Filtrelemek

Filtreler dis dunyadaki bir aksiyon hakkinda elde edilen gurultulu sinyalleri, tersine cevirerek arka plandaki aksiyon hakkinda hesaplama yapabilmemizi saglar. Mesela Kalman Filtreleri (KF) icin gizlenmis konum bir robotun nerede oldugu, bir senetin fiyati gibi bir sey olabilir, gizli konum bilgisi x_t degiskeninde o konum hakkindaki gurultulu olcum y_t icindedir. Hem gizli konumlar arasindaki gecis, hem de olcumun gurultusu lineer bir fonksiyon uzerindendir.

$$x_{t+1} = Ax_t + v$$

$$y_t = Hx_t + w$$

v ve w'in dagilimi Gaussian'dir ve kovaryans sirasiyla Q ve R icindedir.

Zaman faktorunu de dahil etmek gerekirse;

$$\hat{x}_t^t = E[x_t|y_0, .., y_t]$$

$$P_t^t = E[(x_t - \hat{x}_{t|t})(x_t - \hat{x}_{t|t})'|y_0, ..., y_t]$$

Filtremenin amaci x_{t+1} ve P_{t+1} hesabini yeni bir olcum y_{t+1} uzerinden yapmak olacak. "Gizli" x_t derken bunu kastediyorduk, bu deger bize verilmiyor, sadece xt ve x_{t+1} arasindaki gecisin nasil oldugunu biliyoruz, gurultunun nasil eklendigini biliyoruz, ama bunlarin bilsek bile elde bir suru bilinmeyen var. Filtrelemenin matematiksel numaralari sayesinde bunu hesaplayabiliyor olacagiz. Yani yapmamiz gereken "oku tersine cevirmek", yani x_t'nin y_t uzerindeki sartsal bagliligini (conditional dependence) ortaya cikartmak, bunu y_t'nin x_t'ye olan sartsal bagimliligini tersine cevirerek yapmak. Ana denklemin iki tarafinin da beklentisini (expectation) alalim:

$$E \ x_{t+1} = \hat{x}_{t+1} = A\mu_t = A\hat{x}_t$$

Simdi iki tarafin kovaryansini alalim ve P_t'yi $cov \ x(t)$ olarak belirtelim:

$$P_{t+1} = AP_tA' + Q$$

Bu gecis "zaman guncellemesi" olarak adlandirilir. Normal dagilimlari t anindan $t+1$ anina gecirmemizi saglar. y iceren formullerde benzer bir durum var.

$$\hat{x}_{t+1}^t = Ax_t^t$$

$P_{t+1}^t = AP_t^t A' + Q$

$y_{t+1} = Cx_{t+1} + w_t$

$E[y_{t+1}|y_0, .., y_t] = E[Cx_{t+1} + w_t|y_0, .., y_t]$

$\hat{y}_{t+1}^t = C\hat{x}_{t+1}$

Kovaryans icin benzer durum

$E[(y_{t+1} - \hat{y}_{t+1}^t)(y_{t+1} - \hat{y}_{t+1}^t)'|y_0, ..., y_t] = C_{t+1}^t C' + R$

Simdi daha zor is olan oku tersini cevirmeye gelelim. Eger amacimiz p(xt —yt) denklemini elde etmek ise o zaman bu iki degiskeni iceren birlesik dagilimi (joint distribution) elde etmek zorundayiz. Iki Gaussian'in birlesiminin yeni bir Gaussian oldugunu biliyoruz, o zaman hem x_t hem de y_t'in kendisi cok boyutlu birer Gaussian olduklari icin onlarin birlesimi $p(x_t|y_t)$'in hakikaten devasa bir Gaussian olacagini tahmin edebiliriz.

x_t ve y_t'in birlesimi olan Gaussian'i bulmak demek, bu Gaussian'in ortalamasini (mean) ve kovaryansini bulmak demektir cunku bir Gaussian ortalama ve kovaryansi ile net bir sekilde tanimlanabilir bir seydir. Bir numara yapalim, ve $y_t = Cx_t + w_t$'yi $z = Hu$ seklinde yazalim. Sonra

$$\left[\begin{array}{c} x_t \\ y_t \end{array} \right], H = \left[\begin{array}{cc} I & 0 \\ C & I \end{array} \right], u = \left[\begin{array}{c} x_t \\ w_t \end{array} \right]$$

Boylece daha basit bir denklemin kovaryansini alabiliriz

$cov(z) = H\ cos(u) H'$

$$cov(u) = \left[\begin{array}{cc} P_t & 0 \\ 0 & R \end{array} \right]$$

Tam carpim suna esit

$$\left[\begin{array}{cc} I & 0 \\ C & I \end{array} \right] \left[\begin{array}{cc} P_t & 0 \\ 0 & R \end{array} \right] \left[\begin{array}{cc} I & C' \\ 0 & I \end{array} \right]$$

bunun sonucu ise

$$\left[\begin{array}{cc} P_t & P_t C' \\ CP_t & CP_t C' + R \end{array} \right]$$

Bunu baglantisal denklem icin ve ortalamayi icerecek sekilde yazabiliriz

$$\left[\begin{array}{c} \hat{x}_t^t \\ C\hat{x}_t^t \end{array}\right], \left[\begin{array}{cc} P_t^t & P_t^t C' \\ CP_t^t & CP_t^t + R \end{array}\right]$$

Ayni sekilde x_{t+1}, y_{t+1} birlesik dagilim icin

$$\left[\begin{array}{c} \hat{x}_{t+1}^t \\ C\hat{x}_{t+1}^t \end{array}\right], \left[\begin{array}{cc} P_{t+1}^t & P_{t+1}^t C' \\ CP_{t+1}^t & CP_{t+1}^t C' + R \end{array}\right] \tag*{(1) (eq1)}$$

Simdi x_{t+1}^{t+1} 'in ortalama ve varyansi icin parcali Gaussian kavramini anlatmaliyiz. Bir n boyutlu Gaussian daha kucuk boyutlardaki p ve q alt Gaussian'lara parcalanabilir (tabii ki $n = p + q$). Yani su ifade kullanilabilir

$$\mu = \left[\begin{array}{c} \mu_1 \\ \mu_2 \end{array}\right], \Sigma = \left[\begin{array}{cc} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{array}\right] \tag*{(2) (eq2)}$$

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{(p+q)/2}|\Sigma|^{1/2}} exp\bigg\{ -\frac{1}{2} \left(\begin{array}{c} x_1 - \mu_1 \\ x_2 - \mu_2 \end{array}\right)' \left[\begin{array}{cc} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{array}\right]^{-1} \left(\begin{array}{c} x_1 - \mu_1 \\ x_2 - \mu_2 \end{array}\right) \bigg\}$$

Uzun cebirsel islemlerden sonra $p(x_1|x_2)$ ifadesini elde ederiz. Buradan sartlanmis (conditioned) μ ve Σ alinir.

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) \tag*{(3) (eq3)}$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

Simdi denklem eq3 3'u alip eq1 1'in icine koydugumuzda ve eq2 2'deki yerlesim yapisini dikkate aldigimizda $\hat{x}_{t+1}^{t+1}$ ve P_{t+1}^{t+1} formullerini ortaya cikartabiliriz.

$$\hat{x}_{t+1}^{t+1} = x_{t+1}^t + P_{t+1}^t C'(CP_{t+1}^t C' + \Sigma_w)^{-1}(y_{t+1} - C\hat{x}_{t+1}^t)$$

$$P_{t+1}^{t+1} = P_{t+1}^t - P_{t+1}^t C'(CP_{t+1}^t C' + R)^{-1} CP_{t+1}^t$$

Eger $K = P_{t+1}^t C'(CP_{t+1}^t C' + \Sigma_w)^{-1}$ dersek

$$\hat{x}_{t+1}^{t+1} = \hat{x}_{t+1}^t + K_t(y_{t+1} - C\hat{x}_{t+1}^t$$

$$P_{t+1}^{t+1} = P_{t+1}^t - K_t CP_{t+1}^t$$

Ornek: Veriye Duz Cizgi Uydurmak (Line Fitting)

Eger elimizde bir cizgiye uydurmak icin kullanacagimiz tum veri olsaydi, uydurma islemi icin en az kareler (least sqaures) yontemini kullanabilirdik. Kalman Filtreleri bize yeni veri geldigi anda, her seferinde, azar azar bir cizgiyi uydurmamizi sagliyor. Hatta matematiksel olarak ispalanmistir ki eger baslangic noktasi ayniysa, azar azar veriyi KF ile almanin sonunda, tum veriyi bir kerede en az karesel yontem ile uydurmak ayni sonucu verir.

Peki bu uydurma islemini nasil yapariz? Burada veriyi nasil temsil ettigimiz konusunda ufak bir numara kullanmamiz lazim.

Kendimize bir soru soralim: bu sistemin konum bilgisi nedir? Bir robotu izliyorsak mesela soru cevabi basittir, onun x, y gibi kordinat bilgisi. Duz cizgi fit ederken takip edilen bunlar degil, bize gerekli olan bir cizginin "egimi (slope)". Yani hem bir cizginin y eksenini kestigi nokta, hem de cizginin egimi xt konum bilgisi icinde dahil edilecek. Burada KF literaturunden gelen x, y harfleri birbirine karismasin diye cizginin degerlerini xx_t ve yy_t olarak tanimlayacagiz. O zaman x_t vektoru suna benzer:

$$x_t = \begin{bmatrix} yy_t \\ a \end{bmatrix}$$

ki burada a harfi egimi temsil etmektedir. a bir sabit olduguna gore KF her zaman diliminde ayni kalacak bir degiskeni hesaplayacaktir. Cogunlukla KF ile her zaman diliminde degisik olan degerlerin hesaplandigini goruruz, bu uygulamaya gore degisen bir seydir, matematiksel bir mecburiyet degildir. A matrisimiz ile de biraz numara yapmamiz gerekli. Bu matris x_t'yi donusturup x_{t+1}'i elde etmemizi saglayan sey olduguna gore A'nin soyle olmasi gerekir:

$$A_t = \begin{bmatrix} 1 & \Delta xx \\ 0 & 1 \end{bmatrix}$$

Bu matrisi x_t ile carptigimizda $yy_t \cdot 1 + a \cdot \Delta xx$ degerini elde ediyoruz, ki bu deger bir cizgi uzerinde bir sonraki noktayi temsil ediyor. Dis olcumu veren gurultu matrisi H ise

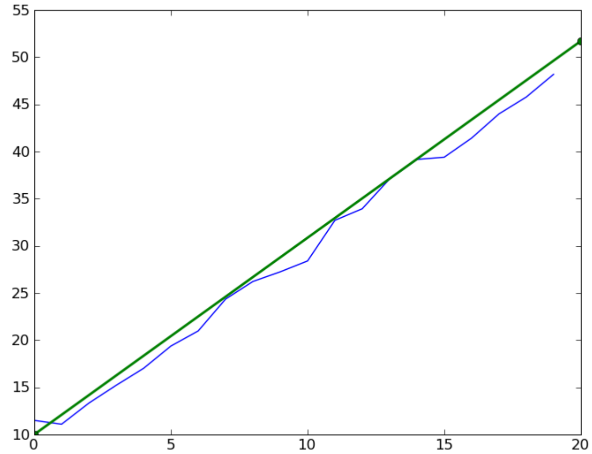
$$H = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

seklinde. Bunu x_t ile carptigimizda y_t'yi (iki kere) elde ettigimizi gorecegiz. Not: Niye iki kere? Kodlama sirasinda boyutlarin uyumlu olmasi icin boyle gerekti, cok buyuk bir rahatsizlik degil. Kod altta gorulebilir.

```
from pylab import *
from numpy import *
```

```
slope = 2

#
# x_{t+1} = A x_t + Q
# y_t = Hx_t + R
#
def Kalman(obs,x,mu_init,nsteps):

    ndim = shape(mu_init)[0]

    Q = zeros((ndim, ndim))
    A = eye(ndim)
    H = array([[1, 0], [1, 0]])

    mu_hat = mu_init
    cov = ones((ndim, ndim))
    R = eye(ndim) * 10

    m = zeros((ndim,nsteps),dtype=float)
    ce = zeros((ndim,ndim,nsteps),dtype=float)

    for t in range(1,nsteps):
        # Make prediction
        # A is transformation matrix, equals to
        # TR: Tahmini yap
        # A transofmrasyon matrisi ve suna esit
        # | 1 delta_x |
        # | 0     1   |
        A = array([[1, x[t]-x[t-1]], [0, 1]])
        mu_hat_est = dot(A,mu_hat)
        cov_est = dot(A,dot(cov,transpose(A))) + Q

        # Update estimate
        # TR: tahmini guncelle
        error_mu = obs[:,t] - dot(H,mu_hat_est)
        error_cov = dot(H,dot(cov,transpose(H))) + R
        K = dot(dot(cov_est,transpose(H)),linalg.inv(error_cov))
        mu_hat = mu_hat_est + dot(K,error_mu)
```

```
        m[:,t] = mu_hat
        cov = dot((eye(ndim) - dot(K,H)),cov_est)
        ce[:,:,t] = cov
        print "mu_hat="+str(mu_hat)
    return mu_hat

N = 20

#
# create sample data
# TR: ornek veri yarat
#

obs = zeros((2, N))
x = xrange(N)
for i in xrange(N):
    obs[0, i] = obs[1, i]  = (slope*i)+random.normal(10)

print "obs="+str(obs.shape)

mu_hat = Kalman(obs, x, mu_init=array([0, 0]),nsteps=N)

plot(obs[0, :])
plot([0,N], [10,N*mu_hat[1]], 'go-', label='line 1', linewidth=2)
show()
```



Ornek: Obje Takibi

Daha degisik bir ornekten bahsedelim. Bu ornekte OpenCV kutuphanesinden elde ettigimiz 2 boyutlu degerleri y_t icin kullanacagiz. Degerler OpenCV'nin bir satranc tahtasi seklinin kose noktalarini otomatik olarak bulabilen cvFindChessboardCorners cagrisindan gelecek (ayrica cvDrawChessboardCorners ile bu noktalari ekranda aninda gosterebilecegiz).

Elimizdeki "gurultulu" olcumler iki boyutlu noktasal degerler. Gurultulu cunku kamera bize bu imajlari aktarirken hata eklemis olabilir, OpenCV fonksiyonu hesabi yaparken hata eklemis olabilir, bir suru olasilik var.

Bu ornekte, ayrica, ilk kez KF ortaminda boyut degisikligi olasiligini net bir sekilde gorebiliyoruz. Gizli konum bilgisi x_t 3 boyutlu bir nokta, ama elimizdeki olcum 2 boyutlu bir "yansima". Yansima sirasinda kacinilmaz olarak deger kaybediliyor, bir boyutun bilgisi ortadan yokoluyor. Ama tum bu bilinmezlere ragmen Kalman filtresinin bizim icin gizli bilgiyi hesaplamasini istiyoruz.

Bu problemde A matrisi ne olacaktir? Obje takibi konularinda A'nin ne oldugunu hayal etmek daha kolay, A matrisi iki zaman dilimi arasindaki "hareketi" temsil edecek. Bu problemdeki ek bir kolaylik bu hareketi onceden bildigimiz, ve hareketin tek yonde oldugu. Yani resimde benim tuttugum kartonu ne kadar hizla hareket ettirdigimi ben onceden probleme bildiriyorum. Yer degisikligini d olarak betimledim, ve A soyle oldu:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dikkat edersek A 4x4 boyutunda, 3x3 degil. 3 boyutlu kordinatlari temsil etmek icin homojen kordinat sistemini kullandigimiz icin boyle oldu, o sebeple zaten x_t de 4x1 oldu, ona uymak icin A'nin degismesi gerekiyordu. Ax_t carpiminin hakikaten kartonu hareket ettirdigini gostermek icin bu carpimi bir ornek uzerinde yapalim: Diyelim ki $x_t = [a_1 \ a_2 \ a_3 \ a_4]$ o zaman Ax_t ya da x_{t+1} su hale gelir: $[a_1 \ a_2 \ a_3 + d \ a_4]$.

Bakiyoruz, hakikaten de d kadarlik bir yer degisimi z kordinati, yani derinlik uzerinde eklenmis. Test amaclarimiz icin d = -0.5 aldik, yani satranc tahta kartonunun her zaman diliminde kameraya dogru 0.5 cm ilerledigini belirttik. Tabii bu da kabaca bir tahmindi (her ne kadar hareketi yaptiran ben olsam bile!), ama filrelemenin gucunu burada goruyoruz. Benim tahminimde “gurultu” yani “hata payi” var, olcumde gurultu var, tum bunlar ust uste konsa bile filtre yine de gizli konumu bulacak.

Olcumsel donusumu temsil eden H'e ben onun temeli olan yansima (projection) kelimesinden gelen P matrisinden bahsedelim. Yansima matrisi goruntu (vision) literaturunde tek delikli kamera (pinhole camera) modelinden ileri gelen bir matristir ve bu matrisi hesaplamak ayarlama / kalibrasyon (calibration) denen apayri bir islemin parcasidir. OpenCV icinde kalibrasyon icin fonksiyonlar var, biz de bunlari denedik, kalibrasyon icin kullandigimiz resimlerle alakali olmali, elde edilen sonuclardan memnun kalmadik. Alternatif olarak sunu yaptik; resimde gorulen yesil yuzey bizim programin olusturdugu hayali bir yuzey. Filtrenin o anki tahminini P uzerinden goruntuye yansitarak bu yuzeyi olusturduk, boylece deneme / yanilma yontemiyle pek cok P degerini deneyerek, yuzeyin resimde gorulen masanin sonunda cikacak sekilde olmasini sagladik. O noktaya gelince istedigimiz P degerini bulmus oluyorduk. Yansitma matrisleri 3x3 olur, KF buna bir dorduncu [0 0 0] satiri ekleyerek onu 4x3 H haline getiriyor.

KF'in baslangic noktasi olarak P'yi bulmak icin kullandigimiz masa sonunu kullandik. Kararsizlik olcutu Q icin, ki bu degisken bir Gaussian kovaryansidir, $Q = I \cdot 150cm$ degerini kullandik, yani oldukca buyuk bir kararsizlik degeri kullandik. Sebep baslangic degeri olan masa ortasini sectik, ve takip edecegimiz satranc tahtasinin nerede oldugunu bilmiyoruz, “emin degiliz”. Bu kararsizligi sayisal olarak programa bildirmis olduk.

Alttaki resimlerde filtrenin tahminini temsil eden yesil yuzeyin satranc tahtasini basariyla takip ettigini goreceksiniz.

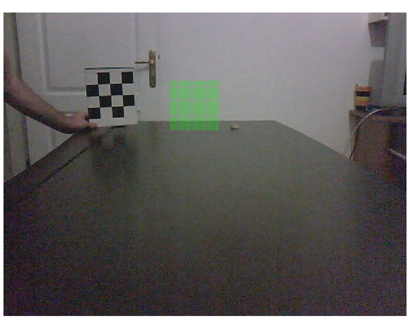
```
from numpy import *

# x_{t+1} = A x_t + Sigma_x
# y_t = Hx_t + R
class Kalman:
    # T is the translation matrix
    # K is the camera matrix calculated by calibration
    def __init__(self, K, mu_init):
        self.ndim = 3
        self.Sigma_x = eye(self.ndim+1)*150
        self.A = eye(4)
        self.A[2,3] = -0.5
        self.H = append(K, [[0], [0], [0]], axis=1)
        self.mu_hat = mu_init
        self.cov = eye(self.ndim+1)
        self.R = eye(self.ndim)*1.5

    def normalize_2d(self, x):
        return array([x[0]/x[2], x[1]/x[2], 1.0])

    def update(self, obs):

        # Make prediction
        print "self.mu_hat=" + str(self.mu_hat)
        self.mu_hat_est = dot(self.A, self.mu_hat)
        prod = dot(self.A, dot(self.cov, transpose(self.A)))
        self.cov_est = prod + self.Sigma_x
        print "self.mu_hat_est=" + str(self.mu_hat_est)
        print "self.cov_est=" + str(self.cov_est)

        # Update estimate
        prod = self.normalize_2d(dot(self.H, self.mu_hat_est))
```

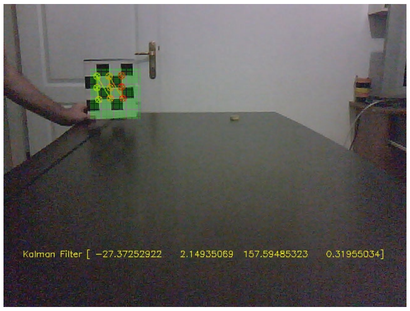
```
        self.error_mu = obs - prod

        prod = dot(self.cov, transpose(self.H))
        prod = dot(self.H, prod)
        self.error_cov = prod + self.R
        prod = dot(self.cov_est, transpose(self.H))
        self.K = dot(prod, linalg.inv(self.error_cov))
        self.mu_hat = self.mu_hat_est + dot(self.K, self.error_mu)

        prod = dot(self.K, self.H)
        left = eye(self.ndim+1)
        diff = left - prod
        self.cov = dot(diff, self.cov_est)

if __name__ == "__main__":

    # camera matrix
    K = array([[653.52398682, 0., 326.47888184],
               [0., 653.76440430, 259.63595581],
               [0., 0., 1.]])

    kalman = Kalman(K, mu_init=array([1., 1., 165., 1]))
    kalman.update(array([100.0, 100.0, 1.]))
    kalman.update(array([120.0, 120.0, 1.]))

import cv
import sys
from kalman_3d import *
from K import *

def proj_board(im, xl, yl, z):
    color = cv.CV_RGB(0, 255, 0)
    image_size = cv.GetSize(im)
    print "image_size=" + str(image_size)
    for x in arange(xl-9, xl+9, 0.5):
        for y in arange(yl-9, yl+9, 0.5):
            X = array([x, y, z])
            q = dot(K, X)
```

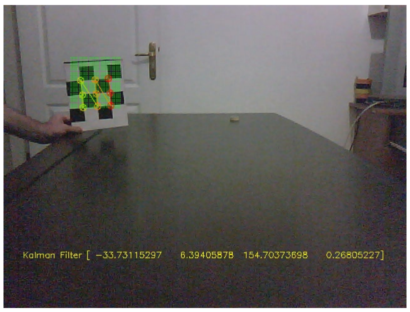
```
            q = [int(q[0]/q[2]), int(q[1]/q[2])]
            #if q[0] < image_size.height and q[1] < image_size.width an
            cv.Set2D(im, im.height-q[1], q[0], color)

def show_data(image, mu_x):
    line_type = cv.CV_AA
    pt1 = (30, 400)
    font = cv.InitFont (cv.CV_FONT_HERSHEY_SIMPLEX, 0.8, 0.1, 0, 1, cv.C
    cv.PutText (image, "Kalman Filter " + str(mu_x), pt1, font, cv.CV_RO

def detect(image):
    image_size = cv.GetSize(image)

    # create grayscale version
    grayscale = cv.CreateImage(image_size, 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    storage = cv.CreateMemStorage(0)
    #cv.ClearMemStorage(storage)

    im = cv.CreateImage (image_size, 8, 3)

    status, corners = cv.FindChessboardCorners( grayscale, (dim,dim))
    if status:
        cv.DrawChessboardCorners( image, (dim,dim), corners, status)
        is_x = [p[0] for p in corners]
        is_y = [p[1] for p in corners]
        return is_x, is_y
    return [], []

if __name__ == "__main__":

    print "Press ESC to quit, 't' to take a picture (image will be "
    print "saved in a snap.jpg file"

    snap_no = 1
    frame_no = 0

    # create windows
```

```python
cv.NamedWindow('Camera', cv.CV_WINDOW_AUTOSIZE)

# create capture device
device = 0 # assume we want first device

capture = cv.CreateFileCapture (sys.argv[1])
#capture = cvCreateCameraCapture (0)

dim = 3

pts = dim * dim
mid = int(pts / 2)

cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_WIDTH, 640)
cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_HEIGHT, 480)

# check if capture device is OK
if not capture:
    print "Error opening capture device"
    sys.exit(1)

kalman = Kalman(K, mu_init=array([1., 1., 165., 1.]))

frame = cv.QueryFrame(capture)
#proj_board(frame, 1, 1, 160)
print frame.height
cv.ShowImage('Camera', frame)
cv.SaveImage('snap-00.jpg', frame)
k = cv.WaitKey()

while 1:
    frame_no += 1
    print frame_no

    # capture the current frame
    frame = cv.QueryFrame(capture)
    image_size = cv.GetSize(frame)
    if frame is None:
```

```
        break

    is_x, is_y = detect(frame)

    if len(is_x) > 0 :
        print "is_x[5]=" + str(is_x[5])
        print "is_y[5]=" + str(is_y[5])
        kalman.update(array([is_x[5], frame.height-is_y[5], 1.]))
        proj_board(frame,
                   kalman.mu_hat[0],
                   kalman.mu_hat[1],
                   kalman.mu_hat[2])

        show_data(frame, kalman.mu_hat)

    # display webcam image
    cv.ShowImage('Camera', frame)

    # handle events
    k = cv.WaitKey(40)
    #k = cv.WaitKey()
    if k == "t":
        cvSaveImage('cb-kf-' + str(snap_no) + '.jpg', frame)
        snap_no += 1
    if k == 27: # ESC
        print 'ESC pressed. Exiting ...'
        break
```







Parcacik Filtreleri

Filtrelemede tek yontem Kalman filtreleri degil. KF kararsizlik Gaussian olarak gosterilebiliyorsa cok faydali, ve hizli bir yontem. Bir KF bellekte cok az yer tutar, 3 boyutlu bir Gaussian icin 3x1 boyutunda bir ortalama vektoru, ve 3x3 boyutunda bir kovaryans matrisi yeterlidir, yani $3 + 9 = 12$ sayi.

Parcacik filtreleri (PF) bir dagilimi "ayriksal" olarak temsil ederler. Yani diyelim ki tek boyutlu bir dagilimi 100 eleman iceren bir dizin ile temsil edebiliriz, o zaman dagilimin degerlerini 100 tane noktada tasimamiz gerekir. Bunun faydalari her turlu dagilim seklini temsil edebilmemiz. Gaussian sadece belli bir sekilde olabilir, tek bir tepe noktasi olmalidir, vs. Ayriksal temsil ile 2, 3, istedigimiz kadar tepe noktasi olan (ya da hic olmayan) bir dagilim kullanabiliriz.

Bu neye yarar? Birden fazla hipotezi ayni anda isletebilmemize yarar. KF ile tepe noktasi en iyi tahminimizdir (mesela.. satranc kartonu masa ortasinda), PF ile birkac tahmini ayni anda hesaplatmak mumkun olabilir.

Daha detaylandirmak gerekirse, PF kodlamasi x_t icin iki tane veri yapisi gerektirir. Bir veri yapisi dagilimdaki degerleri temsil eden parcaciklardir, digeri ise bu parcaciklarin dagilimdaki onemini temsil eden agirliklardir. Filtreleme sistemi KF'e benzer, once bir gecis uygulanir, ki bu gecis kararsizligi arttiracaktir, fakat ardindan gozlem verisi bir hata fonksiyonu uzerinden dagilim guncellenir. Bu islem sirasinda hatasi yuksek olan parcaciklar cezalandirilir, onlarin agirligi azalir, otekilerinki yukselir. Her parcacik icin hata fonksiyonu sudur:

$$w^{[i]} = \frac{1}{1 + (y^{[i]} - p^{[i]})^2)}$$

$y^{[i]}$ gozlem degeri, $p^{[i]}$ gecis uygulandiktan sonra elimizdeki tahminimizdir, ki bu KF dunyasindaki $Ax_t + Q$'nun karsiligidir. PF icin hareket gecisi soyle hesaplanir: Bir uniform dagilimdan ornekleme yapilir, ve bu orneklenen degerler x'e eklenir. Ornekleme icin z-kordinati icin $Unif(-0.1, -1)$'i, x kordinati icin $Unif(-40, 40)$'i kullandik. Yani ileri dogru 0.1 ve 1 santimetre arasinda bir hareket ekliyoruz, ve saga ve sola donuk olarak 80 santimetrelik bir kararsizligi hesaplara ekliyoruz.

Ustteki formulde $(y^{[i]} - p^{[i]})^2$ e niye 1 degeri ekledigimiz aciktir herhalde, bu sayede hata fonksiyonunun olasilik degerlerini andiran bir sonuc dondurmesini istiyoruz. Cok ufak hatalar icin $1 + hata$ bolunendeki 1'i bolecek, ve 1'e yakin bir deger geri getirecek. Istedigimiz de bu zaten, kucuk hatalarin daha buyuk agirliga sebebiyet vermeleri, buyuk hatalarin ise tam tersi sonuca sebep olmalari.

Tekrar ornekleme (resampling) surecinde parcaciklar tekrar duzenlenerek agirligi cok olan parcaciklarin agirligi az olanlara gore daha fazla tekrarlanmasini istiyoruz. Dikkat: tekrar ornekleme sureci yeni parcacik degerleri yaratmiyor, sadece mevcut olanlari tekrarliyor ya da onlari atliyor.

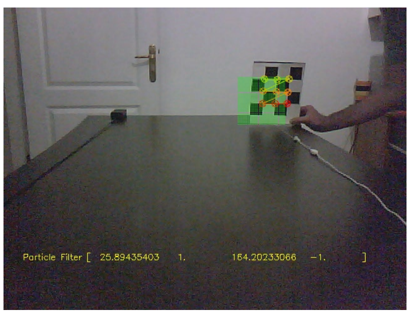
```
from numpy import *
from numpy.random import *

class PF:

    def __init__(self, K, n):
        self.H = append(K, [[0], [0], [0]], axis=1)
        self.n = n
        self.x = zeros((self.n, 4))
        self.x[:,:] = array([1., 1., 165., -1])

    def normalize_2d(self, x):
        return array([x[0]/x[2], x[1]/x[2], 1.0])

    def resample(self, weights):
      n = len(weights)
      indices = []
      C = [0.] + [sum(weights[:i+1]) for i in range(n)]
      u0, j = random(), 0
      for u in [(u0+i)/n for i in range(n)]:
        while u > C[j]:
          j+=1
        indices.append(j-1)
      return indices

    def update(self, y):
        u = uniform(-0.1, -1, self.n) # forward with uncertainty
        self.x[:,2] += u
        u = uniform(-40,40, self.n) # left right uncertainty
        self.x[:,0] += u
        p = dot(self.x,self.H.T)
        for i, item in enumerate(p): # modify in place
```

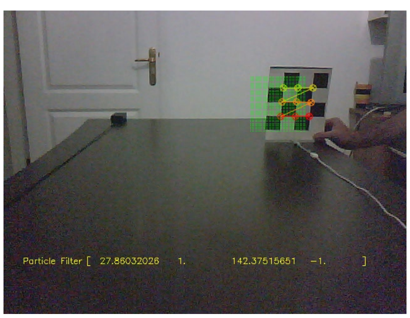
```
                p[i,:] = self.normalize_2d(item)
            self.w  = 1./(1. + (y-p)**2)
            self.w = self.w[:,0]+self.w[:,1]
            #self.w = self.w[:,0]
            self.w /= sum(self.w)
            self.x  = self.x[self.resample(self.w),:]

    def average(self):
        return sum(self.x.T*self.w, axis=1)

if __name__ == "__main__":

    K = array([[700., 0., 300.],[0., 700., 330.],[0., 0., 1.]])
    p = PF(K, 100)
    p.update(array([100.,100.,1.]))
    print p.average()

import sys
import cv
from numpy import *
from K import *
from PF import *

def proj_board(im, xl, yl, z):
    color = cv.CV_RGB(0, 255, 0)
    image_size = (im.width, im.height)
    print "image_size=" + str(image_size)
    for x in arange(xl-9, xl+9, 0.5):
        for y in arange(yl-9, yl+9, 0.5):
            X = array([x, y, z])
            q = dot(K, X)
            q = [int(q[0]/q[2]), int(q[1]/q[2])]
            #if q[0] < image_size.height and q[1] < image_size.width an
            cv.Set2D(im, im.height-q[1], q[0], color)

def detect(image):
    image_size = cv.GetSize(image)
```

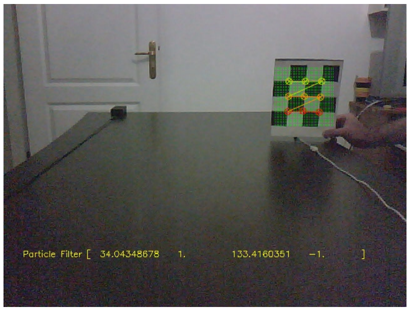
```
    # create grayscale version
    grayscale = cv.CreateImage(image_size, 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    storage = cv.CreateMemStorage(0)
    #cvClearMemStorage(storage)

    im = cv.CreateImage (image_size, 8, 3)

    status, corners = cv.FindChessboardCorners( grayscale, (dim,dim))
    if status:
        cv.DrawChessboardCorners( image, (dim,dim), corners, status)
        is_x = [p[0] for p in corners]
        is_y = [p[1] for p in corners]
        return is_x, is_y
    return [], []

def show_data(image, mu_x):
    line_type = cv.CV_AA
    pt1 = (30, 400)
    font = cv.InitFont (cv.CV_FONT_HERSHEY_SIMPLEX, 0.8, 0.1, 0, 1, cv.C
    cv.PutText (image, "Particle Filter " + str(mu_x), pt1, font, cv.CV


if __name__ == "__main__":

    print "Press ESC to quit, 't' to take a picture (image will be "
    print "saved in a snap.jpg file"

    snap_no = 0
    frame_no = 0

    # create windows
    cv.NamedWindow('Camera')

    # create capture device
    device = 0 # assume we want first device

    capture = cv.CreateFileCapture (sys.argv[1])
```

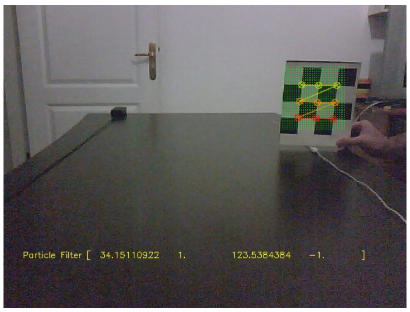
```
dim = 3
forward_step = -1.

pts = dim * dim
mid = int(pts / 2)

cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_WIDTH, 640)
cv.SetCaptureProperty(capture, cv.CV_CAP_PROP_FRAME_HEIGHT, 480)

# check if capture device is OK
if not capture:
    print "Error opening capture device"
    sys.exit(1)

pf = PF(K, 200)

frame = cv.QueryFrame(capture)
proj_board(frame, 1, 1, 160)
cv.ShowImage('Camera', frame)
k = cv.WaitKey()

while 1:
    frame_no += 1
    frame = cv.QueryFrame(capture)

    image_size = cv.GetSize(frame)
    if frame is None:
        break

    is_x, is_y = detect(frame)

    if len(is_x) > 0:
        print "is_x[5]=" + str(is_x[5])
        print "is_y[5]=" + str(is_y[5])
        pf.update(array([is_x[5], frame.height-is_y[5], 1.]))
        mu_x = pf.average()
        print mu_x
        proj_board(frame, mu_x[0], mu_x[1], mu_x[2])
```

```
        show_data(frame, mu_x)

    # display webcam image
    cv.ShowImage('Camera', frame)

    # handle events
    #k = cvWaitKey(40)
    k = cv.WaitKey()
    if k == "t":
        cv.SaveImage('snap-' + str(snap_no) + '.jpg', frame)
        snap_no += 1
    if k == 27: # ESC
        print 'ESC pressed. Exiting ...'
        break
```









Kaynaklar

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

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

S. Marsland, Machine Learning: An Algorithmic Perspective, CRC Press, 2009.

S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, MIT Press, Cambridge, MA, 2005

C. Bishop Pattern Recognition and Machine Learning , 2006.

Rabiner L. R. , A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Proceedings of IEEE vol. 77, no. 2, pp. 257-286, 1989.

Roweis S. and Z. Ghahramani, A Unifying Review of Linear Gaussian Models, Neural Computation 11(2):305-345, 1999.

Ghahramani Z., H. E. Hinton, Parameter Estimation for Linear Dynamical Systems, Technical Report CRG-TR-96-2 ftp://ftp.cs.toronto.edu/pub/zoubin/tr96-2.ps.gz], Department of Computer Science, University of Toronto, 1996.

Ghahramani Z., H. E. Hinton, Switching State Space Models, Technical Report CRG-TR-96-3, Dept. Comp. Sci., Univ. Toronto, 1996.

Shumway R., H. S. Stoffer Time series analysis and its applications 2nd Edition, New York, Springer, (Springer texts in statistics), 2000.

Jordan M. I. , C. Bishop An Introduction to Graphical Models, Not yet published, 2000.

Kalman R. E., A New Approach to Linear Filtering and Prediction Problems, Transactions of the ASME-Journal of Basic Engineering, 82 (Series D): 35-45, 1960.

Kalman, R.E. and R.S. Bucy, New results in filtering and prediction theory, Trans. ASME J. Basic Eng., 83, 95-108, 1961.

Welling, M., The Kalman Filter - Lecture Tutorial, California Institute of Technology, 2008.

Lall, S., Modern Control 2 Lecture Notes, Stanford University, 2006.