 與 ROS Package 結合在一起，讓車子執行 123 木頭人。此章節一共分為三大部分：物體辨識、如何創造 ROS Package、123 木頭人專題教學。在學習完此章節後你將能熟悉 OpenCV、Python、Jupyter-Notebook、ROS Package。

1 物體偵測

課程目標：學生能熟悉如何使用 OpenCV，並在程式平台 Jupyter-Notebook 上實踐。
需先備知識與需先安裝好的資料庫：
程式平台 Jupyter-Notebook
開源電腦視覺函式庫 OpenCV
開源程式 Python

1.1 基本介紹：

Jupyter-Notebook 為一個十分便於編譯 python 的平台，它不僅有圖形介面且也能即時的顯現執行程式的結果，因此為了讓學員們能熟悉運用 OpenCV，本章將以 Juputer-Notebook 為平台一步一步帶領大家熟悉 OpenCV，並運用此平台實踐物體偵測的演算法。

步驟一、開啟 Jupyter-Notebook
duckietown $ cd /duckietown/tutorials/python/
進入到範例程式檔所處的資料夾
duckietown $ jupyter-notebook 01-tutorial-object-detector.ipynb
開啟範例程式

步驟二、讀取所有所需的函式庫
由於接下來要寫的程式會運用到大量的函式庫，此區的程式將會讀取數學函式庫 (numpy)、科學計算函式庫 (scipy)、開源電腦視覺函式庫 (cv2)、時間函式庫 (time)、繪圖函式庫 (pyplot)

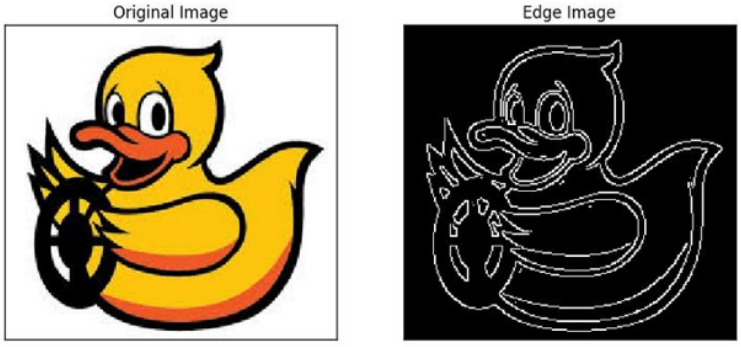
Import Packages

```
In [1]: import numpy as np
        import scipy as sp
        import cv2
        import time
        from matplotlib import pyplot as plt
        %matplotlib inline
        # set display defaults
        plt.rcParams['figure.figsize'] = (10, 10)        # large images
        plt.rcParams['image.interpolation'] = 'nearest'  # don't interpolate: show square pixels
```

步驟三、執行範例程式
邊緣偵測
在物件偵測中，邊緣偵測是一個相當重要的工具，因為它能將圖形的細節給表現出來，因此這部分所示範的是將一張圖片讀取後，進行坎尼邊緣偵測 (Canny edge detection)，並將結果與原圖視覺化以進行比較。

Basic OpenCV

```
In [3]: img = cv2.imread('duckietown.jpg')
edges = cv2.Canny(img,140,200)

dst1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.subplot(121),plt.imshow(dst1, cmap = 'brg')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```



藉由圓形圖案來辨識車子

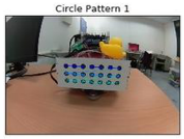
本小節我們會在車子背後貼圓形圖案，並藉由斑點辨識 (blob detection) 來辨識圓形圖案。程式將會先將圖片讀取進來後，再宣告一些基本變數，最後使用 OpenCV 函式庫所提供的斑點辨識函式來辨識圓形圖案，並將辨識到的圓形圖案視覺化。

Vehicle (Circle Pattern) Detector

See detail in vehicle_detection/src/vehicle_detection_node.py

```
In [6]: img = cv2.imread('01-tutorial/circle.jpg')

params = cv2.SimpleBlobDetector_Params()
circlepattern_dims = tuple([7, 3])
params.minArea = 10
params.minDistBetweenBlobs = 2
simple_blob_detector = cv2.SimpleBlobDetector(params)

%time (detection, corners) = cv2.findCirclesGrid(img, circlepattern_dims, flags=cv2.CALIB_CB_SYMMETRIC_GRID,bl

# Visualization
dst = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

cv2.drawChessboardCorners(dst, circlepattern_dims, corners, detection)

plt.subplot(311),plt.imshow(dst, cmap = 'brg')
plt.title('Circle Pattern 1'), plt.xticks([]), plt.yticks([])

plt.show()
```

```
CPU times: user 40 ms, sys: 0 ns, total: 40 ms
Wall time: 42.1 ms
```



函式代表意義:

cv2.simpleBlobDetector_Params() : 設定斑點辨識 (blob detector) 的參數

tuple() : 設定要找到的圓圈的行與列.

params.minArea() : 設定斑點辨識要尋找圈圈涵蓋的最小區域

params.minDistBetweenBlobs : 設定斑點辨識要尋找圈圈之間的最短距離

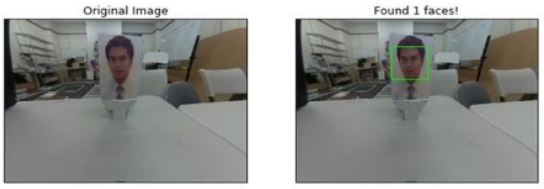
cv2.findCirclesGrid : 找到圓圈
cv2.drawChessboardCorners : 在偵測到的圈圈之間畫線

臉部偵測
本小節我們使用了 OpenCV 裡的串接式決策分類器 (Cascade Classifier)，使用現成的訓練模型來辨識人臉。因此在這部分的程式，會先讀取圖片及訓練模型，在設定一些基本參數後，進行辨識。最後再將人臉框出來，並視覺化。

Face Detector

See detail in arg_nctu/wama/face_detector_wama/src/face_detector_wama_node.py

```
In [4]: img = cv2.imread('01-tutorial/face.jpg')
        vis = cv2.imread('01-tutorial/face.jpg')

        faceCascade = cv2.CascadeClassifier('01-tutorial/haarcascade_frontalface_default.xml')
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        %time faces = faceCascade.detectMultiScale(gray, scaleFactor=2, minNeighbors=5, minSize=(10, 10), flags = cv2.

        # Visualization
        dst = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        dst1 = cv2.cvtColor(vis, cv2.COLOR_BGR2RGB)
        for (x, y, w, h) in faces:
            cv2.rectangle(dst1, (x, y), (x+w, y+h), (0, 255, 0), 2)


        plt.subplot(121),plt.imshow(dst, cmap = 'brg')
        plt.title('Original Image'), plt.xticks([]), plt.yticks([])
        plt.subplot(122),plt.imshow(dst1, cmap = 'brg')
        plt.title("Found {0} faces!".format(len(faces))), plt.xticks([]), plt.yticks([])

        plt.show()

CPU times: user 72 ms, sys: 4 ms, total: 76 ms
Wall time: 78 ms
```



函式代表意義:
cv2.CascadeClassifier() : 讀取串接式決策分類器 (cascade classifier)
detectMultiscale() : 設定串接式決策分類器 (cascade classifier) 參數
scaleFactor : 設定要將圖片放大幾倍
minNeighbors : 指定每個候選矩陣至少包含的鄰近元素個數
minSize / maxSize : 設定偵測物體在圖片中尺寸的極值
flags : reference/tag refer to the old feature
cv2.rectangle() : 畫出一個矩形

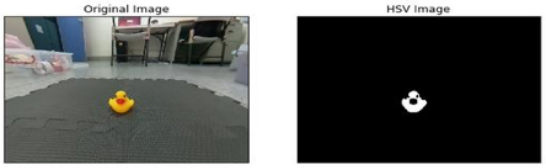
物體辨識 (以鴨子為例)
這一部分會分解成三個小部分，第一部分為運用 HSV 色彩辨識偵測鴨子。第二部分則是寫下運用輪廓來辨識鴨子的演算法，最後一部分則是使用第二部分的演算法來偵測鴨子。
第一部分先讀取圖片後，將圖片轉成 HSV 色彩空間，再藉由 HSV 色彩辨識的方式將鴨子身體的顏色 (黃色) 給濾出來，最後進行視覺化。
函式代表意義:
np.array : 建立新的陣列
cv2.inRange() : 設定範圍
cv2.threshold() : 設定門檻值

Duckie Detector

See detail in src/mdoap/src/static_object_detector_node.py

You may need to change the high/low color thresholds

In [8]:
```
CONE = [np.array(x, np.uint8) for x in [[0,80,80], [22, 255,255]] ]
DUCK = [np.array(x, np.uint8) for x in [[15,100,150], [35, 255, 255]] ]

img = cv2.imread('01-tutorial/duckie_1.jpg')
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
frame_threshed = cv2.inRange(hsv_img, DUCK[0], DUCK[1])
ret,thresh = cv2.threshold(frame_threshed,15,255,0)

dst1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.subplot(121),plt.imshow(dst1, cmap = 'brg')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(frame_threshed, cmap = 'gray')
plt.title('HSV Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

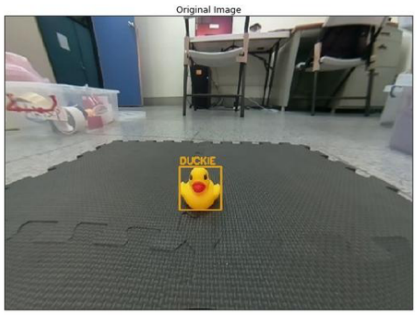


第二部分則是鴨子偵測演算法的函式，待會會被使用到。
函式代表意義:
cv2.findcontours() : 找到二元影像的輪廓
cv2.RETR_CCOMP() : 建立兩個等级的輪廓，上面的一層為邊界，里面的一層為內孔的邊界信息。
cv2.CHAIN_APPROX_SIMPLE() : 壓縮水平、垂直、對角方向的元素，只保留該方向的終點座標
cv2.contourArea() : 輪廓面積
cv2.boundingRect() : 計算邊界矩形
cv2.arcLength() : 計算輪廓周長
cv2.drawContours() : 使用得到的點劃出輪廓與形狀

最後一部分將用前一部分所寫好的函式來偵測鴨子的輪廓，如果偵測到的輪廓有吻合鴨子的輪廓，程式將會將鴨子偵測出來，且框出它，並視覺化。

Let's detect duckie!

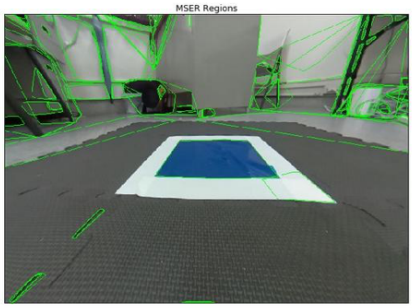
In [10]:
```
img = cv2.imread('01-tutorial/duckie_1.jpg')

duck_contours = get_filtered_contours(img, "DUCK_COLOR")

for (cnt, box, ds, aspect_ratio, mean_color)  in duck_contours:
    # plot box around contour
    x,y,w,h = box
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, "DUCKIE", (x,y-5), font, 0.5, mean_color, 2)
    cv2.rectangle(img,(x,y),(x+w,y+h), mean_color,2)

dst1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.subplot(111),plt.imshow(dst1, cmap = 'brg')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.show()
```



最大穩定極值區域 (Maximally Stable Extremal Regions)
此部分為示範運用最大穩定極值區域的概念來將圖片切割成許多小塊，每一個小塊都是一個輪廓。程式中，會先讀取圖片，並將圖片轉成灰階圖，接著使用 OpenCV 的 MSER 函式來將圖片切成許多小塊的輪廓並視覺化。

Parking Space (Blue Region) Detector

In [11]:
```
img = cv2.imread('01-tutorial/park.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
dst1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

vis = dst1.copy()
mser = cv2.MSER()
regions = mser.detect(gray, None)
hulls = [cv2.convexHull(p.reshape(-1, 1, 2)) for p in regions]
cv2.polylines(vis, hulls, 1, (0, 255, 0))

plt.subplot(111),plt.imshow(vis,cmap = 'gray')
plt.title('MSER Regions'), plt.xticks([]), plt.yticks([])

plt.show()
```



函式代表意義: cv2.MSER() : 回傳所有圖片中偵測到的點 cv2.convexHull() : 將那些點連接起來

1.2 牛刀小試:

臉部偵測器乃本章最為複雜的演算法之一，請你嘗試善用本章教過的串接式決策分類器來對以下兩張圖片進行臉部辨識，將所有的臉部都偵測出來。
(提示：針對不同的圖片，參數的調整也都會不一樣。)

2 創造一個 ROS package (Python)

課程目標：學生能熟悉如何創造 ROS package
需先備知識與需先安裝好的資料庫：
需先安裝好機器人作業系統 (ROS)

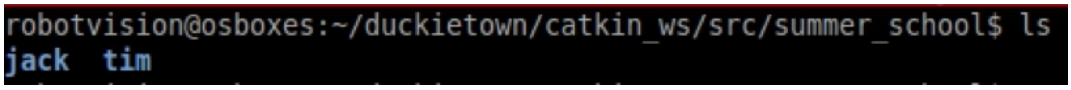
2.1 ROS 基本介紹：

ROS 是現今最流行的機器人作業系統，國際名校如史丹佛大學、麻省理工學院皆廣泛使用這套系統。我們已於前幾章介紹過 ROS 的概念，在本小節我們將進行實作。我們會先學習創建一個 ROS package 需要打那些指令、引進不同的函式庫需要去改那些檔案。最後去學習將事先已寫好的 launch 檔進行修改，然後跑起來。我們一共會分為六個步驟：第一步驟會先把主要的 package 先創立出來。第二步驟會創造出一個 launch 檔，目的是去呼叫第三步驟會去創立的 package 中的 launch，這個方式在往後會大量應用。第三、四、五步驟則是創立一個新的 package 及 launch 檔，並且會實作如何修改”CMakeLists.txt”與”package.xml”去連結函式庫，第六步驟是執行剛剛寫好的 launch 檔，並使用 ROS 的圖形化模擬環境 RVIZ 去將傳輸的訊息視覺化。

2.2 ROS 實驗步驟：

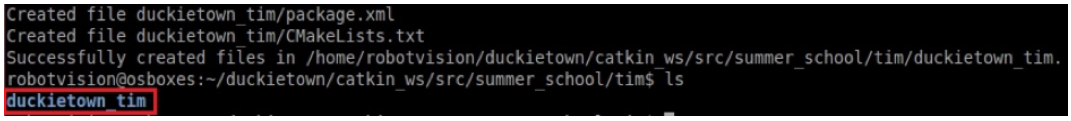
步驟零、初始化 ROS 環境與 ROS Master
此步驟是為了設定好 ROS 的環境，我們將 ROS 的環境架在”duckietown”這個資料夾底下，這個資料夾裡有兩個 bash 檔”environment.sh”和”set_ros_master.sh”，在使用 ROS 前我們必須先執行過這兩個 bash 檔，才能把 ROS 基本環境設定好，接著才能做之後創建 ROS package 的動作。
laptop $ssh ubuntu@duckiebot.local
(指令解釋：用筆電連線到 Pi 上，進行遠端控制)
duckiebot $ cd /duckietown
duckiebot $ source environment.sh
(指令解釋：執行 bash 檔”environment.sh”)
duckiebot $ source set_ros_master.sh [duckiebot]
(指令解釋：執行 bash 檔”set_ros_master.sh”)

步驟一、創造你的 Ros package
此步驟的目的是要創建一個完整的 ROS package，因此我們使用到 catkin_create_pkg”這個指令，它可以幫我們自動把 package 建好，且還會幫我們自動加入兩個檔案(“CMakeLists.txt”與”package.xml”)，這兩個檔案定義了我們的程式可以使用哪些函式庫，因此十分重要。未來若想連結其他函式庫，也需去更改這兩個檔案。
duckiebot $ cd /duckietown/catkin_ws/src/summerschool2017_nctu/
duckiebot $ mkdir [duckiebot]
(指令解釋：創建新目錄”duckiebot”，創完後輸入指令”ls”，你會看到一個資料夾 (以 duckiebot 的名字來命名，那是你剛才創的)

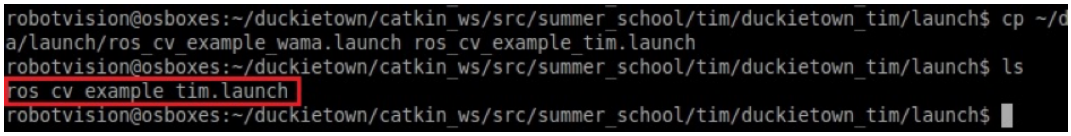


duckiebot $ cd /duckietown/catkin_ws/src/summerschool2017_nctu/[duckiebot]
duckiebot $ catkin_create_pkg duckietown_[duckiebot]
(指令解釋：創造 ROS package “duckiebot”)

duckiebot $ cd duckietown_[duckiebot]

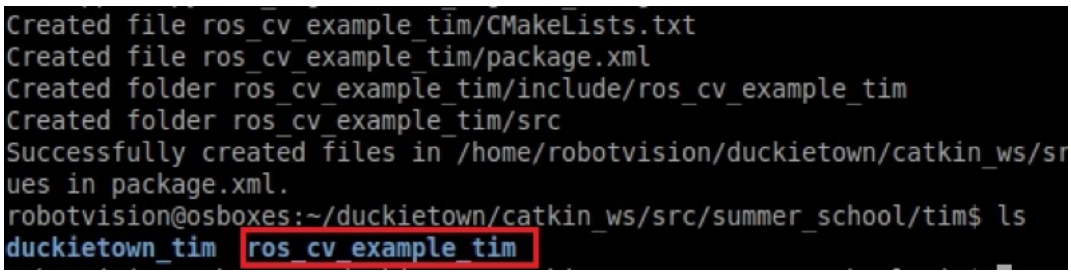


步驟二、在”duckietown_[duckiebot]”這個目錄底下創造名為 launch 的資料夾
前幾章已說明過 launch 對 ROS 的重要性與概念，因此為了建立 launch 檔，我們習慣先建立一個名為”launch”的資料夾，在這資料夾底下再去建立 launch 檔，因此在此步驟中便是教大家先建立資料夾，再去複製我們的範例 launch 檔到我們剛剛建立的資料夾底下。最後簡單修改範例 launch 檔後便完成。
duckiebot $ mkdir launch
duckiebot $ cd launch
duckiebot $ cp /duckietown/catkin_ws/src/spring2016_nctu/wama/duckietown_wama/launch/ros_cv_example_wama.launch ros_cv_example_[duckiebot].launch (指令解釋：將事先已寫好的範例程式複製到此資料夾底下，複製好後輸入指令”ls”，你會看到以下檔案 (ros_cv_example_[duckiebit].launch)



duckiebot $ vim ros_cv_example_[duckiebot].launch
Modify the following: duckiebot=your duckiebot name
<!– ros_cv_example_duckiebot–>
<remap from=”ros_cv_example_duckiebot_node/image” to=”camera_node/image/compressed”/>
<include file=”$(find ros_cv_example_duckiebot)/launch/ros_cv_example_duckiebot_node.launch”>
(指令解釋：打開 launch 檔後需要按照以上方式將所有”duckiebot”改寫成你鴨子車的名字。)

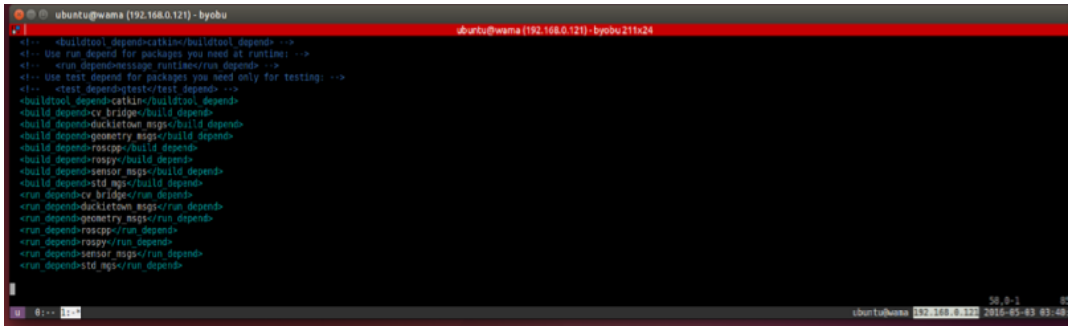
步驟三、創造一個名為 ros_cv_example_[duckiebot] 的 ros package
在此步驟中，我們會再創立一個新的 ROS package，在這個 package 裡，我們會先實作修改”CMakeLists.txt”與”package.xml”，將所有的資料庫都鏈結起來。
duckiebot $ cd /duckietown/catkin_ws/src/summerschool2017_nctu/[duckiebot]
duckiebot $ catkin_create_pkg ros_cv_example_duckiebot
duckietown_msgs geometry_msgs roscpp rospy std_msgs sensor_msgs cv_bridge (指令解釋：創造 ROS package “ros_cv_example_duckiebot”，且加入一些需要用到的函式庫，輸入以上指令後你會看到以下結果)



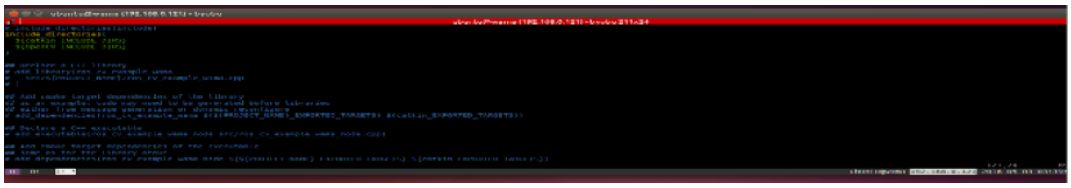
(附註：加入的函式庫”duckietown_msgs”、”geometry_msgs”、”std_msgs”、”sensor_msgs”是 ROS 常用到傳

遞訊息的 ROS message 的函式庫，而”roscpp”、”rospy”、”cv_bridge”則是 ROS 與 OpenCV 的函式庫)

duckiebot $ cd ros_cv_example_duckiebot/
duckiebot $ vim package.xml (指令解釋：打開檔案後你會看到程式(如下)，裡面有許多函式庫作為我們之後寫的程式的函式庫 dependency)



duckiebot $ vim CMakeLists.txt (指令解釋：打開檔案後你會看到程式(如下)，請加入這一行“find_package(OpenCV REQUIRED)”與加入這一行 $OpenCV_INCLUDE_DIRS”如以下幾張圖的方式，目的是為了去找到函式庫 OpenCV，如此之後去使用此函式庫才沒有問題。)



duckiebot $ cd src
duckiebot $ cp /duckietown/catkin_ws/src/spring2016_nctu/wama/ros_cv_example_wama/src/ros_cv_example_wama_node.py ros_cv_example_duckiebot_node.py
(指令解釋：將範例 python 程式複製此目錄底下)

步驟四、在此目錄”ros_cv_example”創造一個名為 launch 的資料夾
我們會像步驟二一樣去建立 launch 這個資料夾，之後把之前已經寫好的 launch 檔複製到此資料夾底下，再進行基本的修改。
duckiebot $ cd /duckietown/catkin_ws/src/summerschool2017_nctu/duckiebot/ros_cv_example
duckiebot $ mkdir launch
duckiebot $ cd launch
duckiebot $ cp /duckietown/catkin_ws/src/spring2016_nctu/wama/ros_cv_example_wama/launch/ros_cv_example_wama_node.launch ros_cv_example_duckiebot_node.launch
duckiebot $ vim ros_cv_example_duckiebot_node.launch
Modify the following: duckiebot=your duckiebot name
    <arg name=”pkg_name” default=”ros_cv_example_duckiebot” doc=”name of the package”/>
    <arg name=”node_name” default=”ros_cv_example_duckiebot_node” doc=”name of the node”/>
(指令解釋：複製範例 node 檔到這個資料夾，並進行以上修改，目的是因為每個人的鴨子車名都不一樣，因此需要修改。)

步驟五、使用 catkin_make 來編譯剛剛建立的 package 還有寫好的程式
在 ROS 中，當我們創建好 package 與寫完程式後，為了編譯他們，我們使用 catkin_make(可以看成是 cmake 的進階版)，如果 catkin_make 都順利過了，代表程式正確、資料庫鏈結成功，就可以去放心執行了。
duckiebot $ cd /duckietown/catkin_ws
(指令解釋：只有進入這個目錄才能進行 catkin_make)
duckiebot $ catkin_make
(指令解釋：進行 catkin_make 編譯所有的在此目錄底下的程式)

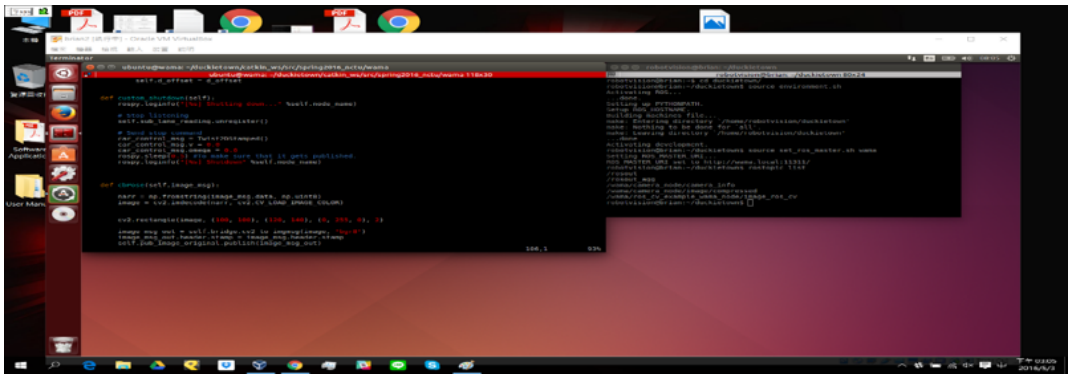
步驟六、用 launch 檔執行 package 裡的程式
在此步驟中，我們要來執行剛剛寫好的程式與 launch 檔。我們會先在鴨子車執行 launch 檔，之後在筆電上使用 ROS 的圖形化模擬環境 RVIZ 來把傳送的訊息視覺化。
duckiebot $ roslaunch duckietown_duckiebot ros_cv_example_tim.launch veh:=[duckiebot]
(指令解釋：執行 launch 檔)
(附註: 以下步驟在筆電上執行)
laptop $ cd /duckietown
laptop $ source environment.sh
(指令解釋：執行 bash 檔”environment.sh”)
laptop $ source set_ros_master.sh duckiebot
(指令解釋：執行 bash 檔”set_ros_master.sh”)
laptop $ rviz (指令解釋：開啟圖形化模擬環境 rviz)

RVIZ 操作解釋、
在 RVIZ 中請按 add 鍵加入 topic”/duckiebot/ros_cv_example_duckiebot_node/image_ros_cv”，加入後便可以看到圖片。

2.3 牛刀小試：

相信經由這一節的解釋，你已了解如何實作 ROS package，因此希望你可以使用在本節已創好的 ROS package，然後修改執行的 python 檔”ros_cv_example_duckiebot_node.py”讓你的程式辨識人臉並顯示在 RVIZ 上。

提示、
藉由以下指令你可以加入一些 OpenCV 的函式:
duckiebot $ vim /duckietown/catkin_ws/src/summerschool2017_nctu/duckiebot/ros_cv_example_duckiebot/src/ros_cv_example_duckiebot_node.py



例如 cv2.rectangle 可以幫你的圖形畫一個矩形

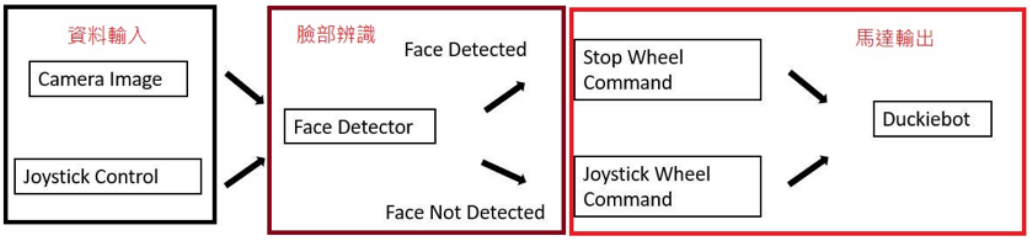
3 1-2-3 木頭人

在學習玩前兩節的 OpenCV 物體辨識演算法與完整建立一個 ROS package 並且編譯執行後，接著我們要來實作一個專題：1-2-3 木頭人，此專題會運用前兩節所學，並且延伸，目標最終是使用搖桿來操作 duckiebot，並且如果它有看到人臉它會自動停止。

先備知識與工具：
臉部偵測器 (jupyter-notebook)
原本就有的 duckietown package
duckietown camera.launch
duckietown joystick.launch
你自己的 ROS package
如何去建立 Ros package
face_detector_duckiebot.launch
如何去建立 ros launch
onetwothree_stop_duckiebot.launch

3.1 題目介紹：

架構
在此專題中輸入為相機拍攝的影像與搖桿控制，演算法的部分是臉部偵測會持續地進行，如果臉部沒有被偵測到，搖桿控制訊號將會轉成車子馬達的訊號給車子馬達，車子便會移動，但若是有偵測到臉，演算法將會傳送一個停止的訊號給車子馬達，車子便會停下來。



3.2 實驗步驟：

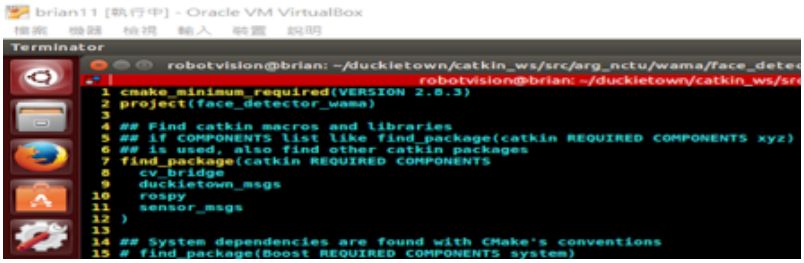
在本次專題練習裡，大部分的程式皆已寫好，但有些讓學生能進行填空已做為對 ROS package、ROS node、ROS launch 的概念練習，在此四大步驟裡，我們會反覆訓練到如何創建 ROS package、如何連結資料庫，如何將 ROS node 與 launch 修改完整，最後使用一個 launch 去呼叫諸多 ROS package 的 launch 與 node 完成 1-2-3 木頭人的任務。

步驟一、建立一個 launch 檔 face_detector_duckiebot_node.launch
duckiebot $ cd /duckietown
duckiebot $ source environment.sh
(指令解釋：執行 bash 檔”environment.sh”)
duckiebot $ source set_ros_master.sh [duckiebot]

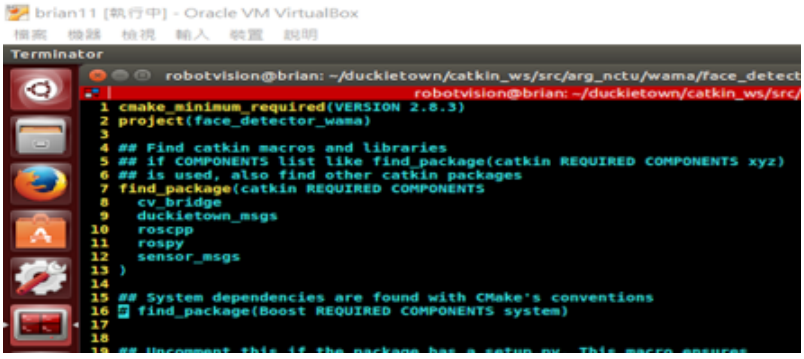
(指令解釋：執行 bash 檔”set_ros_master.sh”)
duckiebot $ cd /duckietown/catkin_ws/src/summer_school/duckiebot

步驟二、建立一個 ros package
duckiebot $ catkin_create_pkg face_detector_duckiebot duckietown_msgs sensor_msgs rospy cv_bridge
(指令解釋：創造 ROS package“face_detector_duckiebot”，且加入一些需要用到的函式庫。)
duckiebot $ cd /duckietown/catkin_ws/src/summer_school/duckiebot /face_detector_ duckiebot
duckiebot $ vim CMakeLists.txt
(指令解釋：修改”CMakeLists.txt”，增加“$OpenCV_INCLUDE_DIRS”與 roscpp 如下圖)

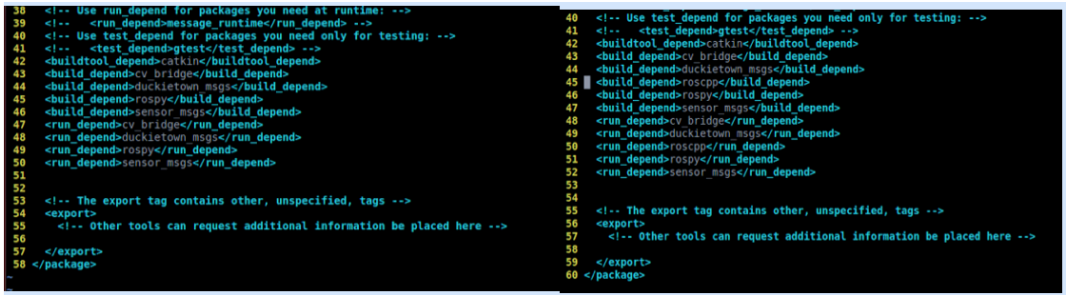
前：



後：



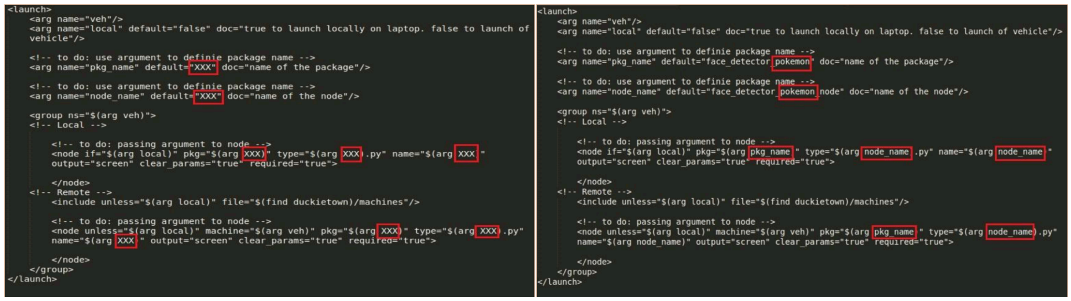
duckiebot $ cd /duckietown/catkin_ws/src/summer_school/duckiebot/face_detector_duckiebot
duckiebot $ vim package.xml (指令解釋：修改”package.xml”，增加“roscpp”如下圖)



```
<!-- Use run_depend for packages you need at runtime: -->
<!--   <run_depend>message_runtime</run_depend> -->
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>cv_bridge</build_depend>
<build_depend>duckietown_msgs</build_depend>
<build_depend>rospy</build_depend>
<build_depend>sensor_msgs</build_depend>
<run_depend>cv_bridge</run_depend>
<run_depend>duckietown_msgs</run_depend>
<run_depend>rospy</run_depend>
<run_depend>sensor_msgs</run_depend>


<!-- The export tag contains other, unspecified, tags -->
<export>
  <!-- Other tools can request additional information be placed here -->

</export>
</package>
```

```
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>cv_bridge</build_depend>
<build_depend>duckietown_msgs</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>sensor_msgs</build_depend>
<run_depend>cv_bridge</run_depend>
<run_depend>duckietown_msgs</run_depend>
<run_depend>roscpp</run_depend>
<run_depend>rospy</run_depend>
<run_depend>sensor_msgs</run_depend>


<!-- The export tag contains other, unspecified, tags -->
<export>
  <!-- Other tools can request additional information be placed here -->

</export>
</package>
```

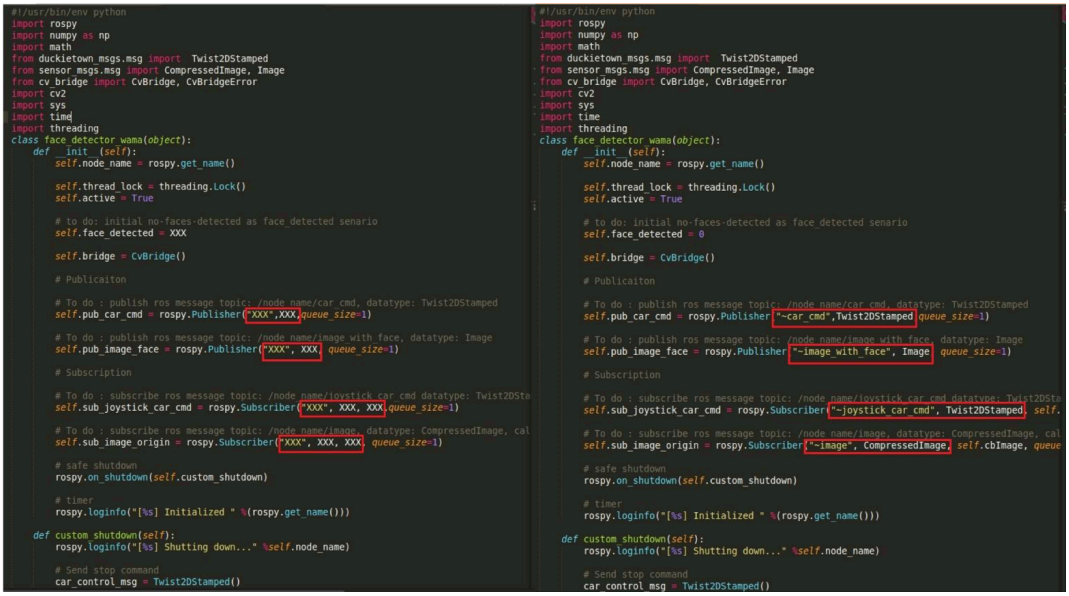
duckiebot $ cd /duckietown/catkin_ws/src/summer_school/duckiebot /face_detector_ duckiebot
duckiebot $ mkdir launch
duckiebot $ cd /duckietown/catkin_ws/src/summer_school/duckiebot/face_detector_duckiebot/launch
duckiebot $ cp /duckietown/tutorials/ros/face_detector_tutorial_node.launch face_detector_detector_node.launch
duckiebot $ vim face_detector_tutorial_node.launch
(指令解釋：開啟”face_detector_tutorial_node.launch”, 並將程式裡所有 XXX 修改成你的鴨子車名)



```
<launch>
    <arg name="veh"/>
    <arg name="local" default="false" doc="true to launch locally on laptop. false to launch of
    vehicle"/>

    <!-- to do: use argument to definie package name -->
    <arg name="pkg_name" default="XXX" doc="name of the package"/>

    <!-- to do: use argument to definie package name -->
    <arg name="node_name" default="XXX" doc="name of the node"/>

    <group ns="$(arg veh)">
    <!-- Local -->

        <!-- to do: passing argument to node -->
        <node if="$(arg local)" pkg="$(arg XXX)" type="$(arg XXX).py" name="$(arg XXX)"
        output="screen" clear_params="true" required="true">

        </node>
    <!-- Remote -->
        <include unless="$(arg local)" file="$(find duckietown)/machines"/>

        <!-- to do: passing argument to node -->
        <node unless="$(arg local)" machine="$(arg veh)" pkg="$(arg XXX)" type="$(arg XXX).py"
        name="$(arg XXX)" output="screen" clear_params="true" required="true">

        </node>
    </group>
</launch>
```

```
<launch>
    <arg name="veh"/>
    <arg name="local" default="false" doc="true to launch locally on laptop. false to launch of vehicle"/>

    <!-- to do: use argument to definie package name -->
    <arg name="pkg_name" default="face_detector_pokemon" doc="name of the package"/>

    <!-- to do: use argument to definie package name -->
    <arg name="node_name" default="face_detector_pokemon_node" doc="name of the node"/>

    <group ns="$(arg veh)">
    <!-- Local -->

        <!-- to do: passing argument to node -->
        <node if="$(arg local)" pkg="$(arg pkg_name)" type="$(arg node_name).py" name="$(arg node_name)"
        output="screen" clear_params="true" required="true">

        </node>
    <!-- Remote -->
        <include unless="$(arg local)" file="$(find duckietown)/machines"/>

        <!-- to do: passing argument to node -->
        <node unless="$(arg local)" machine="$(arg veh)" pkg="$(arg pkg_name)" type="$(arg node_name).py"
        name="$(arg node_name)" output="screen" clear_params="true" required="true">

        </node>
    </group>
</launch>
```

步驟三、建立一個 python 檔 face_detector_duckiebot_node.py
duckiebot $ cd /duckietown/catkin_ws/src/summer_school/duckiebot/face_detector_duckiebot
(指令解釋：進入資料夾”face_detector_duckiebot”)
duckiebot $ mkdir src
(指令解釋：創建新資料夾”src”)
duckiebot $ cp /duckietown/tutorials/ros/face_detector_tutorial_node.py face_detector_duckiebot_node.py
(指令解釋：複製”face_detector_tutorial_node.py”到此資料夾底下)
duckiebot $ vim face_detector_duckiebot_node.py
(指令解釋：開啟 face_detector_duckiebot_node.py，並改成如下圖所示)



```
#!/usr/bin/env python
import rospy
import numpy as np
import math
from duckietown_msgs.msg import  Twist2DStamped
from sensor_msgs.msg import CompressedImage, Image
from cv_bridge import CvBridge, CvBridgeError
import cv2
import sys
import time
import threading
class face_detector_wama(object):
    def __init__(self):
        self.node_name = rospy.get_name()

        self.thread_lock = threading.Lock()
        self.active = True

        # to do: initial no-faces-detected as face_detected senario
        self.face_detected = XXX

        self.bridge = CvBridge()

        # Publicaiton

        # To do : publish ros message topic: /node_name/car_cmd, datatype: Twist2DStamped
        self.pub_car_cmd = rospy.Publisher("XXX",XXX,queue_size=1)

        # To do : publish ros message topic: /node_name/image_with_face, datatype: Image
        self.pub_image_face = rospy.Publisher("XXX", XXX, queue_size=1)

        # Subscription

        # To do : subscribe ros message topic: /node_name/joystick_car_cmd datatype: Twist2DSta
        self.sub_joystick_car_cmd = rospy.Subscriber("XXX", XXX, XXX,queue_size=1)

        # To do : subscribe ros message topic: /node_name/image_ datatype: CompressedImage, cal
        self.sub_image_origin = rospy.Subscriber("XXX", XXX, XXX, queue_size=1)

        # safe shutdown
        rospy.on_shutdown(self.custom_shutdown)

        # timer
        rospy.loginfo("[%s] Initialized " %(rospy.get_name()))

    def custom_shutdown(self):
        rospy.loginfo("[%s] Shutting down..." %self.node_name)

        # Send stop command
        car_control_msg = Twist2DStamped()
```

```
#!/usr/bin/env python
import rospy
import numpy as np
import math
from duckietown_msgs.msg import  Twist2DStamped
from sensor_msgs.msg import CompressedImage, Image
from cv_bridge import CvBridge, CvBridgeError
import cv2
import sys
import time
import threading
class face_detector_wama(object):
    def __init__(self):
        self.node_name = rospy.get_name()

        self.thread_lock = threading.Lock()
        self.active = True

        # to do: initial no-faces-detected as face_detected senario
        self.face_detected = 0

        self.bridge = CvBridge()

        # Publicaiton

        # To do : publish ros message topic: /node_name/car_cmd, datatype: Twist2DStamped
        self.pub_car_cmd = rospy.Publisher("~car_cmd",Twist2DStamped,queue_size=1)

        # To do : publish ros message topic: /node_name/image_with_face, datatype: Image
        self.pub_image_face = rospy.Publisher("~image_with_face", Image, queue_size=1)

        # Subscription

        # To do : subscribe ros message topic: /node_name/joystick_car_cmd datatype: Twist2DSta
        self.sub_joystick_car_cmd = rospy.Subscriber("~joystick_car_cmd", Twist2DStamped, self.

        # To do : subscribe ros message topic: /node_name/image_ datatype: CompressedImage, cal
        self.sub_image_origin = rospy.Subscriber("~image", CompressedImage, self.cbImage, queue

        # safe shutdown
        rospy.on_shutdown(self.custom_shutdown)

        # timer
        rospy.loginfo("[%s] Initialized " %(rospy.get_name()))

    def custom_shutdown(self):
        rospy.loginfo("[%s] Shutting down..." %self.node_name)

        # Send stop command
        car_control_msg = Twist2DStamped()
```

duckiebot $ cd /duckietown/catkin_ws
duckiebot $ catkin_make
(指令解釋：進行 catkin_make 編譯所有的在此目錄底下的程式)
duckiebot $ roslaunch face_detector_duckiebot face_detector_duckiebot_node.launch veh:= duckiebot
(指令解釋：執行 launch 檔確定以上操作是否成功)

步驟四、創建還有測試 onetwothree_stop_duckiebot.launch
duckiebot $ cd /duckietown/catkin_ws/src/fall2016_nctu/duckiebot
duckiebot $ catkin_create_pkg duckietown_nctu_duckiebot
(指令解釋：創造 ROS package “duckietown_nctu_duckiebot”)
duckiebot $ cd /duckietown/catkin_ws/src/fall2016_nctu/duckiebot /duckietown_nctu_duckiebot
duckiebot $ mkdir launch
(指令解釋：創建資料夾”launch”)
duckiebot $ cd /duckietown/catkin_ws/src/fall2016_nctu/duckiebot /duckietown_nctu_duckiebot/launch
duckiebot $ cp /duckietown/tutorials/ros/onetwothree_stop_tutorial.launch onetwothree_stop_duckiebot.launch
(指令解釋：複製”onetwothree_stop_tutorial.launch”到此資料夾裡)
duckiebot $ cd /duckietown/catkin_ws
duckiebot $ catkin_make
(指令解釋：進行 catkin_make 編譯所有的在此目錄底下的程式)
duckiebot $ roslaunch duckietown_nctu_duckiebot onetwothree_stop_duckiebot.launch veh:=duckiebot
(指令解釋：執行 launch 檔”onetwothree_stop_duckiebot.launch”)