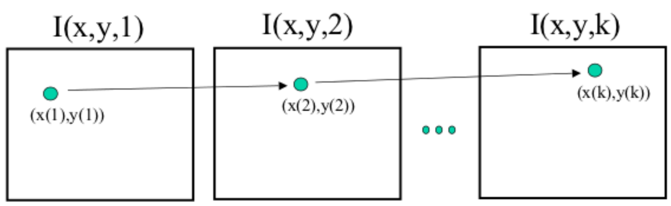
Piksel Takibi, Optik Akis, Lucas Kanade Algoritmasi

Hareket halindeki bir kameranin aldigi goruntulerdeki herhangi bir pikseli nasil takip ederiz?

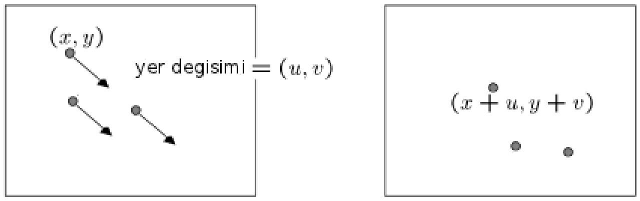
Matematiksel olarak temsil etmek gerekirse, zamana gore degisen 2 boyutlu goruntuyu bir fonksiyon olarak dusunelim, ki bu fonksiyonun degerleri ayriksal olarak, imajin ta kendisi. Bir $I(x(t), y(t), t)$ fonksiyonu piksel degerlerini veriyor. Bu fonksiyonda x, y ekran kordinatlarina tekabul ediyor, t ise zaman, $1, 2, ..$ gibi degerleri indeks degerleri var, mesela $I(100, 200, 1)$, bize 1. video karesindeki $x = 100, y = 200$ kordinatlarindaki piksel degerini verecek.

x, y degiskenleri parametrize edildi, bir noktayi takip etmek istiyoruz cunku, ve t'ye gore bu takip edilen noktanin x, y kordinatlari belli bir gidisat yonunde degisiyor.

Su faraziyeyi yaparak takip problemimizi kolaylastirabiliriz. Diyelim ki takip edilen bir nokta, goruldugu her karede ayni piksel rengindedir. Bu cok siradisi bir faraziye degil, resim karelerinden bir araba geciyor mesela, ve bu arabanin uzerindeki piksellerin renkleri, en azindan iki kare arasinda degismiyor. Isik seviyesi, golgede olma, vs. gibi durumlarda biraz degisebilir, fakat basitlestirme amaciyla bu faraziye gecerlidir.



Bir diger faraziye, kameralar hareket ettiklerinde alinan iki goruntu arasindaki tum piksellerin yer degisimi genellikle ayni yonde olmasidir. Bu degisim yonunu $< u, v >$ vektoru olarak gorebiliriz, ve bu degiskenler iki goruntu arasindaki degisimde tum pikseller icin ayni olacaktir. Bu da normal, kamerayi alip mesela saga dogru hareket ettiriyoruz, ve goruntudeki tum pikseller sola dogru 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 formulde dx/dt ve dy/dt, hareket halindeki (zaman gecerken) noktanin sonsuz kucuklukteki 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, bunlarin nasil hesaplanacagini cok iyi biliyoruz!

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

Alttaki ise resim karelerinin zamana gore turevidir.

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

Daha derli toplu olarak gostermek gerekirse ana formul nihai olarak soyle

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

ya da

$\nabla I \cdot < u, v >= -I_t$

Simdi u, v'nin hesaplanmasina gelelim. Ustteki formulu bir veri noktasi icin yazmak yeterli degil. Ama bu formulu hem takip ettigimiz, hem de onun etrafindaki pikseller icin yazarsak (onlarin yer degisimi de ayni degil mi?), ve bu sistemi cozersek, sonuca varabiliriz.

Iki tane bilinmeyenimiz var, ama boylece pek cok formul elde ediyoruz. Veriler gurultulu oldugu icin, aslinda bilinmeyenden "daha fazla" formul elde etmek iyi, bu tur denklem sistemlerina "cok esitlige sahip (overdetermined)" denir, ve boyle tur sistemler En Az Kareler (Least Squares) ile cozulur. Tum bunlari biraraya koyunca su ortaya cikar.

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

Gradyanlarin belli noktalarda hesaplandigini unutmayalim, o sebeple p_1, p_2 gibi piksel noktalarini bu fonksiyonlara geciyoruz.

Bu sistemi

$A \ d = b$

olarak gosterebiliriz, ki $d =< u, v >$. Sol tarafi A^T ile carpalim

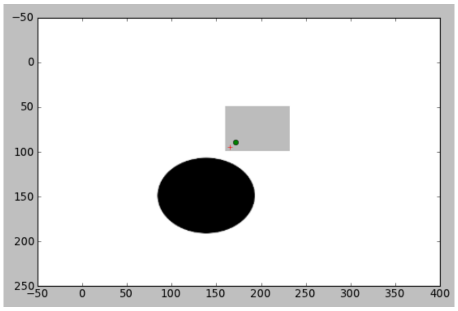
$A^T A \ d = A^T b$

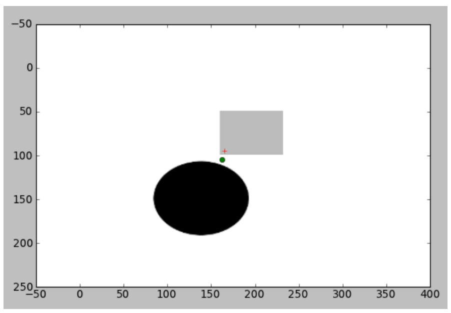
Eger $A^T A$'nin matris tersini iki tarafla carparsak, d yanliz kalir, ve sonuc elde edilir.

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

Test icin uc tane resim kullandik, bu resimlerden `flow1-bw-0.png` baslangic resmi, bu resmin ortasindaki objeleri GIMP kullanarak elle kopyaladik, bir ust sag capraza dogru, bir alt sol capraza dogru, ve iki yeni resim elde ettik (`upright.png`, `dleft.png`). Takip edilen nokta gri dortgenin alt sol kosesinde. Lucas Kanade algoritmasi bu noktayi takip ederek, yesil ile isaretledi.

OpenCV uzerinden bir video'daki imajin takip edilmesinin ornegi ise `track-chess.py` icinde bulunabilir.





```
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()
```

Not

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 geldigi yerdeki grilik degerinin ayni oldugunu (yine) farzediyoruz. Sonra $I(x+u,y+v)$'nin birinci dereceden Taylor Acilimini yapiyoruz,

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

ya da

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

Grilik ayniligini ise soyle belirtebiliriz

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

Taylor acilini ustteki formulde I yerine gecirelim

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

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 imajin, sonrasi ve oncesi arasindaki fark degil midir? O zaman bu hesabi imajin zamana gore alinmis turevi olarak gorebiliriz, yani $I_t = I(x,y) - H(x,y)$. Yerine koyalim

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

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

Boylece ayni denkleme erismis olduk. Bu aslinda normal, birinci dereceden Taylor acilimi ile tam diferansiyel denklemi (ve Zincirleme Kanununu) birbiriyle cok yakindan alakasi var.

Kaynaklar

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

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

http://dl.dropbox.com/u/1570604/skfiles/campy/chessb-left.avi