

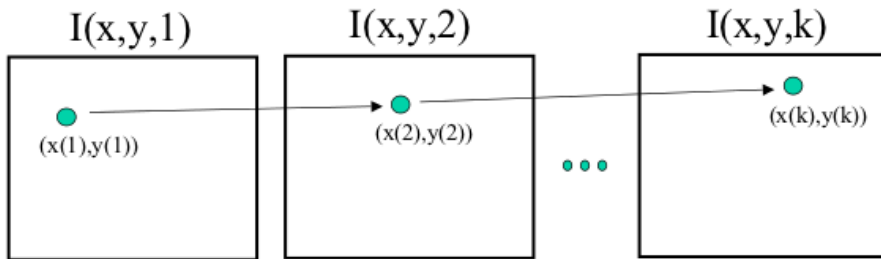
Piksel Takibi, Optik Akis, Lucas Kanade Algoritması

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

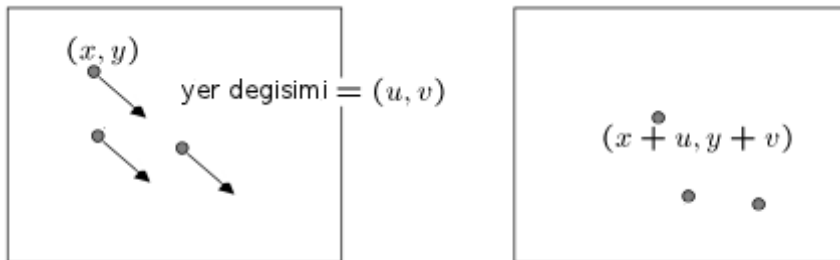
Matematiksel olarak temsil etmek gerekirse, zamana göre değişen 2 boyutlu görüntüyü bir fonksiyon olarak düşünelim, ki bu fonksiyonun değerleri ayrıksal olarak, imajın ta kendisi. Bir $I(x(t), y(t), t)$ fonksiyonu piksel değerlerini veriyor. Bu fonksiyonda x, y ekran koordinatlarına tekabül ediyor, t ise zaman, 1, 2, .. gibi değerleri indeks değerleri var, mesela $I(100, 200, 1)$, bize 1. video karesindeki $x = 100, y = 200$ 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 yonunde değişiyor.

Su faraziye yapılarak takip 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 mesela, 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 geçerlidir.



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. Bu da normal, kamerayı alıp mesela sağa doğru hareket ettiriyoruz, ve görüntüdeki tüm pikseller sola doğru gidiyorlar.



Tüm bunları modelimizde nasıl kullanırız?

Takip edilen nokta her karede aynı renkte ise, şu ifade doğru demektir

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

Eğer bu fonksiyonun zamana göre türevini alırsak

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

sonucu gelir. Esitliğin sağ tarafı sıfır, çünkü bir sabitin türevini aldık. 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 formülde dx/dt ve dy/dt , hareket halindeki (zaman geçerken) noktanın sonsuz küçüklikteki yer değişimi. Ayriksal bağlamda arka arkaya iki kare içindeki yer değişimi. O zaman,

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

Alttağiler ise mesafesel (spatial) gradyanlardır, bunların nasıl hesaplanacağını çok iyi biliyoruz!

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

Alttağiki ise resim karelerinin zamana göre türevidir.

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

Daha derli toplu olarak göstermek gerekirse ana formül nihai olarak şöyle

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

ya da

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

Şimdi u, v 'nin hesaplanmasına geleyim. Üstteki formülü bir veri noktası için yazmak yeterli değil. Ama bu formülü hem takip ettiğimiz, hem de onun etrafındaki

pikseller için yazarsak (onların yer değişimi de aynı değil mi?), ve bu sistemi çözersek, sonuca varabiliriz.

İki tane bilinmeyenimiz var, ama böylece pek çok formül elde ediyoruz. Veriler gürültülü olduğu için, aslında bilinmeyenden "daha fazla" formül elde etmek iyi, bu tür denklemlerine "çok eşitliğe sahip (overdetermined)" denir, ve böyle tür sistemler En Az Kareler (Least Squares) ile çözülür. Tüm bunları bir araya koyunca su ortaya çıkar.

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_k) & I_y(p_k) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_k) \end{bmatrix}$$

Gradyanların belli noktalarda hesaplandığını unutmayalım, o sebeple p_1, p_2 gibi piksel noktalarını bu fonksiyonlara geçiriyoruz.

Bu sistemi

$$A d = b$$

olarak gösterebiliriz, ki $d = \langle u, v \rangle$. Sol tarafı A^T ile çarpalım

$$A^T A d = A^T b$$

Eğer $A^T A$ 'nin matris tersini iki tarafla çarparsak, d yalnız kalır, ve sonuç elde edilir.

Bu denklemi Python Numpy'da `pinv` kullanarak çözeriz.

Test için üç tane resim kullandık, bu resimlerden `flow1-bw-0.png` başlangıç resmi, bu resmin ortasındaki objeleri GIMP kullanarak elle kopyaladık, bir üst sağ çapraz doğru, bir alt sol çapraz doğru, ve iki yeni resim elde ettik (`upright.png`, `dleft.png`). Takip edilen nokta gri dörtgenin alt sol köşesinde. Lucas Kanade algoritması bu noktayı takip ederek, yeşil ile işaretledi.

```
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

import warnings
warnings.simplefilter("ignore", np.ComplexWarning)

im1 = np.asarray(Image.open('flow1-bw-0.png'))
im2 = np.asarray(Image.open("upright.png"))
fx, fy, ft = deriv(im1, im2)
print fx[:5]

[[ 34.37477011  45.94010835  51.877951   ...,  53.83264716  51.877951
  45.94010835]
 [ 26.01168277  34.76327322  39.25648957 ...,  40.73562489  39.25648957
  34.76327322]
 [ 11.72919465  15.67546405  17.70154632 ...,  18.36851839  17.70154632
  15.67546405]
 [  3.51803959   4.70167857   5.30937909 ...,   5.50942984   5.30937909
   4.70167857]
 [  0.6961225   0.93033183   1.05057892 ...,   1.09016341   1.05057892
   0.93033183]]

import scipy.signal as si
from PIL import Image
import numpy.linalg as lin

def lk(im1, im2, i, j, window_size) :
    fx, fy, ft = 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]

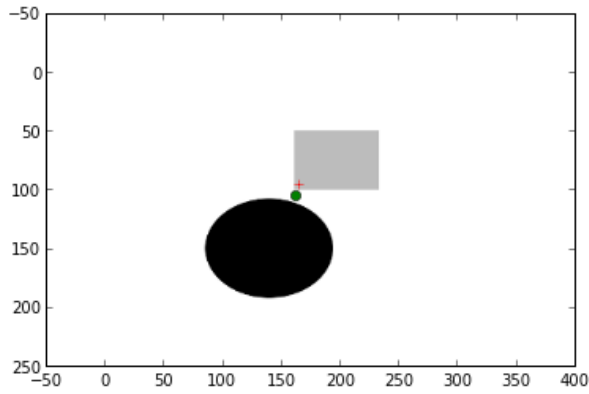
```

```

def test (image1,image2,output):
    x=165
    y=95
    win=50
    im1 = np.asarray(Image.open(image1))
    im2 = np.asarray(Image.open(image2))
    u, v = lk(im1, im2, x, y, win)
    plt.imshow(im1, cmap='gray')
    plt.hold(True)
    plt.plot(x,y,'+r');
    # 3 ile carptik cunku vektor degisimi iyi hesaplandi ama
    # grafikleme icin cok ufakti, ikinci yesil nokta iyi gozuksun
    # diye onu biraz buyuttuk
    plt.plot(x+u*3,y+v*3,'og')
    plt.savefig(output)

test('flow1-bw-0.png','dleft.png','lk_1.png')

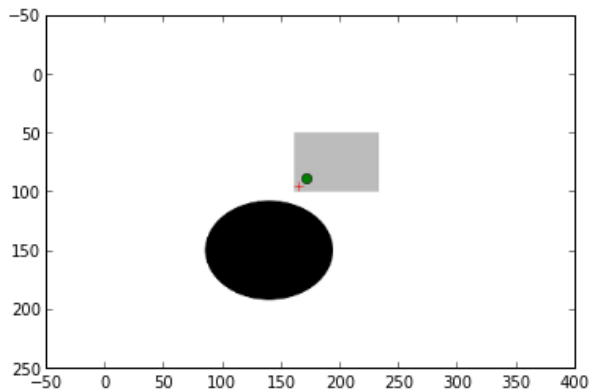
```



```

test('flow1-bw-0.png','upright.png','lk_2.png')

```



Bu matematiksel modele alternatif bir bakis soyle olabilir. Iki imaj karesi icinde birincisine $I(x, y)$ ikincisine $H(x, y)$ diyelim, burada t uzerinden parametrizasyon olmasin; x, y pikselinin H icinde u, v kadar yer degisiminden sonra, bu noktalarin

I'de geldiği yerdeki grilik degerinin aynı olduğunu (yine) farzediyoruz. Sonra $I(x + u, y + v)$ 'nin birinci dereceden Taylor Acilimini yapıyoruz,

$$I(x + u, y + v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \dots$$

ya da

$$I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

Grilik aynılığını ise şöyle belirtebiliriz

$$I(x + u, y + v) - H(x, y) = 0$$

Taylor acilini üstteki formülde I yerine geçirelim

$$I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v - H(x, y) = 0$$

H'in yerini degistirelim

$$I(x, y) - H(x, y) + I_x u + I_y v = 0$$

Su ifade $I(x, y) - H(x, y)$ nedir? Bunlar iki imajın, sonrası ve öncesi arasındaki fark değil midir? O zaman bu hesabi imajın zamana göre alınmış türevi olarak görebiliriz, yani $I_t = I(x, y) - H(x, y)$. Yerine koyalım

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

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

Boylece aynı denkleme erismiş olduk. Bu aslında normal, birinci dereceden Taylor acilimi ile tam diferansiyel denklemi (ve Zincirleme Kanununu) birbiriyle çok yakından alakası var.

Ufak Piksel Degisimleri

Konu hakkında bir nokta daha su; Lucas-Kanade yöntemi 1. derece Taylor acilimi kullandığı için ufak piksel değişimleri için geçerlidir, çünkü Taylor acilimi yerel bir noktaya çok yakın bölgelerde bir fonksiyona yakın sonuçlar verir. Bu da aklımızda bulunsun.

Kaynaklar

R. Collins Ders Notlari, www.cse.psu.edu/~rcollins/CSE486

Khurram Hassan-Shafique, CAP 5415 Lecture Notes, Spring 2003

<http://web.yonsei.ac.kr/jksuhr/articles/Kanade-Lucas-Tomasi%20Tracker.pdf>