

第三章 - Python 與 Jupyter Notebook /線段偵測/道路判斷與車體控制

呂承龍

1 Python 與 Jupyter Notebook

1.1 Python 介紹

Python 是現在非常廣用的程式語言之一，相對於其他程式語言而言，它具有較易上手且較易閱讀的特徵。Python 語言特別強調其可讀性，讓程式碼可以輕易地被他人理解。

另外，Python 與 C C++ Java 等語言很不同的一點是，Python 是一種直譯式的語言，意思是不需要先編譯 (Compile) 在執行，而是邊執行邊編譯。目前，Python 也被廣泛地使用在資料科學裡面。

1.2 Jupyter Notebook 介紹

Jupyter Notebook 是一個網路應用程式，可以在裡面書寫程式碼並在網路上做分享。對於我們而言，他最方便的地方在於能將程式碼分成一個一個的區塊 (Cell)，並且分別觀看每個區塊的輸出結果。因為這樣的分辨性，讓我們可以分別檢查每段的程式，是否有期望中的輸出抑或是挑出其中的 Bug。

不過要注意的是，Jupyter Notebook 只能使用直譯式語言，例如 python 等等。在使用上要特別注意。

1.3 小試身手！

開啟 Jupyter Notebook

接下來我們就來做一些練習，這裡會特別解釋 Duckietown 裡面會用到的部分。首先我們要先用 Jupyter notebook 打開練習用的檔案。

```
$ duckietop@ cd ~/duckietown/tutorials/summer2017_nctu/
```

```
$ duckietop@ jupyter notebook 4-1.python_tutorial.ipynb
```

快捷鍵教學

在快捷鍵的部分，我們按上下可以在不同的區間 (cell) 之間移動，按下 Enter 可以進入到 cell 裡面進行編輯，按下 ESC 可以離開。Shift+Enter 一起按下就可以執行該區段的程式碼。按下 a/b 可以分別在該 cell 的上面或下面加一個新的 cell，連按 dd 則可以刪除不需要的 cell。1/2/3 則可以分別加上不同粗細的註解。

以上就是常用的快捷鍵的部分，按下 h 就會重新顯示快捷鍵的列表，供大家使用。各位一定要自己試過，玩玩看，才會學得快喔！

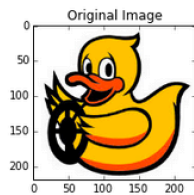
Cell 區間

我們來稍微解釋一下 cell 區間。請看下圖

每個區間都有一個程式碼的部分，請大家就把程式碼打在這裡。若該程式碼有輸出的話，下面則會顯示輸出的樣子。特別要注意的是旁邊的有一個 In[]，裡面可能有數字可能沒數字。數字代表的是執行的順序 (如果有被執行的話)。因為程式碼之間順序是有差別的，若大家在還沒呼叫函式庫就先使用該函式庫的函式的話，就會出錯。

```
In [1]: import cv2
from matplotlib import pyplot as plt
%matplotlib inline

img = cv2.imread('images/duckietown.jpg')
dst1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.subplot(121), plt.imshow(dst1, cmap = 'bgr')
plt.title('Original Image')
plt.show()
```



基礎 Python(print、迴圈、陣列)

開始要進入 Python 的部分了，首先我們先看 print 的部分，接下來的部分大家一定要自己動手玩玩看，調調裡面的參數，才會學得上手喔！

```
In [6]: print("Hello Duckietown!")
Hello Duckietown!
```

```
In [7]: x=1
print(x)
1
```

```
In [8]: for i in range(1, 10):
print i
1
2
3
4
5
6
7
8
9
```

```
In [9]: x=['B', 'L', 'F', 'F']
print(x[0])
B
```

首先我們可以發現到，python 在參數宣告的部分很簡單，不需要特別宣告變數型態 (例如：int, double 等等)，在 print 的地方也很簡單，只要直接將要輸入的變數放進去即可。如果要多個變數夾雜的話，可以用“，”隔開如下。

print x, ‘*’, x, ‘=’, x*x

另外再 for 迴圈的部分，會使用 in range(n, m) 來表達此迴圈的 i 要從 n 到 m 但不包含 m。

再提醒一下各位，Python 是一種不需要大括號的語言，也就是說他是靠縮排來表示不同部分，縮排可以用 tab 或是四個空白表示，但同一程式內只能用一種來當縮排喔！

接下來我們看一下陣列的部分

```

In [9]: x=['B', 'L', 'F', 'F']
        print(x[0])
B

In [10]: route = ""
        for i in x:
            if i != 'B':
                route += i
        print route
LFF

In [11]: x=0
        if x==1 :
            print("x equals 1")
        else :
            print("x doesn't equal 1")
x doesn't equal 1

```

我們可以看到，Python 要宣告一個陣列也是非常簡單的，也可以結合 for 迴圈一起使用，大家可以自己試試看。

Numpy 與 Scipy

Numpy 與 Scipy 是我們在分析與實驗常用到的兩個函式庫，我們一邊看圖一邊解釋

import lib

```

In [15]: import numpy as np
        import scipy as sp

```

array declaration

```

In [16]: x = np.array([1, 2, 3])
        for i in range(x.shape[0]) :
            print x[i]
        print x.shape, type(x), type(x[0])
1
2
3
(3,) <type 'numpy.ndarray'> <type 'numpy.int64'>

```

Guess what is the output of x.shape

```

In [17]: x = np.array([[1, 2, 3], [4, 5, 6]])
        for i in range(x.shape[0]) :
            for j in range(x.shape[1]) :
                print x[i][j]
        print x.shape, type(x), type(x[0]), type(x[0][0])
1
2
3
4
5
6
(2, 3) <type 'numpy.ndarray'> <type 'numpy.ndarray'> <type 'numpy.int64'>

```

我們要使用函式庫首先就要 import 他，後面的 as np, as sp 則是代表當我們之後要使用該函式庫的函式時，我們可以用 np 取代 numpy，待會我們就會看到範例。

首先第一部分就是 numpy 的陣列，他的陣列夠強大，可以方便我們使用。

接著我們可以看到一個關鍵字 `x.shape`，他的輸出會是 `x` 陣列個維度的大小，所以像是我們第一個 cell 的陣列只有一維、該維長度是 3，所以 `x.shape` 輸出會是 `(3,)`。下面一個 cell 就可以看到若有多維的陣列輸出會是什麼樣子。

至於 `type()`，則可以輸出該資料的型態，裡面可以是變數，也可以是陣列等等。

```
In [18]: x = np.zeros((3, 4))
print x

[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
```

```
In [19]: x = np.ones((3, 4))
print x

[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
```

```
In [20]: x = np.arange(10, 30, 5)
print x

[10 15 20 25]
```

而 `numpy` 也可以讓我們很快速產生一個零陣列或都是一的陣列。另外 `arange` 則是一個可以很方便產生固定距離數字陣列的函式，不過產生出來的東西只有一維的，如果我們要讓他變成多維的話，可以使用下面這種函式 `reshape`。

array manipulation

```
In [21]: x = np.arange(15)
print x
x = x.reshape(3,5)
print x

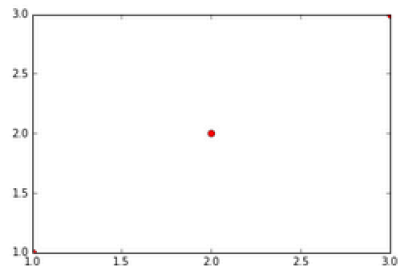
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

用 Python 畫圖

我們在算數學的過程，常常會需要將其視覺化，接下來我們要介紹一種常用的函式庫，請看以下。`matplotlib` 是我們常用來畫圖的函式庫，我們會使用裡面的 `pyplot` 的

```
In [22]: from matplotlib import pyplot as plt
%matplotlib inline
```

```
In [23]: plt.plot([1,2,3], [1,2,3], 'ro')
#toggle axis
plt.axis([0, 5, 0, 5])
plt.show()
```



接下來這部分則是矩陣視覺化，有以顏色表示也有用單純的灰階來表示。

```
In [5]: x = np.arange(100).reshape(10,10)
print x
```

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

```
In [6]: plt.matshow(x)
#toggle for gray colormap
plt.matshow(x, cmap=plt.cm.gray)
plt.show()
```



微積分與函數

剛剛有說明，numpy 跟 scipy 都是很好用的工具，當然也可以拿來算微積分等等數學。

在算數學時也常有把函數畫出圖來的動作，在 numpy 裡面也是做得到的喔！

對電腦來說，是沒有連續的東西，因此畫函數的第一步驟，是標出樣本點 (Sample)，這裡面所用的就是 meshgrid 函數。將自己需要的 x, y 座標點做出一個陣列，在丟進 meshgrid 函數，meshgrid 會輸出兩個函數，待會分別拿來當 x, y 座標的樣本點。接下來要就是畫圖的部分，我們要畫的是 $z = \sin(2x + 2y)/(2x + 2y)$ 。一樣我們先 meshgrid，接下來非常簡單，只要把丟進需要的函數就可以了，然後我們使用 contourf 來畫函數。

Calculus

```
In [7]: from scipy import integrate

In [8]: def f(x) :
         return x**2
         result, err = integrate.quad(f, 0, 1)
         print result
         print err
0.3333333333333333
3.70074341542e-15
```

numpy meshgrid

```
In [9]: import numpy as np

In [10]: nx, ny = (3,2)

In [11]: x = np.linspace(0, 1, nx)

In [12]: x
Out[12]: array([ 0. ,  0.5,  1. ])

In [13]: y = np.linspace(0, 1, ny)

In [14]: y
Out[14]: array([ 0.,  1.])

In [15]: xv, yv = np.meshgrid(x, y)

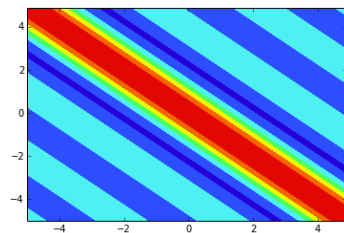
In [16]: xv
Out[16]: array([[ 0. ,  0.5,  1. ],
                [ 0. ,  0.5,  1.]])

In [17]: yv
Out[17]: array([[ 0.,  0.,  0.],
                [ 1.,  1.,  1.]])
```

example for function evaluation

```
In [18]: import numpy as np
         import matplotlib.pyplot as plt
         x = np.arange(-5, 5, 0.1)
         y = np.arange(-5, 5, 0.1)

In [19]: xx, yy = np.meshgrid(x, y, sparse = True)
         #print xx
         #print yy
         z = np.sin(xx**2+yy**2)/(xx**2+yy**2)
         h = plt.contourf(x, y, z)
         plt.show()
```



牛刀小試

基本的 Python 以及 Jupyter Notebook 教學就到這邊囉！這邊有兩道小題目給大家做練習。
第一題：用 for loop 做出一個九九乘法表。

1. Use for loop to Print a multiplication table (from 1~9)

```
In [45]: # It should look like this
# 1*1=1 1*2=2 . . .
# 2*1=2 2*2=4 . . .
```

Hint : You should use two layer of for loop.

```
In [46]: for i in range(1, 9):
          print i, ' * ', i, ' = ', i*i

1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
```

第二題：畫出 $z = \cos(x^2 + y^2)$ 函數。

2. Draw $z = \cos(x^2 + y^2)$

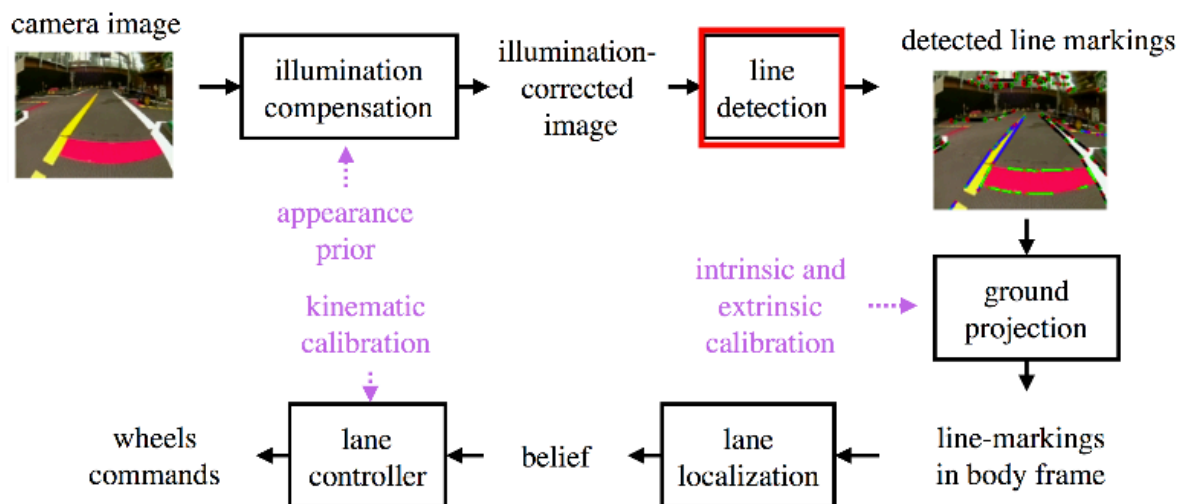
Hint : We have done draw $z = \sin(x^2 + y^2)/(x^2 + y^2)$, just modify the code!

```
In [ ]: #Step:
#1. import numpy & matplotlib.pyplot
#2. meshgrid the x, y
#3. use the result of meshgrid to draw
```

2 線段偵測

2.1 自動駕駛流程圖

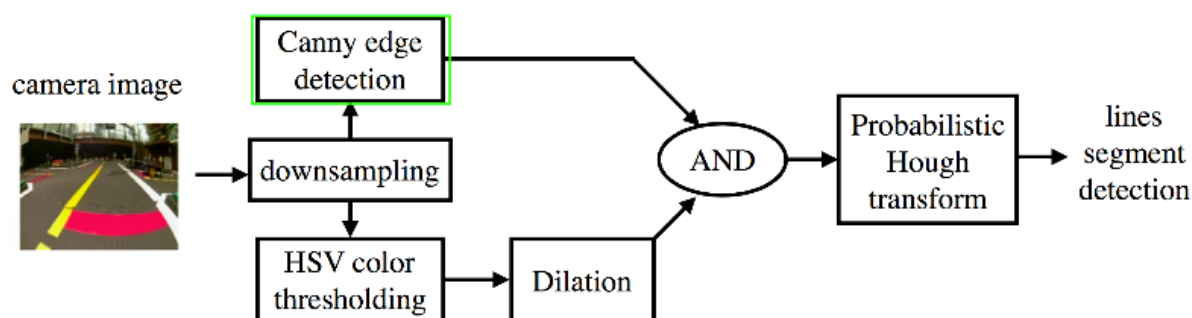
這裡我們要先解釋一下在 Duckietown 裡自動駕駛的流程圖。



在鏡頭得到圖片之後，這些圖會先利用程式調整光線對比，接著我們就會去偵測圖裡面的線段，也就是道路邊緣的線段。接著偵測出來的光線會利用相機校正後得到的參數，得到每個線段在真實世界中的座標位置。接著利用這些線段判斷道路的位置以及車子的位置(專業用語是 Pose” 姿態”)。最後再給馬達做相對應的動力。

2.2 線段偵測原理

接著我們來解釋線段偵測的原理。



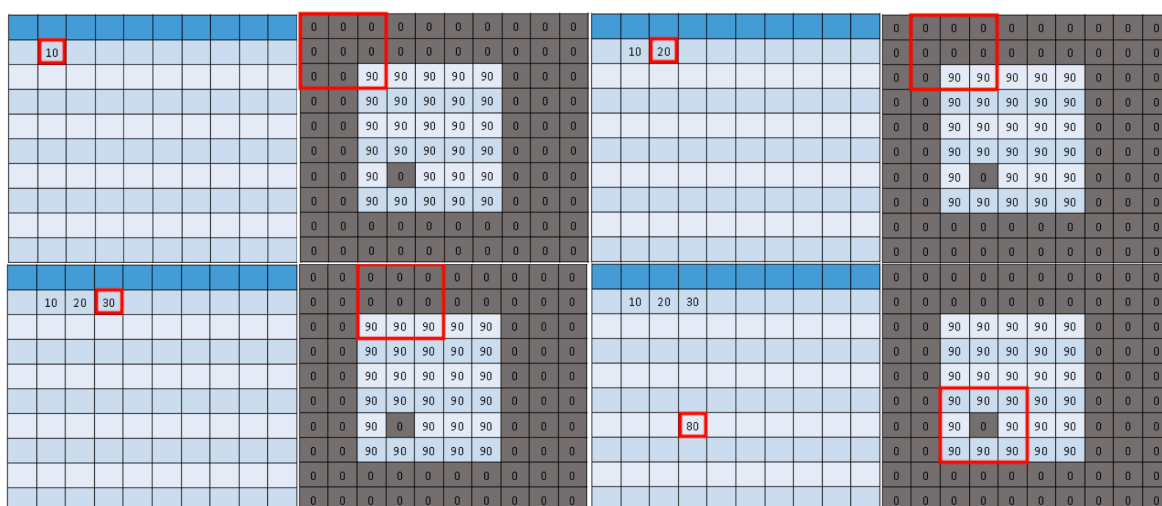
首先會先縮小圖片以增加運算速度，接著圖片會分別經過 Canny Edge Detector (Canny 邊緣運算子) 以及 HSV

Color Thresholding (HSV 顏色閾值判斷)。將兩邊所得的接過經過 AND 邏輯閘。最後再做 Hough Transform (霍夫轉換) 就可以得到結果了。

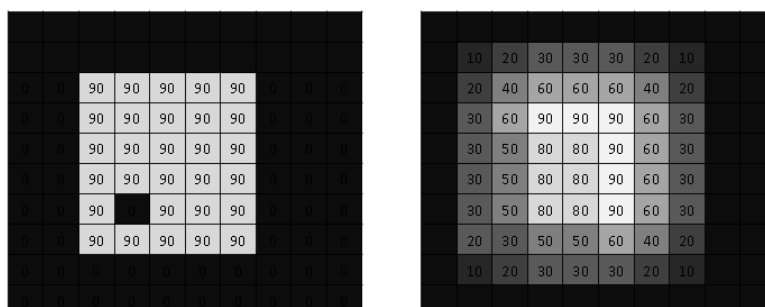
2.3 Canny Edge Detector (Canny 邊緣運算子)

二維卷積

在解釋 Canny Edge Detector 之前我們先解釋偵測線斷的原理。一般而言，我們都是用 Convolution (卷積) 來判斷線段。我們先看下面的例子。



先來解釋二維的卷積是怎麼運作的。基本上我們會把一個 Kernel(紅色框框部分) 與圖片中的像素點對齊，每個點乘以一之後，全部相加取平均，就是這個 Kernel 核心的那個像素點的值。接下來我們把 Kernel 不斷地向右移，就可以得到大部分像素點新的值。最後我們把最外圍的像素點補上 0(有時候是補上最大值，看使用的地方而定) 就大功告成了。



Sobel Operator & Canny Edge Detector

那麼二維卷積可以怎麼幫助我們呢？我們介紹一種 Sobel Operator (索貝爾算子)。Sobel Filter 基本上就是對圖片做二維卷積，但他的 Kernel 長這樣：

-1	0	1
-2	0	2
-1	0	1

Vertical

1	2	1
0	0	0
-1	-2	-1

Horizontal

左邊的 Vertical Kernel 可以偵測出垂直的線段，右邊的 Horizontal Kernel 可以偵測出水平的線段。待會我們會看實際的結果。

接下我們就來講解 Canny Edge Detector 又是怎麼做的。相對 Sobel Operator 他就複雜多了。首先我們的圖片會先移除雜訊，接著我們會找出圖片中的梯度，並且除去較不真實的結果，然後再取出有可能是邊緣的地方。雖然以上步驟看似困難，但我們都可以用 OpenCV 的函式輕易完成，我們先稍微看過程式碼跟結果。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('duckietown.jpg',0)
edges = cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
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()
```

Original Image



Edge Image



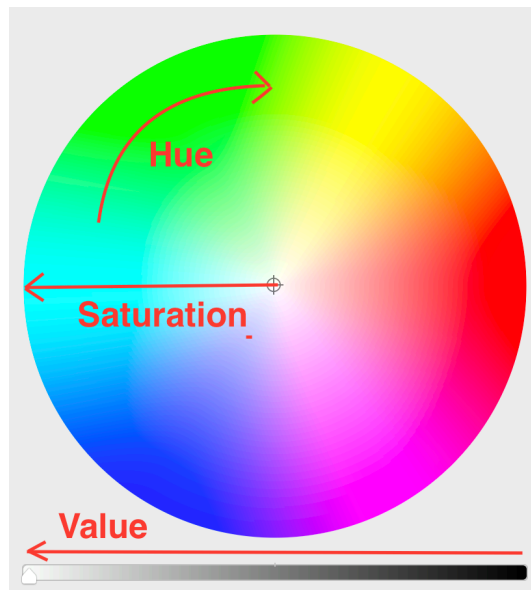
實際應用

我們把 Canny 邊緣運算子套在我們要運算的圖片上面就是這個樣子。可以看到圖中我們已經把裡面的邊緣都匡起來了，也可以找到許多道路線段的痕跡。



2.4 HSV Color Space & Thresholding (HSV 色彩空間與閾值判斷)

HSV Color Space (HSV 色彩空間) 在這裡我們先簡單介紹 HSV 色彩空間是什麼，我們可以用一個有趣的例子來介紹



這是蘋果電腦的調色盤，他就是使用 HSV 色彩空間調出各種顏色。HSV 色彩空間其實就是另一種表示顏色的方式，就像我們平常使用的 RGB。那我們可以看到圖中，”Hue”表示類似不同的色彩區域，”Saturation”則類似色彩飽和度，”Value”則類似色彩量的多寡。而每個值分別是 0 ~ 255。(除了”Hue”是 0 ~ 127) 至於為何要使用 HSV 色彩空間則是因為，在 HSV 空間裡差不多的顏色會是在連起來的區塊裡，較適合我們拿來做色彩的判斷。

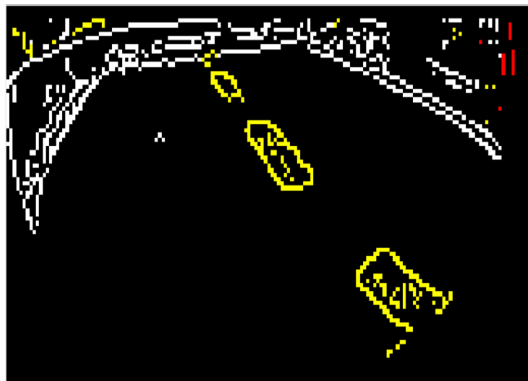
HSV Thresholding (HSV 閾質判斷) 如果我們要找出黃色和白色線段，就要找出圖中黃色和白色的區域。



當該像素點位於我們需要的 HSV 白色/黃色的空間內，我們就把它標記起來，這就是標記後的結果。就是所謂的 HSV Thresholding。

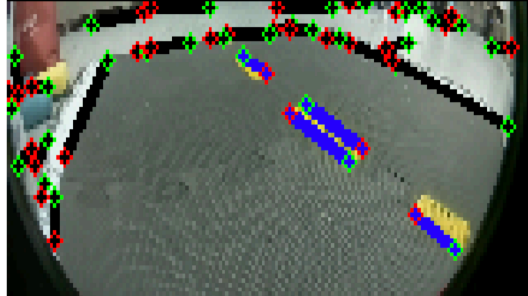
2.5 AND 邏輯閘和 Hough Transform (霍夫轉換)

最後我們將兩張圖用 AND 邏輯閘結合再一起



在裡面我們就可以分辨出道路黃線與白線的部分。接下來我們會使用 Hough Transform(霍夫轉換) 找出線段的部分，因為真正的到路線會以線段的方式出現，可以幫助我們去除明顯不是道路線段的部分，也是我們之後判斷車體在哪裡的一個基準。

下圖中黑色就是電腦判斷為白色線段的部分，藍色為電腦判斷為黃色線斷的部分。



2.6 小試身手！

接著我們就來看實際程式碼的部分囉！我們會將剛剛說的理論部分直接以程式碼實現！請大家用 Jupyter Notebook 開啟 4.2 的檔案。

縮小圖片與邊緣偵測

首先這部分就是簡單地縮小圖片

```
In [5]: import cv2
import numpy as np
from matplotlib import pyplot as plt

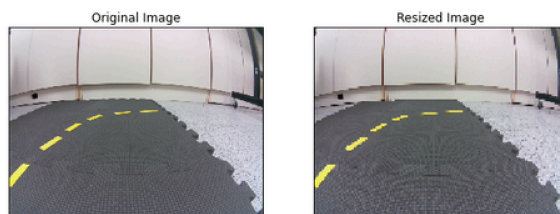
#Use your own image
img = cv2.imread("images/curve-right.jpg")

image_cv = cv2.resize(img, (160, 120), interpolation=cv2.INTER_NEAREST)

dst1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
#dst1 = img #if don't change color
plt.subplot(121), plt.imshow(dst1, cmap = 'bgr')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])

dst2 = cv2.cvtColor(image_cv, cv2.COLOR_BGR2RGB)
plt.subplot(122), plt.imshow(dst2, cmap = 'bgr')
plt.title('Resized Image'), plt.xticks([]), plt.yticks([])

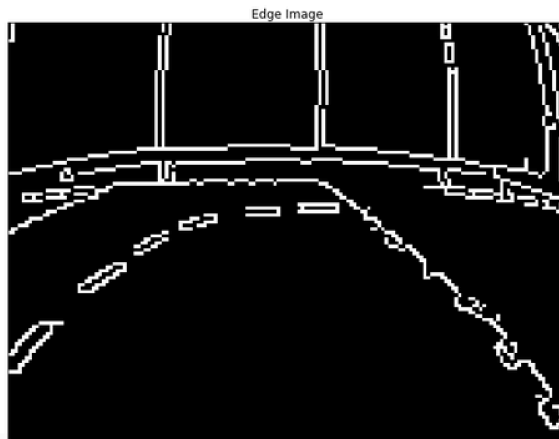
plt.show()
```



以及找到圖片中的邊緣

這些部分雖然原理複雜，但用函式庫可以很快就完成喔！大家也可以試試 Canny 邊緣運算子的兩個參數，看看哪些參數的效果最好。

```
In [6]: gray = cv2.cvtColor(image_cv,cv2.COLOR_BGR2GRAY)
edges=cv2.Canny(gray,100,350)
plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```



HSV Thresholding (HSV 閾質判斷)

接下來我們要做 HSV 閾質判斷，也就是哪邊是黃色、哪邊是白色以及紅色。在判斷之前我們要先定義”什麼是黃色、紅色、白色“

```
In [8]: hsv_white1 = np.array([0,0,150])
hsv_white2 = np.array([180,100,255])
hsv_yellow1 = np.array([25,50,50])
hsv_yellow2 = np.array([45,255,255])
hsv_red1 = np.array([0,100,100])
hsv_red2 = np.array([15,255,255])
hsv_red3 = np.array([165,100,100])
hsv_red4 = np.array([180,255,255])
```

大家可以看到，當我的 Hue 落在 0 到 180、Saturation 落在 0 到 100、Value 落在 150 到 255 時，就是白色！其他以此類推。接下來是判斷以及判斷結果展示

```

In [9]: hsv = cv2.cvtColor(image_cv,cv2.COLOR_BGR2HSV)
        #print hsv
        white = cv2.inRange(hsv,hsv_white1,hsv_white2)
        #print white
        yellow = cv2.inRange(hsv,hsv_yellow1,hsv_yellow2)
        red1 = cv2.inRange(hsv,hsv_red1,hsv_red2)
        red2 = cv2.inRange(hsv,hsv_red3,hsv_red4)
        red = cv2.bitwise_or(red1,red2)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3, 3))
        white = cv2.dilate(white, kernel)
        yellow = cv2.dilate(yellow, kernel)
        red = cv2.dilate(red, kernel)
        # Uncomment '#' to plot with color
        x = cv2.cvtColor(yellow, cv2.COLOR_GRAY2BGR)
        x[:, :, 2] *= 1
        x[:, :, 1] *= 1
        x[:, :, 0] *= 0
        x = cv2.cvtColor(x, cv2.COLOR_BGR2RGB)

        y = cv2.cvtColor(red, cv2.COLOR_GRAY2BGR)
        y[:, :, 2] *= 1
        y[:, :, 1] *= 0
        y[:, :, 0] *= 0
        y = cv2.cvtColor(y, cv2.COLOR_BGR2RGB)

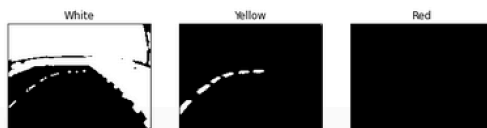
        plt.subplot(131),plt.imshow(white,cmap = 'gray')
        plt.title('White'), plt.xticks([], plt.yticks([]))

        plt.subplot(132),plt.imshow(yellow,cmap = 'gray')
        #plt.subplot(132),plt.imshow(x,cmap = 'bgr')
        plt.title('Yellow'), plt.xticks([], plt.yticks([]))

        plt.subplot(133),plt.imshow(red,cmap = 'gray')
        plt.subplot(133),plt.imshow(y,cmap = 'bgr')
        plt.title('Red'), plt.xticks([], plt.yticks([]))

        plt.show()

```



結合邊緣偵測及 HSV 閾質判斷
接下來就要做我們所謂 AND 開的部分

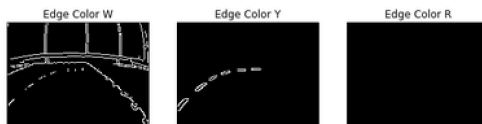
```

In [10]: edge_color_white=cv2.bitwise_and(edges,white)
        edge_color_yellow=cv2.bitwise_and(edges,yellow)
        edge_color_red=cv2.bitwise_and(edges,red)

        plt.imshow(edge_color_yellow,cmap = 'gray')
        plt.title('Edge Color Y'), plt.xticks([], plt.yticks([]))
        plt.subplot(131),plt.imshow(edge_color_white,cmap = 'gray')
        plt.title('Edge Color W'), plt.xticks([], plt.yticks([]))
        plt.subplot(132),plt.imshow(edge_color_yellow,cmap = 'gray')
        plt.title('Edge Color Y'), plt.xticks([], plt.yticks([]))
        plt.subplot(133),plt.imshow(edge_color_red,cmap = 'gray')
        plt.title('Edge Color R'), plt.xticks([], plt.yticks([]))

        plt.show()

```



找出線段
最後就是利用 Hough Transform 找出線段

```

In [16]: lines_white = cv2.HoughLinesP(edge_color_white,1,np.pi/180,10,np.empty(1),1.5,1)
lines_yellow = cv2.HoughLinesP(edge_color_yellow,1,np.pi/180,10,np.empty(1),1.5,1)
lines_red = cv2.HoughLinesP(edge_color_red,1,np.pi/180,10,np.empty(1),1.5,1)

color = "yellow"
lines = lines_yellow
bw = yellow

color = "white"
lines = lines_white
bw = white

#color = "red"
#lines = lines_red
#bw = red

if lines is not None:
    lines = np.array(lines[0])
    print "found lines"

else:
    lines = []
    print "no lines"

found lines

```

索貝爾算子 (Sobel Operator) 我們來看一下，雖然沒有用到但是也很有趣的索貝爾算子

```

In [8]: import cv2
import numpy as np
from matplotlib import pyplot as plt

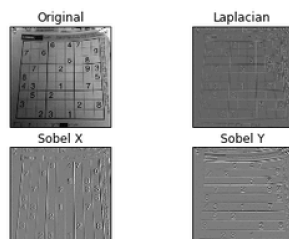
img = cv2.imread('images/sudoku.jpg',0)

laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([], plt.yticks([]))
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([], plt.yticks([]))
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([], plt.yticks([]))
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([], plt.yticks([]))

plt.show()

```



我們可以看到，水平與垂直的索貝爾算子可以分別得到垂直與水平的邊緣。

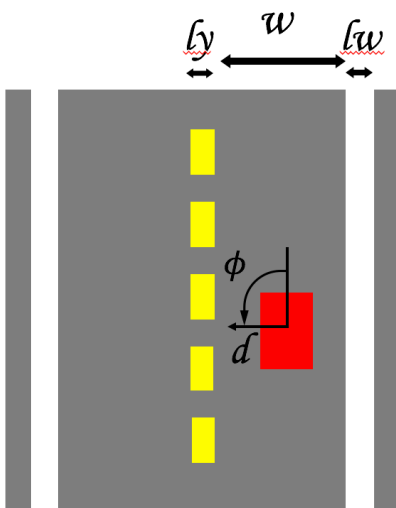
3 Lane Filter & 車體控制

3.1 Lane Filter

Lane Filter 較難找到一個適合的中文翻譯，不過基本上 Lane Filter 在做的就是判斷車體在道路中相對的位置。我們接下來就來解釋他怎麼運作的吧！

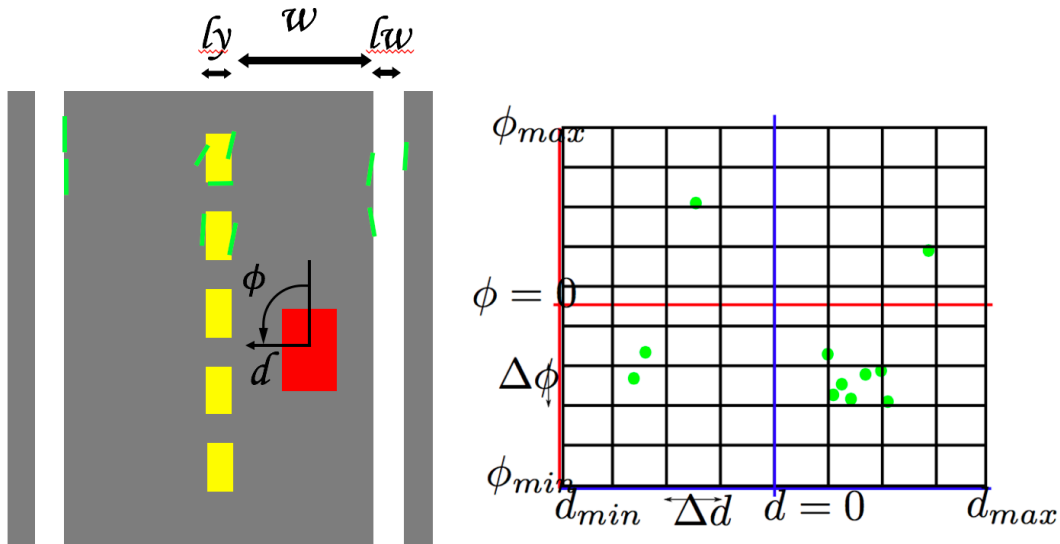
這裡我們要分成四個部分來解釋，首先想知道車體在道路中的相對位置，就要知道我們怎麼表示車子的位置。接下來我們會利用一種“投票”的機制來決定車子的位置。最後我們會用貝式濾波器來讓產生這個預測與執行的控制迴圈。(貝式率波器將不會出現在本書，有興趣可上交大 OCW 觀看)

State (車子狀態) State(狀態) 可以用來描述一系列機器人與環境可以對未來造成影響的部分。聽起來很複雜，但我們這裡先告訴大家描述車體的位置的方式。請看下圖



我們可以看到，這裡使用兩個變數描述車體的位置。 d 表示車體與道路中線的距離，而 ϕ 則表示該位置與垂直的相對角度。

Measure Model (量測模型) 接下來我們要做的就是剛剛有提到的“投票”機制，我們會利用上一章所提到的，產生的線段。利用這些線段進行投票，以描述出車體與道路間的相對位置。我們會把各線段利用剛剛的表示方法拿來做計算，也就是會有這樣的一個轉換
將這些得到的線段位置拿去做一個“投票”的動作，就可以決定出車子的位置。



3.2 Car Command (車體控制)

最後我們若得到車子的相對位置，就可以對他進行相對應的控制，利用一個 `car_command` 的函數來發出一個 topic。

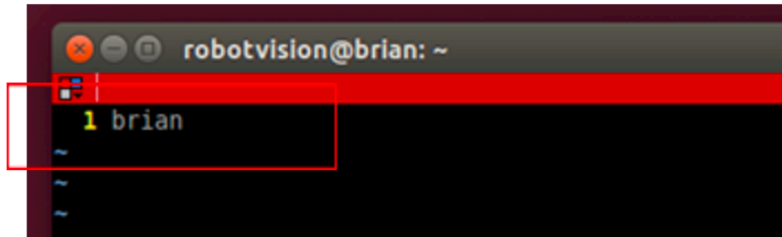
```
def car_command(v, omega, duration):
    # Send stop command
    car_control_msg = Twist2DStamped()
    car_control_msg.v = v
    car_control_msg.omega = omega
    pub_car_cmd.publish(car_control_msg)
    rospy.sleep(duration)
    #rospy.loginfo("Shutdown")
    car_control_msg.v = 0.0
    car_control_msg.omega = 0.0
    pub_car_cmd.publish(car_control_msg)
```

3.3 小試身手！

說了這麼多，我們不如直接來試試看吧！在我們開始之前有一些部分要進行設定，為了等等的車體控制的部分。

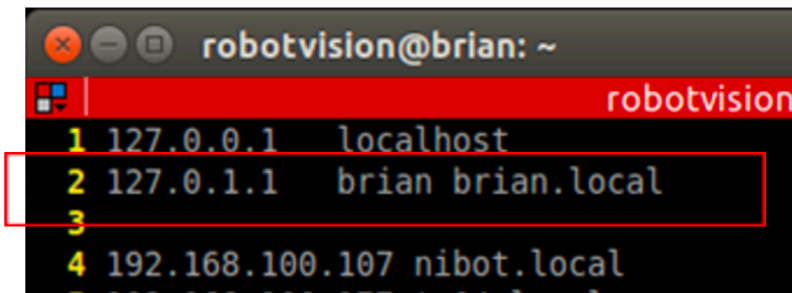
首先我們需要跟改筆電這裡的 hostname，請更改 robotvision 至你想要的 hostname

\$ duckietop: sudo vim /etc/hostname



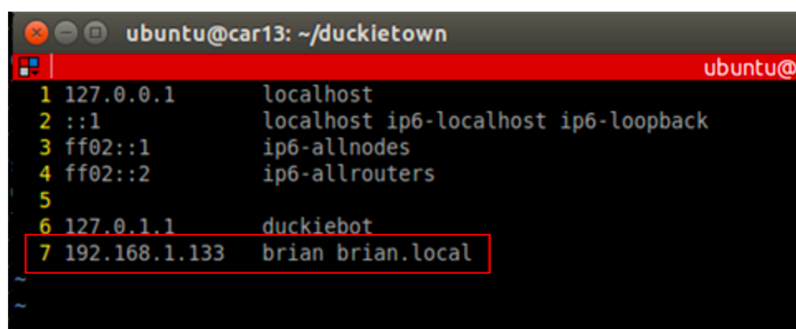
```
robotvision@brian: ~  
1 brian
```

增加” 127.0.1.1 hostname hostname.local”
\$ duckietop: sudo vim /etc/hosts



```
robotvision@brian: ~  
1 127.0.0.1 localhost  
2 127.0.1.1 brian brian.local  
3  
4 192.168.100.107 nibot.local
```

連線進車子，將筆電的 IP 位址加進去
\$ duckiebot: sudo vim /etc/hosts



```
ubuntu@car13: ~/duckietown  
1 127.0.0.1 localhost  
2 ::1 localhost ip6-localhost ip6-loopback  
3 ff02::1 ip6-allnodes  
4 ff02::2 ip6-allrouters  
5  
6 127.0.1.1 duckiebot  
7 192.168.1.133 brian brian.local
```

接下來我們需要把車子上的 launch 打開，已接受筆電傳的訊息。記得將 car13 改成你的車子的名稱
\$ duckiebot: cd ~/duckietown
\$ duckiebot: source environment.sh
\$ duckiebot: source set_ros_master.sh
\$ duckiebot: roslaunch duckietown_nctu_wama joystick_jupyter.launch veh:=car13

最後我們要開啟 Jupyter Notebook 來控制車子。

```
$ duckietop: source ~/duckietown/catkin_ws/devel/setup.bash
```

```
$ duckietop: source ~/duckietown/set_ros_master.sh car13
```

```
$ duckietop: cd ~/duckietown/tutorials/summer2017_nctu
```

```
$ duckietop: jupyter notebook 4-3.lane_filter_drive.ipynb
```