In [21]:

```
%load_ext autoreload
%autoreload 2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import time
import imagetools.assignment3 as im
from imagetools.provided import *
%matplotlib notebook
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Separable Convolutions

In [16]:

```
def shift(x, k, l, boundary):

    n1, n2 = x.shape[:2]

    if boundary is 'periodical':
        xshifted = np.zeros(x.shape)
        irange = np.mod(np.arange(n1) + k, n1)
        jrange = np.mod(np.arange(n2) + l, n2)
        xshifted = x[irange, :][:, jrange]
        return xshifted
    k = -k
    l = -l
    if boundary is 'extension':
        xshifted = np.zeros(x.shape)
        irange = np.pad(np.arange(max(0,-k),min(n1-k,n1)),(max(k,0),max(0,-k)), 'constant', constant
_values=(0, n1-1))
        jrange = np.pad(np.arange(max(0,-l),min(n2-l,n2)),(max(l,0),max(0,-l)), 'constant', constant
_values=(0, n2-1))
        xshifted = x[irange, :][:, jrange]
        return xshifted
    if boundary is 'mirror':
        xshifted = np.zeros(x.shape)
        irange = np.pad(np.arange(max(0,-k),min(n1-k,n1)),(max(k,0),max(0,-k)), 'reflect')
        jrange = np.pad(np.arange(max(0,-l),min(n2-l,n2)),(max(l,0),max(0,-l)), 'reflect')
        xshifted = x[irange, :][:, jrange]
    return xshifted
```

Question 1

In [106]:

```
def kernel1(name,tau):
    eps=1e-3
    if name is 'gaussian':
        i =0
        while (np.exp(-(i**2)/(2*tau**2)) >eps):
            i+=1

        nx = i-1
        ny =i-1
        #points in the range
        i = np.arange(-nx, nx+1, 1)
        j = np.arange(-ny, ny+1, 1)

        #create a grid and evaluate the kernel atevery i,j
        xx, yy = np.meshgrid(i, j, indexing='ij')
        f_ij = np.exp(-(xx**2+yy**2 )/(2*tau**2))
```

```
    if name is 'exponential':
        i =0
        while (np.exp(-np.sqrt((i**2))/(tau)) >eps):
            i+=1

        nx = i-1
        ny =i-1
        #points in the range
        i = np.arange(-nx, nx+1, 1)
        j = np.arange(-ny, ny+1, 1)
        #create a grid and evaluate the kernel atevery i,j
        xx, yy = np.meshgrid(i, j, indexing='ij')
        f_ij = np.exp(np.sqrt(xx**2 + yy**2)/-tau)


    if name is 'box':
        f_ij =np.ones((tau,tau))

    Z =np.sum(f_ij)
    nu = (1/Z)*f_ij
    return nu

def kernel(name, tau):

    eps=1e-3
    if name is 'grad1_forward':
        nu = np.zeros((3, 1))
        nu[1, 0] = -1
        nu[2, 0] = 1
    if name is 'grad2_forward':
        nu = np.zeros((1, 3))
        nu[0, 1] = -1
        nu[0, 2] = 1
    if name is 'grad1_backward':
        nu = np.zeros((3, 1))
        nu[0, 0] = -1
        nu[1, 0] = 1
    if name is 'grad2_backward':
        nu = np.zeros((1, 3))
        nu[0, 0] = -1
        nu[0, 1] = 1
    if name is 'laplacian1':
        nu = np.zeros((3, 1))
        nu[0, 0] = 1
        nu[1, 0] = -2
        nu[2, 0] = 1
        return nu
    if name is 'laplacian2':
        nu = np.zeros((1, 3))
        nu[0, 0] = 1
        nu[0, 1] = -2
        nu[0, 2] = 1
        return nu

    if name.startswith('gauss'):
        i =0
        while (np.exp(-(i**2)/(2*tau**2)) >eps):
            i+=1

        ny =i-1
        #points in the range
        j = np.arange(-ny, ny+1, 1)

        f_ij = np.exp(-(j**2)/(2*tau**2))

    if name.startswith('exp'):
        i =0
        while (np.exp(-np.sqrt((i**2))/(tau)) >eps):
            i+=1

        ny =i-1
        #points in the range
        j = np.arange(-ny, ny+1, 1)

        f_ij = np.exp(np.sqrt(j**2)/-tau)
```

```
    if name.startswith('box'):
        f_ij =np.ones((tau))


    if name.endswith('1'):
        sh = f_ij.shape[0]
        f_ij = f_ij.reshape(sh,1)
        Z =np.sum(f_ij)
        nu = (1/Z)*f_ij

    if name.endswith('2'):
        sh = f_ij.shape[0]
        f_ij = f_ij.reshape(1,sh)
        Z =np.sum(f_ij)
        nu = (1/Z)*f_ij

    return nu
```

Question 2

In [257]:

```
def convolve(x, nu, boundary='periodical', separable=None):


    if separable is None:
        n1, n2 = x.shape[:2]
        s1 = int((nu.shape[0] - 1) / 2)
        s2 = int((nu.shape[1] - 1) / 2)
        xconv = np.zeros(x.shape)

        for k in range(-s1, s1+1):
            for l in range(-s2, s2+1):
                #shift the image
                xshift = shift(x, -k,-l,boundary)

                xshift = shift(x, -k,-l,boundary)
                xconv += (nu[k+s1,l+s2]*xshift)

    if separable is 'product':
        n1, n2 = x.shape[:2]
        s = nu[0].shape[0]
        nuu = nu
        for i in range(2):
            nu = nuu[i]
            s1 = int((nu.shape[0] - 1) / 2)
            s2 = int((nu.shape[1] - 1) / 2)
            xconv = np.zeros(x.shape)

            for k in range(-s1, s1+1):
                for l in range(-s2, s2+1):
                    #shift the image


                    xshift = shift(x, -k,-l,boundary)
                    xconv += (nu[k+s1,l+s2]*xshift)
            x = xconv

    if separable is 'sum':
        n1, n2 = x.shape[:2]
        s = nu[0].shape[0]
        nuu = nu
        xconv = np.zeros(x.shape)
        for i in range(2):
            nu = nuu[i]
            s1 = int((nu.shape[0] - 1) / 2)
            s2 = int((nu.shape[1] - 1) / 2)


            for k in range(-s1, s1+1):
                for l in range(-s2, s2+1):
                    #shift the image


                    xshift = shift(x, -k,-l,boundary)
```

```
                    xshift = shift(x, -k,-l,boundary)
                    xconv += (nu[k+s1,l+s2]*xshift)



    return xconv
```

Question 3

In [164]:

```
tau =4
x = plt.imread('assets/train.png')
nu1   = kernel('gaussian1', tau)
nu2   = kernel('gaussian2', tau)
nu    = ( nu1, nu2 )
t1 = time.time()
xconv = convolve(x, nu, boundary='mirror', separable='product')
t2 = time.time()
tsep = t2-t1

nu    = kernel1('gaussian', tau)
t1 = time.time()
xconv0 = convolve1(x, nu, boundary='mirror')
t2 = time.time()
tnsep = t2-t1

print('Test results are similar?',np.allclose(xconv,xconv0))

import imagetools as im
fig, axes = plt.subplots(ncols=3, figsize=(7, 2))
im.show(x, ax=axes[0])
axes[0].set_title('Original')
im.show(xconv0, ax=axes[1])
axes[1].set_title(" Non-Separable takes {:.2f} s ".format(tnsep))
im.show(xconv, ax=axes[2])
axes[2].set_title(" Separable takes {:.2f} s ".format(tsep))
fig.show()
```

```
Test results are similar? True
```

2 Derivative Filters

Question 4

In [110]:

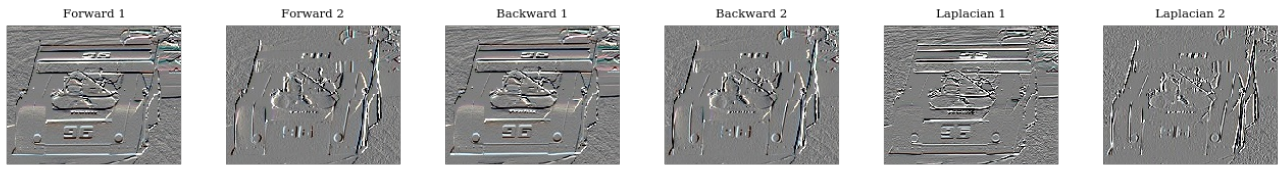
```
from scipy import ndimage, misc
y = plt.imread('assets/race.png')
#forward 1
nu_fwd_1 = kernel('grad1_forward',0)
y_fwd_1 = convolve(y, nu_fwd_1, boundary='mirror')
#forward 2
nu_fwd_2 = kernel('grad2_forward',0)
y_fwd_2 = convolve(y, nu_fwd_2, boundary='mirror')
#backward 1
```

```
nu_bkd_1 = kernel('grad1_backward',0)
y_bkd_1 = convolve(y, nu_bkd_1, boundary='mirror')
#backward 2
nu_bkd_2 = kernel('grad2_backward',0)
y_bkd_2 = convolve(y, nu_bkd_2, boundary='mirror')
#laplacian1
nu_lp_1 = kernel('laplacian1',0)
y_lp_1 = convolve(y, nu_lp_1, boundary='mirror')
#laplacian2
nu_lp_2 = kernel('laplacian2',0)
y_lp_2 = convolve(y, nu_lp_2, boundary='mirror')
print(y_bkd_1.shape)
```

```
(321, 481, 3)
```

In [144]:

```
import imagetools as im
fig, axes = plt.subplots(ncols=6, figsize=(13, 2))
im.show(y_fwd_1[80:240,150:350,:], ax=axes[0], vmin=-0.2, vmax=0.2)
axes[0].set_title('Forward 1')
im.show(y_fwd_2[80:240,150:350,:], ax=axes[1],vmin=-0.2, vmax=0.2)
axes[1].set_title('Forward 2')
im.show(y_bkd_1[80:240,150:350,:], ax=axes[2],vmin=-0.2, vmax=0.2)
axes[2].set_title('Backward 1')
im.show(y_bkd_2[80:240,150:350,:], ax=axes[3],vmin=-0.2, vmax=0.2)
axes[3].set_title('Backward 2')
im.show(y_lp_1[80:240,150:350,:], ax=axes[4],vmin=-0.2, vmax=0.2)
axes[4].set_title('Laplacian 1')
im.show(y_lp_2[80:240,150:350,:], ax=axes[5],vmin=-0.2, vmax=0.2)
axes[5].set_title('Laplacian 2')
fig.show()
```



Question 6 and 8

The backward difference matrix is: $\begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$

And its transpose is: $\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 1 \end{bmatrix}$

that's the negative of the forward difference matrix as required

In [190]:

```
#proof on 4-d signal
sig_4 = (np.array([1, 2,9,5])).reshape(4,1)


s1 = sig_4.shape[0]
s2 = nu_fwd_1.shape[0]

nu_fwd_1mat = np.array([[-1, 1,0,0],[0,-1, 1,0],[0,0,-1,1],[1,0,0,-1]])
res_1 = convolve(sig_4, nu_fwd_1, boundary='periodical')
```

```
res_2 = np.dot(nu_fwd_1mat,np.flip(sig_4,axis=0))

nu_bkd_1mat = np.array([[1, 0,0,-1],[-1,1, 0,0],[0,-1,1,0],[0,0,-1,1]])
res_1 = convolve(sig_4, nu_bkd_1, boundary='periodical')
res_2 = np.dot(nu_bkd_1mat,np.flip(sig_4,axis=0))


print(res_1,res_2)
print(nu_fwd_1mat)
```

```
[[-1.]
 [-7.]
 [ 4.]
 [ 4.]] [[ 4]
 [ 4]
 [-7]
 [-1]]
[[-1  1  0  0]
 [ 0 -1  1  0]
 [ 0  0 -1  1]
 [ 1  0  0 -1]]
```

Question 7 and 8

From the analysis below we observe that the two operations are equal, therefore the linear operations are adjoint

In [202]:

```
x = plt.imread('assets/train.png')
y = plt.imread('assets/race.png')
lhs_1 = convolve(y, nu_bkd_1, boundary='periodical')
lhs = np.sum(lhs_1*x)

rhs_1 = convolve(x, nu_fwd_1, boundary='periodical')
rhs = np.sum(rhs_1*y)

print("Are the two equal?",np.isclose(lhs,-rhs))
```

```
Are the two equal? True
```

In [212]:

```
x_4 = (np.array([1, 2,9,5])).reshape(4,1)
y_4 = (np.array([2, 4,9,8])).reshape(4,1)

lhs_1 = np.dot(nu_bkd_1mat,np.flip(y_4,axis=0))
lhs = np.sum((np.flip(lhs_1,axis=0))*x_4)


rhs_1 = np.dot(nu_fwd_1mat,np.flip(x_4,axis=0))
rhs = np.sum((np.flip(rhs_1,axis=0))*y_4)


print("Are the two equal?",np.isclose(lhs,-rhs))
```

```
Are the two equal? True
```

Question 9

It is true only for the periodical condition

In [223]:

```
boundary = ['extension','mirror','periodical']
for b in boundary:
    conv_1 = convolve(y, nu_fwd_1, b)
    conv_2 = convolve(conv_1, nu_bkd_1, b)
    conv_3 = convolve(y, nu_lp_1, b)
    print("Are the two equal?",np.allclose(conv_2,conv_3), b)
```

```
Are the two equal? False extension
Are the two equal? False mirror
Are the two equal? True periodical
```

Question 10

In [231]:

```
conv1 = np.dot(nu_fwd_1mat,np.flip(y_4,axis=0))
conv2 = np.dot(nu_bkd_1mat,conv1)
conv3 = convolve(y_4, nu_lp_1, 'periodical')

print("Are the two equal?",np.allclose(np.flip(conv2,axis=0),conv3), b)
```

```
Are the two equal? True periodical
```

Question 11

In [236]:

```
def laplacian(x, boundary='periodical'):
    nu1 = kernel('laplacian1',0)
    nu2 = kernel('laplacian2',0)
    nu = (nu1,nu2)
    lap =convolve(x, nu, boundary, separable='sum')
    return lap
```

Question 12

In [249]:

```
def grad(x, boundary='periodical'):
    y =x
    #forward 1
    nu_fwd_1 = kernel('grad1_forward',0)
    y_fwd_1 = convolve(y, nu_fwd_1, boundary)
    #forward 2
    nu_fwd_2 = kernel('grad2_forward',0)
    y_fwd_2 = convolve(y, nu_fwd_2, boundary)

    g = np.stack([y_fwd_1,y_fwd_2],axis=-1)
    return g
```

Question 13

In [253]:

```
def div(f, boundary='periodical'):
    d = np.sum(f, axis=-1)
    return d
```

Question 14

In [267]:

```
lhs = div(grad(x,'periodical'),'periodical')
rhs = laplacian(x,'periodical')
print("Are the two equal?",np.allclose(rhs,lhs), b)
```

```
Are the two equal? False periodical
```

In [268]:

```
grad_x = grad(x,'periodical')
grad_y = grad(y,'periodical')
lap_x = laplacian(x,'periodical')

lhs = np.sum(grad_x*grad_y)
rhs = np.sum(lap_x*y)
print("Are the two equal?",np.allclose(lhs,-rhs), b)
```

```
Are the two equal? True periodical
```