APPUNTI DI ROBOTICA – Elaborazione di Immagini e Visione

(con esempi in Python v.2.7.6)

Prof. Fischetti Pietro

Definizioni (vedi [1]):

■*sistema*, nel suo significato più generico, è un insieme di elementi interconnessi tra di loro ed eventualmente con l'esterno tramite reciproche relazioni, ma che si comporta come un tutt'uno, secondo proprie regole.

■*risposta di un sistema*il risultato y di una trasformazione matematica T applicata ad un segnale x(t), in sintesi:

y(t)=T[x(t)] nel caso continuo e y(n)=T[x(n)] nel caso discreto.

■*sistema a tempo continuo* o semplicemente *sistema continuo* un sistema in cui sia gli ingressi che le uscite sono funzioni a tempo continuo.

■*sistema a tempo discreto* o semplicemente *sistema discreto* un sistema in cui sia gli ingressi che le uscite sono funzioni a tempo discreto.

■*sistema discreto lineare* un sistema in cui l'uscita e' combinazione lineare dei campioni del segnale di ingresso ed eventualmente dei campioni del segnale di uscita, in pratica se e solo se:

$y(n)=T[ax_1(n)+b\,x_2(n)]=T[ax_1(n)]+T[bx_2(n)]=aT[x_1(n)]+bT[x_2(n)]=ay_1(n)+by_2(n)$

■*sistema tempo-invariante* un sistema in cui se y(n) e' la risposta a x(n) allora y(n-k) e' la risposta a x(n-k), in questo caso il sitema puo' venire studiato a partire dal tempo zero essendo invariante alla traslazione nel tempo.

■*sistema causale* se l'uscita ad un certo istante $n=n_0$dipende dall'ingresso solo per $n\leq n_0$. Un sistema lineare tempo-invariante (SLTI) e' causale se la risposta al campione unitario e' zero per n<0. Cioe' l'uscita non anticipa l'ingresso.

■*sistema stabile* se un ingresso limitato in ampiezza produce un segnale in uscita limitato in ampiezza.

■*Segnale campionato* un segnale a tempo discreto e quantizzato nelle ampiezze in cui cioe' i valori possibili appartengono ad un intervallo finito.

Esempio: verificare che il sistema: T[x(n)]=a*x(n)+b (a,b costanti) risulta non lineare, causale, tempo-invariante e stabile.

CONVOLUZIONE 1D

La Convoluzione e' un'operazione (Integrale nel caso continuo Somma nel caso discreto) che lega la risposta di un sistema ad un segnale di ingresso.

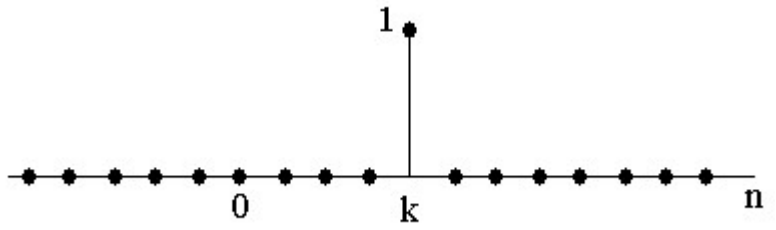
Cominciamo con il definire'campione unitario' la sequenza:

$$\delta(n)=\begin{cases}0, & n\neq 0\\ 1, & n=0\end{cases}$$

Rappresentato in figura:



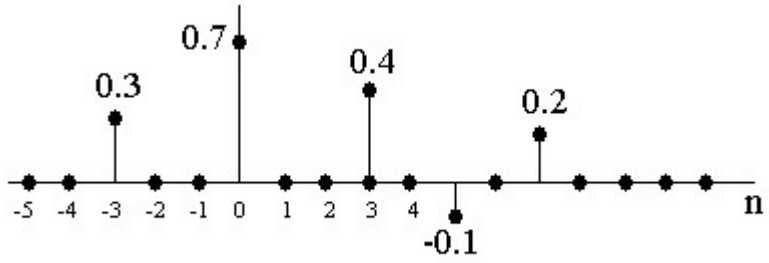
Lo stesso segnale traslato di k campioni cioe' $\delta(n-k)$ si rappresentacome in figura:



Quindi in generale un segnale discreto x si puo' scrivere come una sequenza infinita di impulsi:

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k)$$

Ad esempio il seguente segnale:



Si puo' scrivere come:

$$x(n) = 0.3\delta(n+3) + 0.7\delta(n) + 0.4\delta(n+3) - 0.1\delta(n-5) + 0.2\delta(n-7)$$

La risposta(T[]) di un sistema generico si puo' scrivere come:

y(n)=T[x(n)]=T$[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)]$

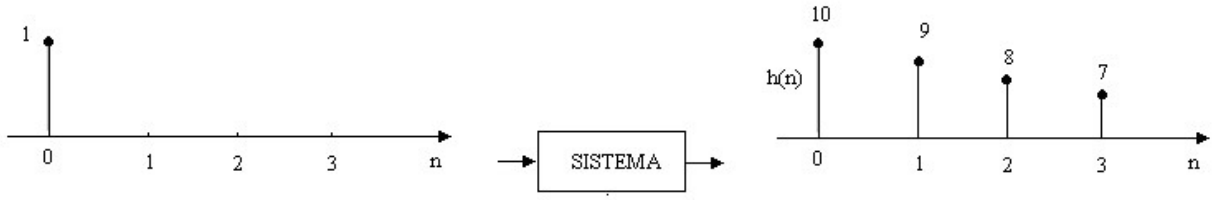
Se il sistema e' lineare allora la relazione precedente si puo' riscrivere come:

y(n)=T$[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)] = \sum_{k=-\infty}^{\infty} x(k)T[\delta(n-k)]$

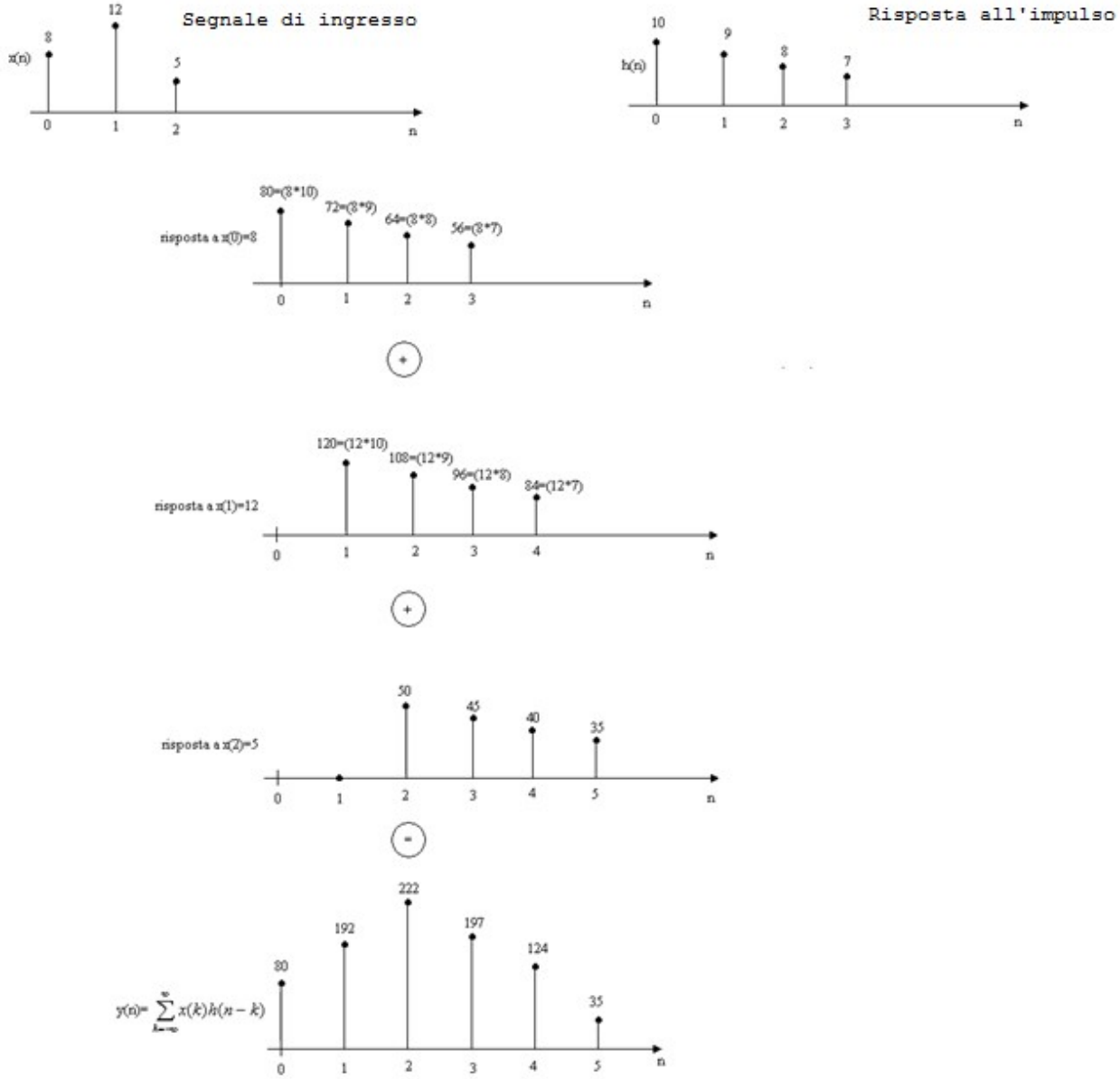
e se il sistema lineare e' anche tempo invariante alla traslazione otteniamo in definitiva quella che si chiama 'somma di convoluzione':

y(n)= $\sum_{k=-\infty}^{\infty} x(k)h(n-k)$ (Sistema Lineare Tempo Invariante - SLTI)

Vediamo graficamente in dettaglio l'operazione di convoluzione, ad esempio data la risposta all'impulso delseguente sistema:



Calcoliamo la risposta ad un ingresso qualsiasi x(n):



Se N e' la lunghezza del segnale d'ingresso e M la lunghezza della risposta all'impulso la risposta sara' lunga M+N-1 punti. Risulta chiara la semplificazione introdotta dai SLTI, in quanto conoscendo la sola risposta all'impulso del sistema, possiamo ottenere la risposta del sistema ad un segnale qualsiasi.

CONV1D.PY - Convoluzione di segnali monodimensionali

```
import sys
option_f = False
option_p = False
hFileName = ""

def main(argv):
    if len(argv)<2:
        print '*** Convolution 1D ***\nusage: conv1D -h<hfileTxt>< inFileTxt'
        sys.exit(-1)

    for i in range(0,len(argv)):
      if argv[i] == "-f":
        option_f = True
      elif argv[i][0] == '-' and argv[i][1] == "h":
        hFileName = argv[i][2:]

    h = list()
    hFile = open(hFileName, 'r')
    for line in hFile:
        h.append(float(line))

    hFile.close()

    x = []
    y=0
    i=0
    try:
        while (True):
            x.append(float(raw_input()))
    except EOFError:
        pass
    M=len(x)+len(h)-1;
    y = [0] * M

    for n in range(0,len(x)):
      for k in range(n,n+len(h)):
          y[k]=y[k] + x[n]*h[k-n]

    for i in range(0,len(y)):
        print y[i]

if __name__ == "__main__":
    try:
        main(sys.argv)
    except Exception as e:
        print e.__doc__
        print e.message
```

Comando:

```
C:\>Conv1D.py -hh.txt < x.txt
```

File: x.txt	**File: h.txt**	**Output:**
8	10	80.0
12	9	192.0
5	8	222.0
	7	197.0
		124.0
		35.0

Stesso esempio utilizzando numpy

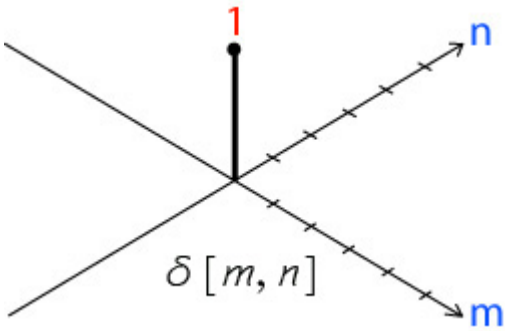
```
Import numpy as np
x = [8, 12, 5]
```

h = [10 ,9, 8, 7] print(np.convolve(x, h))
[80 192 222 197 124 35]

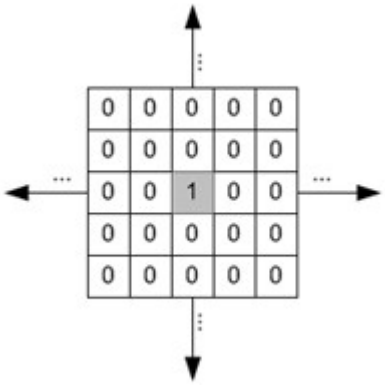
CONVOLUZIONE 2D

La Convoluzione in 2D e' una naturale estensione della convoluzione in 1D in due direzioni nello spazio: orizzontale e verticale [2]. La convoluzione 2D e' spesso utilizzata nell'elaborazione di immagini, ad esempio nell'estrazione dei contorni.

L'impulso (delta) nello spazio si indica con la funzione δ[m, n] che vale 1 solo se m=n=0 e zero altrimenti, e puo' essere rappresentata nella figura seguente (sinistra), e in forma matriciale come nella figura seguente (destra).



Impulso 2D – rappresentazione grafica



Matrice della funzione impulso 2D

Quindi estendendo i ragionamenti fatti per il caso 1D, un segnale 2D (un'immagine) puo' essere rappresentato come somma di funzioni impulso opportunamente amplificati e spostati.

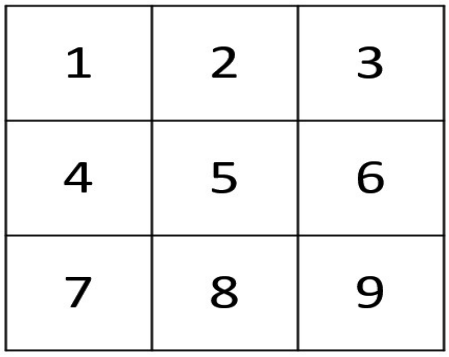
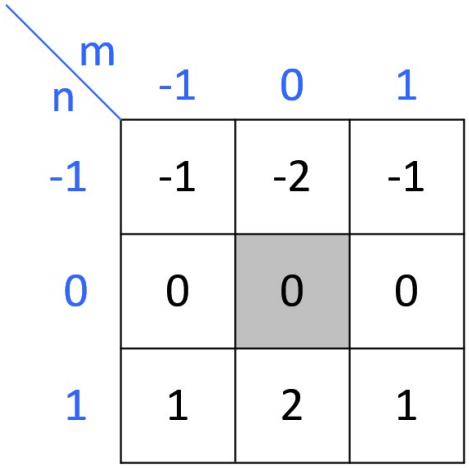
$$x[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot \delta[m-i, n-j]$$

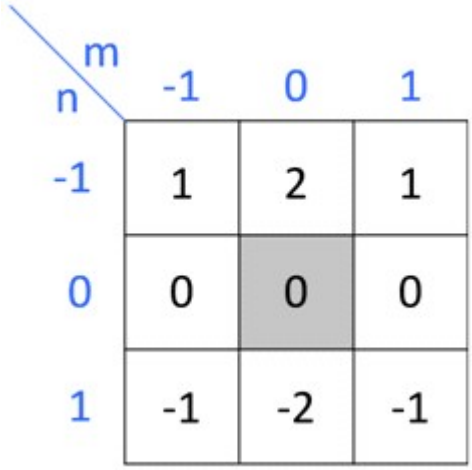
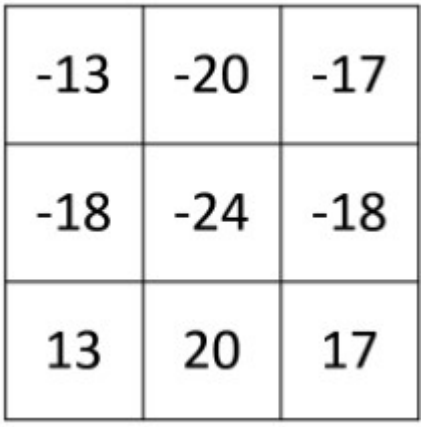
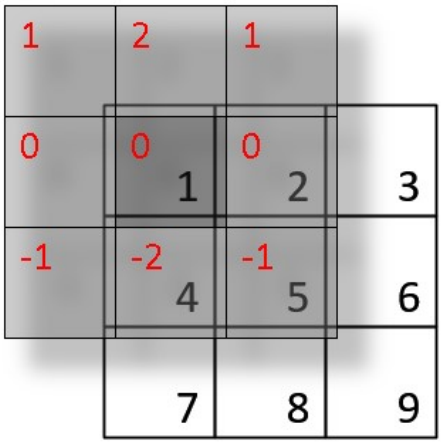
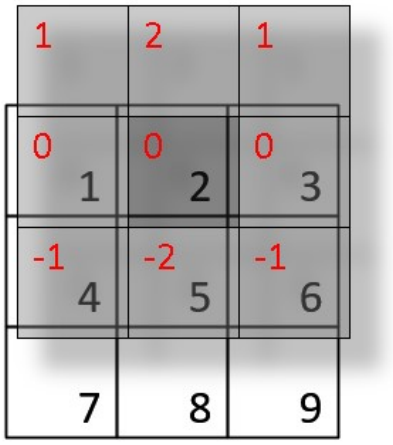
Per esempio, $x[0, 0] = x[0, 0]\cdot\delta[m, n]$, $x[1, 2] = x[1, 2]\cdot\delta[m-1, n-2]$, ….
Quindi l'uscita di un sistema TLI in 2D puo' essere scritta come convoluzione del segnale di ingresso $x[m, n]$ e della risposta all'impulso $h[m, n]$:

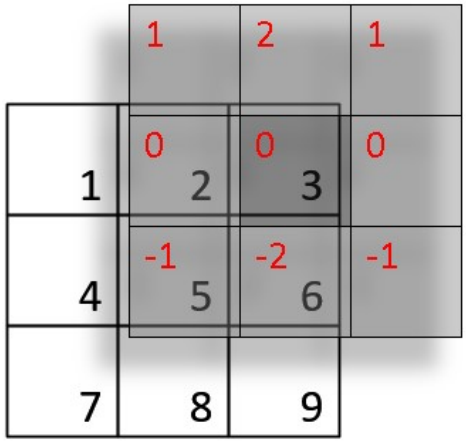
$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$

Notiamo che il centro della risposta all'impulso e' dato da h[0,0]
La risposta all'impulso in 2D viene chiamata kernel nell'elaborazione di immagini.

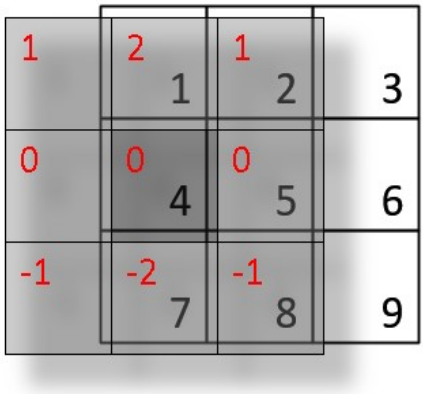
ESEMPIO [3] – Convoluzione 2D in dettaglio come scorrimento del kernel sull'immagine:

	
Input	**Kernel** (n.b. il kernel viene ribaltato in quanto nella relazione precedente c'e': $h[m-i, n-j]$)

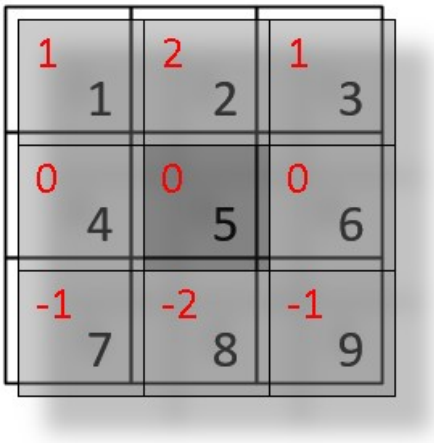
	
Kernel ribaltato (riga 1 scambiata con la riga 3, xolonna 1 scambiata con la 3	**Risultato**
	
y[0,0]=0*1+0*2+0*1+0*0+1*0+2*0+0*(-1)+4*(-2)+5*(-1)=-**13**	**y[1,0]=-20**



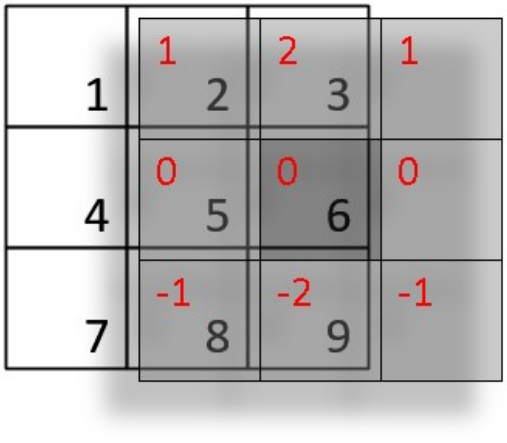
y[2,0]=-17



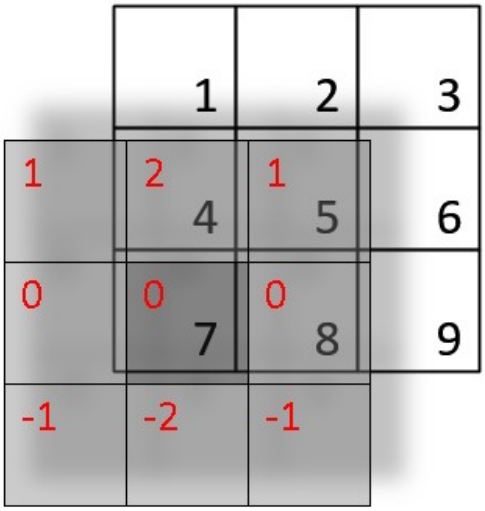
y[0,1]=-18



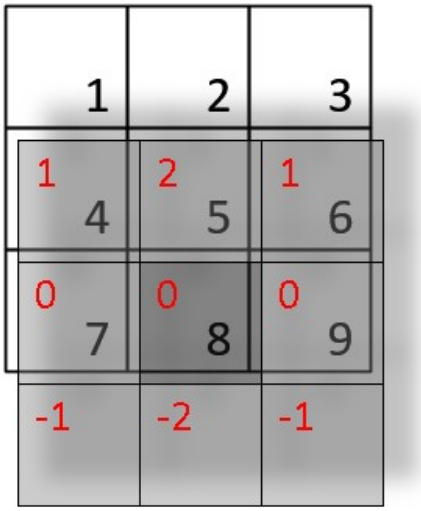
y[1,1]=-24



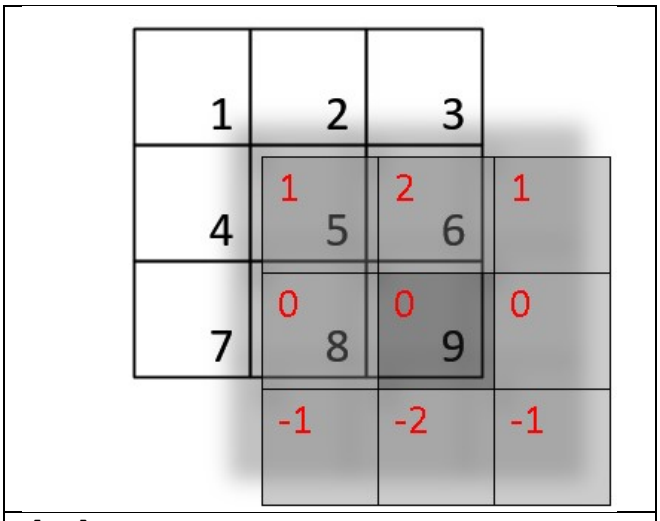
y[2,1]=-18



y[0,2]=13



y[1,2]=20



y[2,2]=17

Risultato finale la Convoluzione 2D:

-13	-20	-17
-18	-24	-18
13	20	17

Output

CONV2D.PY–CONVOLUZIONE 2D

```
def MatrixCreate(NR,NC,val=0):
   return [[val for col in range(NC)] for row in range(NR)]
def MatrixIsEmpty(M):
   if(M==None):
      return True
   return len(M)==1 and len(M[0])==0
def MatrixSize(M):
   if(MatrixIsEmpty(M)):
      return 0,0
   return len(M),len(M[0])
def printMatrix(M):
   print '\n'.join([str(i) for i in M])

def MatrixConvolve2D(M,kernel):
   rows,cols=MatrixSize(M)
```

```
    kRows,kCols=MatrixSize(kernel)
    out = MatrixCreate(rows,cols)

    kCenterX = kCols / 2;
    kCenterY = kRows / 2;

    for i in range(rows):
        for j in range(cols):
            z = MatrixCreate(rows,cols)
            v=M[i][j]
            z[i][j] = v
            for ik in  range(kRows):
                for jk in  range(kCols):
                    m = ik-kCenterX+i
                    n = jk-kCenterY+j
                    if not(m<0 or n<0 or m>=kRows or n>=kCols):
                        z[m][n] = v*kernel[ik][jk]

            for a in range(rows):
                for b in range(cols):
                    out[a][b] += z[a][b]
    return out


inp=[
        [1,2,3],
        [4,5,6],
        [7,8,9]
        ]
kernel =[
        [-1, -2, -1],
        [0,  0, 0],
        [1, 2, 1]
    ]
out = MatrixConvolve2D(inp,kernel)
printMatrix( out)
```

```
[-13, -20, -17]
[-18, -24, -18]
[13, 20, 17]
```

Stesso esempio utilizzando scipy

```
Import scipy
w=np.array([[1,2,3],[4,5,6],[7,8,9]])
k=np.array([[-1,-2,-1],[0,0,0],[1,2,1]])

print(scipy.signal.convolve2d(w,k, mode='same'))
```

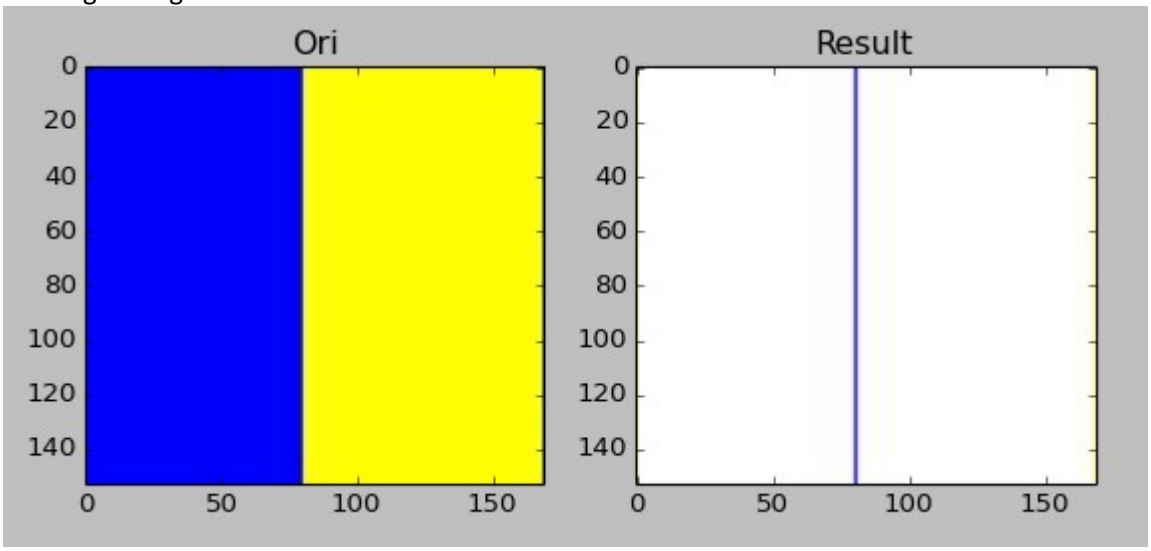
```
[[-13 -20 -17]
 [-18 -24 -18]
 [ 13  20  17]]
```

IMMAGINI-FILTRI DIGITALI 2D

Semplificando al massimo, potremmo evidenziare i controni di un'immagine utilizzando una maschera con valori positivi e negativi es:

kernel=[[-1,1]]

Infatti nel caso in cui ci si trovi in una zona dello stesso colore il risultato della convoluzione sarebbe nullo, mentre in corrispondenza della transizione tra i due colori si otterrebbe un valore diverso da zero come nella figura seguente:



In [4]vengono analizzati (in C/ C++) alcuni dei filtri 2D piu' usati nell'elaborazione di immagini (Blur, Motion, Emboss, Edges, Sharpen, Mean).Di seguito un esempio in Python di estrazione di contorni da un'immagine reale.

FILT2D.PY - Estrazione di contorni da un file immagine.

```
from PIL import Image,ImageOps
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def MatrixSize(M):
   return len(M),len(M[0])

def MatrixCreate(NR,NC,val=0):
   return [[val for col in range(NC)] for row in range(NR)]

def printMatrix(M):
   print '\n'.join([str(i) for i in M])

def MatrixConvolve(M,K,factor=1.0,bias=0.0):
   w,h=MatrixSize(M)
   filterWidth,filterHeight=MatrixSize(K)
   result = np.copy(M)
   for x in range(w):
      for y in range(h):
         red = 0.0
         green = 0.0
         blue = 0.0

         for filterX in range(filterWidth):
```

```
            for filterY in range(filterHeight):
                imageX = (x - filterWidth / 2 + filterX + w) % w
                imageY = (y - filterHeight / 2 + filterY + h) % h
                color = M[imageX][imageY]
                red += color[0] * K[filterX][filterY]
                green += color[1] * K[filterX][filterY]
                blue += color[2] * K[filterX][filterY]

        r = min(max(int(factor * red + bias), 0), 255)
        g = min(max(int(factor * green + bias), 0), 255)
        b = min(max(int(factor * blue + bias), 0), 255)
        result[x,y] = r,g,b
    return result

def MatrixInvertColor(M):
    w,h=MatrixSize(M)
    result = np.copy(M)
    for x in range(w):
        for y in range(h):
            red = 0.0
            green = 0.0
            blue = 0.0
            color = M[x][y]
            r=255-color[0]
            g=255-color[1]
            b=255-color[2]
            result[x,y] = r,g,b
    return result

def LoadImg(sFile):
    im = Image.open(sFile)
    return np.array(im)
    pass

def MatrixToImg(M):
    return Image.fromarray(M)

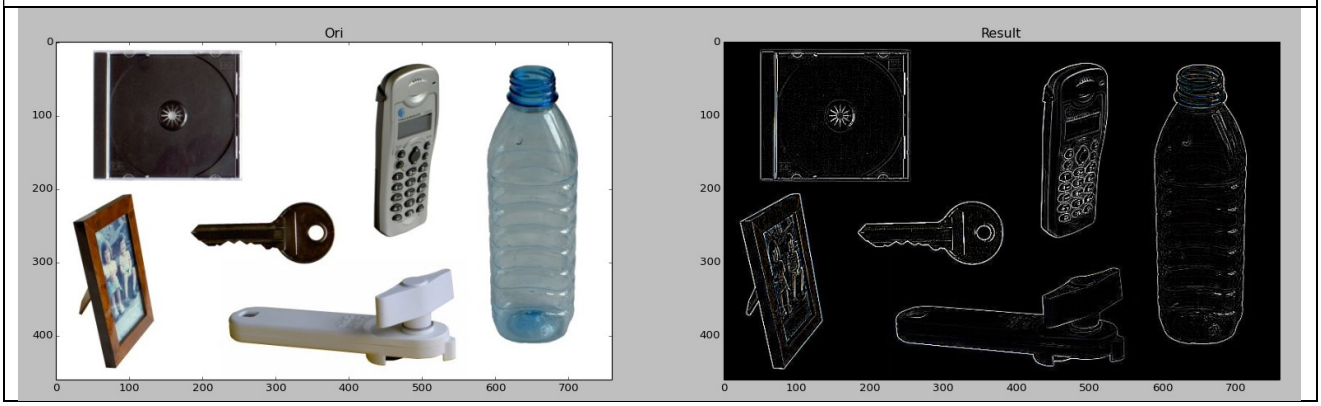
def Draw(img,n,sTitle=None):
    plt.subplot(n+120)
    if sTitle != None:
        plt.title(sTitle)
    imgplot = plt.imshow(img)
    return imgplot

def Show():
    plt.show()

def main():
kernel =[
                [-1, -1, -1],
                [-1,  8, -1],
                [-1, -1, -1]
```

```
            ]
    img = LoadImg("Image1.jpg")
    cv = MatrixConvolve(img,kernel)
    cvi=MatrixToImg(cv)
    Draw(img,1,"Ori")
    Draw(cvi,2,"Result")
    Show()

if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        print e.__doc__
        print e.message
```



NAO – RICONOSCIMENTO DELLA PALLA ROSSA[5]

In [5] vengono applicati alcuni filtri 2D su un'immagine presa dlla telecamera di NAO per determinare la posizione e il raggio di un pallina rossa. L'immagine viene dapprima trasformata da RGB a HSV per facilitare il riconoscimento dato che lo spazio HSV e' meno influenzato dall'ambiente. Successivamente l'immagine HSV viene filtrata tramite una semplice soglia sul colore rosso, successivamnte viene pulita dal rumore tramite filtro Gaussiano, successivamente vengono rimossi eventuali residui di nero all'interno dell'immagine, a questo punto viene applicato un filtro per l'estrazione dei controni (Canny), ed infine tramite la trasformata di Hough viene determinato il centro della pallina.

NAO_RedBall.py – Programma in Python per il Riconoscimento della Palla Rossa

```
import cv2
import numpy as np

img = cv2.imread('redball.png')
RED_MIN = np.array([0, 100, 100],np.uint8)
RED_MAX = np.array([60, 255, 255],np.uint8)

hsv_img = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
frame_threshed = cv2.inRange(hsv_img, RED_MIN, RED_MAX)
cv2.imshow('ori 0',img)

cv2.imshow('HSVresult 1',frame_threshed)

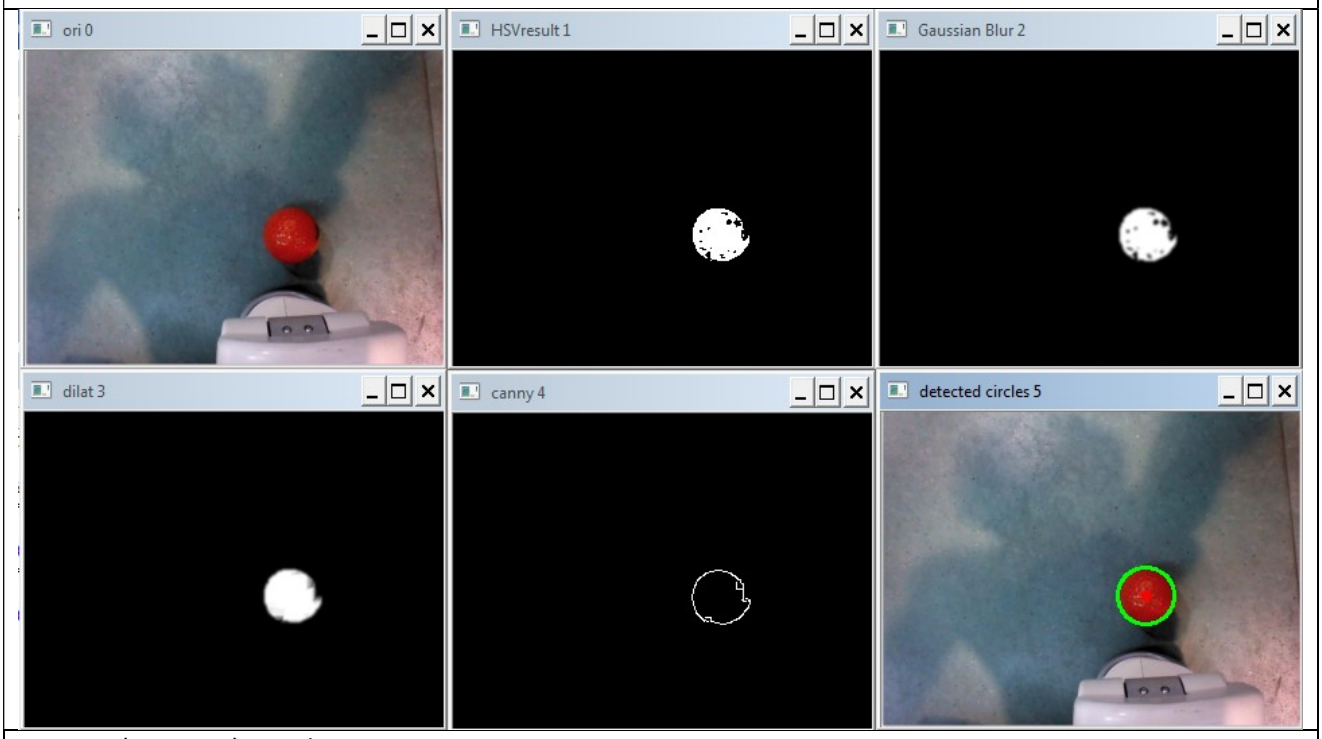
blur = cv2.GaussianBlur(frame_threshed,(5,5),0)
```

```
cv2.imshow('Gaussian Blur 2',blur)

kernel = np.ones((5,5),np.uint8)
dilation = cv2.dilate(blur,kernel,iterations = 3)
blur = cv2.erode(dilation,kernel,iterations = 3)
cv2.imshow('dilat 3',blur)
edges = cv2.Canny(blur,50,300)
cv2.imshow('canny 4',edges)

circles=cv2.HoughCircles(edges,cv2.cv.CV_HOUGH_GRADIENT,1,25,param1=55,param2=25,minRadius=10,
maxRadius=0)
circles = np.uint16(np.around(circles))
for i in circles[0,:]:
   center=(i[0],i[1])
   radius = i[2]
   # draw the outer circle
cv2.circle( img, center, radius, (0,255,0),2)
# draw the center of the circle
cv2.circle( img, center, 2, (0,0,255),3)
   print "center=",center,"    radius=",radius

cv2.imshow('detected circles 5',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Center= (196, 136) radius= 21

UTILIZZO DELLA LIBRERIA OPENCV

Il codice seguente grazie alla Libreria SimpleCV permette di riconoscere la presenza di un volto ripreso da telecamera e di determinarne le coordinate del rettangolo che la contiene

FaceReco.py - RICONOSCIMENTO DI VOLTI

```
from SimpleCV import Camera, Display
```

```
cam = Camera(0)
disp = Display(cam.getImage().size())
while disp.isNotDone():
    img = cam.getImage()
    faces = img.findHaarFeatures('face')
    if faces:
        for face in faces:
            print "Face at: " + str(face.coordinates())
    else:
        print "No faces detected."
    if faces is not None:
        faces = faces.sortArea()
        bigFace = faces[-1]
        bigFace.draw()
    img.save(disp)
```

BIBLIOGRAFIA

[1]A.V.Oppenheim-R.W Shafer, *Elaborazione Numerica dei Segnali*,Franco Angeli
[2] http://www.songho.ca/dsp/convolution/convolution.html
[3] http://www.songho.ca/dsp/convolution/convolution2d_example.html
[4] http://lodev.org/cgtutor/filtering.html
[5] Han Lin-BEHAVIOR DESIGN OF NAO HUMANOID ROBOT: A CASE OF PICKING UP A BALL AND THROWING INTO A BOX - http://www.theseus.fi/handle/10024/92571