

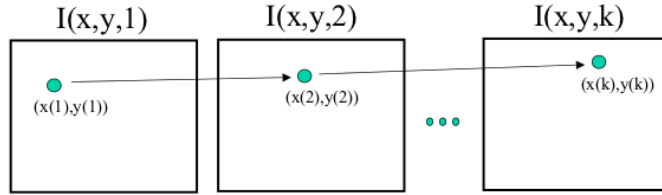
Piksel Takibi, Optik Akis, Lucas Kanade Algoritmasi

Hareket halindeki bir kameranın aldigi görüntülerdeki herhangi bir pikseli nasıl takip ederiz?

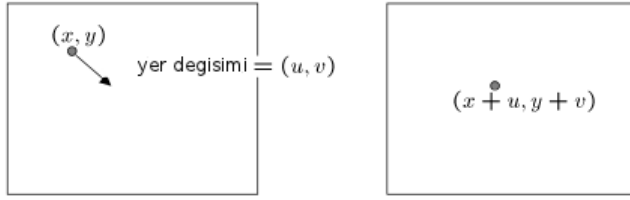
Matematiksel olarak temsil etmek gerekirse, zamana göre değişen 2 boyutlu görüntüyü bir fonksiyon olarak düşünelim, ki bu fonksiyonun değerleri ayrık-sal olarak, imajın ta kendisi. Bir $I(x(t), y(t), t)$ fonksiyonu piksel değerlerini veriyor. Bu fonksiyonda x, y ekran koordinatlarına tekabül ediyor, t ise zaman, 1, 2, ... gibi değerleri var, mesela $I(100, 200, 1)$, bize 1. fotoğraftaki $x = 100, y = 200$ koordinatlarındaki piksel değerini verecek.

x, y değişkenleri parametrize edildi, bir noktayı takip etmek istiyoruz çünkü, ve t 'ye göre bu takip edilen noktanın x, y koordinatları belli bir hızla değişiyor.

O zaman şu faraziye yapılarak problemimizi kolaylaştırabiliriz. Diyelim ki takip edilen bir nokta, görüldüğü her karede aynı piksel rengindedir. Bu çok sıradışı bir faraziye değil, resim karelerinden bir araba geçiyor, ve bu arabanın üzerindeki piksellerin renkleri, en azından iki kare arasında değişmiyor. Işık seviyesi, golgede olma, vs. gibi durumlarda biraz değişebilir, fakat basitleştirme amacıyla bu faraziye işe yarar.



Bir diğer faraziye, kameralar hareket ettiklerinde alınan iki görüntü arasındaki tüm piksellerin yer değişimi genellikle aynı yönde olmasıdır. Bu değişim yönünü $\langle u, v \rangle$ vektörü olarak görebiliriz, ve bu değişkenler iki görüntü arasındaki değişimde tüm pikseller için aynı olacaktır. Kameraları sağa hareket ettiriyoruz, ve görüntüdeki tüm pikseller sola doğru gidiyorlar.



Tum bunlari modelimizde nasil kullaniriz?

Takip edilen nokta her karede ayni renkte ise, su ifade dogru demektir

$$I(x(t), y(t), t) = \text{sabit}$$

Eger bu fonksiyonun zamana gore turevini alirsak

$$\frac{d I(x(t), y(t), t)}{dt} = 0$$

sonucu gelir. Esitligin sagi sifir, cunku bir sabitin turevini aldik. Sol tarafa Zincirleme Kanununu uygularsak,

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

Bu formilde dx/dt ve dy/dt , hareket halindeki (zaman gecerken) noktanin sonsuz kucuklukteki ne kadar yer degimi. Ayriksal baglamda arka arkaya iki kare icindeki yer degisimi. O zaman,

$$\frac{dx}{dt}, \frac{dy}{dt} = u, v$$

Alttakiler ise mesafesel (spatial) gradyanlardir, bunlari nasil hesaplanacagini cok iyi biliyoruz!

$$\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$$

Alttaki ise resim karelerinin zamana gore turevi

$$\frac{\partial I}{\partial t}$$

Daha derli toplu olarak gostermek gerekirse ana formül nihai olarak soyle

$$I_x u + I_y v + I_t = 0$$

ya da

$$\nabla I \cdot \langle u, v \rangle = -I_t$$

Peki bizim sectigimiz (takip edecegimiz) bir piksel icin u, v 'yi nasil hesaplayacagiz? Ustteki formulu hem takip ettigimiz, hem de onun etrafındaki pikseller icin yazarsak, ve bu sistemi cozersek, sonuca varabiliriz. Boylece iki tane bilinmeyenimiz olacak, ama pek cok formül gecek. Veriler gurultulu oldugu icin, aslında bilinmeyenden “daha fazla” formül elde etmek iyi, boylece elimizdeki denklem sistemi cok esitlige sahip (overdetermined) bir hale gelecek, ve bu tur sistemleri En Az Kareler (Least Squares) yontemi ile cozmeyi biliyoruz.

```
import numpy as np
import scipy.signal as si
from PIL import Image

def gauss_kern():
    h1 = 15
    h2 = 15
    x, y = np.mgrid[0:h2, 0:h1]
    x = x-h2/2
    y = y-h1/2
    sigma = 1.5
    g = np.exp( -( x**2 + y**2 ) / (2*sigma**2) );
    return g / g.sum()

def deriv(im1, im2):
    g = gauss_kern()
    Img_smooth = si.convolve(im1, g, mode='same')
    fx, fy = np.gradient(Img_smooth)
    ft = si.convolve2d(im1, 0.25 * np.ones((2,2))) + \
        si.convolve2d(im2, -0.25 * np.ones((2,2)))

    fx = fx[0:fx.shape[0]-1, 0:fx.shape[1]-1]
    fy = fy[0:fy.shape[0]-1, 0:fy.shape[1]-1];
    ft = ft[0:ft.shape[0]-1, 0:ft.shape[1]-1];
    return fx, fy, ft
```

```

if __name__ == "__main__":
    im1 = np.asarray(Image.open('flow1-bw-0.png'))
    im2 = np.asarray(Image.open("flow2-bw-0.png"))
    fx, fy, ft = deriv(im1, im2)

import matplotlib.pyplot as plt
import numpy as np
import scipy.signal as si
from PIL import Image
import deriv
import numpy.linalg as lin

def lk(im1, im2, i, j, window_size) :
    fx, fy, ft = deriv.deriv(im1, im2)
    halfWindow = np.floor(window_size/2)
    curFx = fx[i-halfWindow-1:i+halfWindow,
               j-halfWindow-1:j+halfWindow]
    curFy = fy[i-halfWindow-1:i+halfWindow,
               j-halfWindow-1:j+halfWindow]
    curFt = ft[i-halfWindow-1:i+halfWindow,
               j-halfWindow-1:j+halfWindow]
    curFx = curFx.T
    curFy = curFy.T
    curFt = curFt.T

    curFx = curFx.flatten(order='F')
    curFy = curFy.flatten(order='F')
    curFt = -curFt.flatten(order='F')

    A = np.vstack((curFx, curFy)).T
    U = np.dot(np.dot(lin.pinv(np.dot(A.T,A)),A.T),curFt)
    return U[0], U[1]

if __name__ == "__main__":
    x=165
    y=95
    win=50
    im1 = np.asarray(Image.open('flow1-bw-0.png'))

```

```

print im1.shape
#im2 = np.asarray(Image.open('flow2-bw-0.png'))
im2 = np.asarray(Image.open('upright.png'))
#im2 = np.asarray(Image.open('dleft.png'))
print im2.shape
u, v = lk(im1, im2, x, y, win)
print u, v
plt.imshow(im1, cmap='gray')
plt.hold(True)
plt.plot(x,y, '+r');
plt.plot(x+u*3,y+v*3, 'og')
plt.show()

```