# APÉNDICE A. CÓDIGO

**ex1.m**

```
clear all;

% Llamar a generasec()
[num, sec] = generasec();

% Formato, inicialización de variables
fprintf ('\n');
total1 = 0; total2 = 0; angt = 0;

for (k=2:num)

   % Obtención de las matrices Vx, Vy que describen el campo de flujo óptico
   % entre las imagenes subsecuentes
   [Vx,Vy,reliab] = optflow_corr( sec(:,:,k-1), sec(:,:,k), 3, 15, .5, .01, k+num-1 );
   % Identificación del tipo de movimiento
   [promx, promy, ang] = tipomov(Vx, Vy);
   % Calculando el movimiento total...
   total1=total1+promx;
   total2=total2+promy;
   angt=angt+ang;

end

% Despliegue de resultados
fprintf ('Movimiento total: en X: %i, en Y: %i \n', total1, total2);
fprintf ('Rotacion: %i grados \n', angt);
```

**ex2.m**

```
clear all
% Llamar a lectura de imágenes, para archivos .png,
[I1, I2, I3]  = lectura();

% Formato, inicialización de variables
fprintf ('\n');
total1 = 0; total2 = 0; angt = 0;

% Obtención de las matrices Vx, Vy que describen el campo de flujo óptico
% entre las imagenes I1 e I2
[Vx,Vy,reliab] = optflow_corr( I1, I2, 8, 5, 1, .01, 10 );

% Identificación del tipo de movimiento
[promx, promy, ang] = tipomov(Vx, Vy);

% Calculando el movimiento total...
```

```
total1=total1+promx;
total2=total2+promy;
angt=angt+ang;

% Obtención de las matrices Vx, Vy que describen el campo de flujo óptico
% entre las imagenes I2 e I3
[Vx,Vy,reliab] = optflow_corr( I2, I3, 8, 5, 1, .01, 11 );

% Identificación del tipo de movimiento
[promx, promy, ang] = tipomov(Vx, Vy);

% Calculando el movimiento total
total1=total1+promx;
total2=total2+promy;
angt=angt+ang;

% Despliegue de resultados
fprintf ('Movimiento total: en X: %i, en Y: %i \n', total1, total2);
fprintf ('Rotacion: %i grados \n', angt);
```

**lectura.m**

```
function [I1,I2,I3] = lectura()

%lectura de archivos .png en la misma carpeta del código
I1=imread('TT1.png');
I2=imread('TT2.png');
I3=imread('TT3.png');

% conversión de las imágenes de RGB a intensidad de grises
% despliegue en ventana nueva
I1=rgb2gray(I1);
figure
imshow(I1);
I2=rgb2gray(I2);
figure
imshow(I2);
I3=rgb2gray(I3);
figure
imshow(I3);
```

**generasec.m**

```
function [num, sec] = generasec()

clear all;
num=0;
lx=0; ox=0; px=0;
ly=0; oy=0; py=0;
```

```
dxx=1000; dyy=1000; ra=0;
tip='a';

while ((num<2)||(num>10))
   num = input('\nNúmero de imágenes de la secuencia (2-10): ');
end

%first steps num=1
%dimensiones imagen
fprintf('\nProporciona las características de la imagen inicial \n');
while ((lx<50)||(lx>150)||(ly<50)||(ly>150))
   fprintf('\nDimensiones de la imagen en píxeles: (50:150),(50:150)')
   ly = input('\n  para x: ');
   lx = input('  para y: ');
end

%dimensiones objeto
while ((ox<5)||(ox>15)||(oy<5)||(oy>15))
   fprintf('Dimensiones del objeto rectangular en píxeles: (5:15),(5:15)')
   oy = input('\n  para x: ');
   ox = input('  para y: ');
end
obj = rand(ox,oy);
[lxx,lyy] = size (obj);
dx = lx-ox+1; dy = ly-oy+1;
xx = round(ox/2); yy = round(oy/2);

%localización objeto
while ((px<xx)||(px>dx)||(py<yy)||(py>dy))
   fprintf('Posición inicial del objeto (centro) en coordenadas \n');
   fprintf('  para x (%d:%d)', yy,dy);
   py = input(' ');
   fprintf('  para y (%d:%d)', xx,dx);
   px = input(' ');
end

%crear imagen inicial
sec = zeros(lx,ly,num);
for x=1:ox;
   for y=1:oy;
      sec(px+x-xx, py+y-yy, 1) = obj(x,y);
   end
end
figure;
imshow(sec(:,:,1));

for (k=2:num)
   fprintf('\nImagen subsecuente número %d', k);
   while ((tip~='D')&&(tip~='R')&&(tip~='d')&&(tip~='r'))
      tip = input('\n  Tipo de movimiento: \n  Desplazamiento (D), Rotación (R) ','s');
```

```
    end
    if ((tip=='D')||(tip=='d'))
        while ((px+dxx+ox>lx+1)||(py+dyy+oy>ly+1)||(px+dxx<xx)||(py+dyy<yy))
            dyy = input('\n  Desplazamiento en x: ');
            dxx = input('  Desplazamiento en y: ');
            pxx = px+dxx; pyy = py+dyy;
            if ((pxx+ox>lx+1)||(pyy+oy>ly+1)||(pxx<xx)||(pyy<yy))
                fprintf('ERROR: Objeto fuera de las dimensiones de la imagen \n');
                dxx=1000; dyy=1000;
            end
        end
        px = px+dxx; py = py+dyy;
        xx = round(lxx/2);
        yy = round(lyy/2);
        for x=1:lxx;
            for y=1:lyy;
                sec(px+x-xx, py+y-yy, k) = obj(x,y);
            end
        end
        figure;
        imshow(sec(:,:,k));
    else
        ra = input('\n  Ángulo de rotación deseado: ');
        obj = imrotate(obj,-ra,'bil');
        [lxx,lyy] = size(obj);
        xx = round(lxx/2);
        yy = round(lyy/2);
        for x=1:lxx;
            for y=1:lyy;
                sec(px+x-xx, py+y-yy, k) = obj(x,y);
            end
        end
        figure;
        imshow(sec(:,:,k));
    end
    tip='a';
    dxx=1000; dyy=1000;
    ra=0;
end
```

**optflow_corr.m**

Escrito y desarrollado por Piotr Dollar, como parte de la caja de herramientas de procesamiento de imágenes y video.

```
% Calculate optical flow using cross-correlation.
%
% Calculate optical flow using correlation, followed by lucas & kanade on
% aligned squares for subpixel accuracy.  Locally, the closest patch within
```

```
% some search radius is found.  The distance measure used is the euclidean
% distance between patches -- NOT normalized correlation since we assume
% pixel brightness constancy.  Once the closest matching patch is found,
% the alignment between the two patches is further refined using lucas &
% kanade to find the subpixel translation vector relating the two patches.
%
% This code has been refined for speed, but since it is nonvectorized code
% it can be fairly slow.  Running time is linear in the number of pixels
% but the constant is fairly large.  Test on small image (150x150) before
% running on anything bigger.
%
% INPUTS
%   I1, I2      - input images to calculate flow between
%   patch_r     - determines correlation patch size around each pixel
%   search_r    - search radius for corresponding patch
%   sigma       - [optional] amount to smooth by (may be 0)
%   thr         - [optional] RELATIVE reliability threshold (.01 by default)
%   show        - [optional] figure to use for display (no display if == 0)
%
% OUTPUTS
%   Vx, Vy  - x,y components of optical flow [Vx>0 -> flow is right, Vy>0 -> flow is down]
%   reliab  - reliability of optical flow in given window (cornerness of window)
%
% DATESTAMP
%   29-Sep-2005  2:00pm
%
% See also OPTFLOW_HORN, OPTFLOW_LUCASKANADE

% Piotr's Image&Video Toolbox      Version 1.01
% Written and maintained by Piotr Dollar    pdollar-at-cs.ucsd.edu
% Please email me if you find bugs, or have suggestions or questions!

function [Vx,Vy,reliab] = optflow_corr( I1, I2, patch_r, search_r, sigma, thr, show )
    if( nargin<5 || isempty(sigma)) sigma=1; end;
    if( nargin<6 || isempty(thr))  thr=0.001; end;
    if( nargin<7 || isempty(show)) show=0; end;

    %%% error check inputs
    if( ndims(I1)~=2 || ndims(I2)~=2 ) error('Only works for 2d input images.'); end
    if( any(size(I1)~=size(I2)) ) error('Input images must have same dimensions.'); end
    if( isa(I1,'uint8')) I1 = double(I1); I2 = double(I2); end;

    %%% smooth images (using the 'smooth' flag causes this to be slow)
    I1b = gauss_smooth( I1, [sigma sigma], 'smooth' );
    I2b = gauss_smooth( I2, [sigma sigma], 'smooth' );

    %%% precomputed constants
    subpixelaccuracy = 1;
    siz = size(I1);
    big_r = search_r + patch_r;
```

```
n = (2*patch_r+1)^2;
width_D = 2*search_r+1;
[ndxs,ndys] = meshgrid( 1:width_D, 1:width_D );

%%% hack to penalize more distant translations (closest are best?)
[xs,ys] = meshgrid(-search_r:search_r,-search_r:search_r);
Dpenalty = ((xs.^2 + ys.^2)/search_r^2 + 1) .^(1/20);

%%% pad I1 and I2 by search_r in each direction
I1b = padarray(I1b,[search_r search_r],0,'both');
I2b = padarray(I2b,[search_r search_r],0,'both');
sizB = size(I1b);

%%% precompute gradient for subpixel accuracy
[gEy,gEx] = gradient( I1b );

%%% loop over each window
Vx = zeros( sizB-2*big_r ); Vy = Vx;  reliab = Vx;
for r = big_r+1:sizB(1)-big_r
   for c = big_r+1:sizB(2)-big_r
      T = I1b( r-patch_r:r+patch_r, c-patch_r:c+patch_r );
      IC = I2b( r-big_r:r+big_r, c-big_r:c+big_r );

      % get smallest distance
      D = xeuc2_small( T, IC, 'valid' );
      D = (D+eps) .* Dpenalty;
      [disc, ind] = min(D(:));

      % get offset to smallest distance
      ndx = [ndys(ind(1)) ndxs(ind(1))];
      v = ndx - (width_D + 1)/2;

      % get subpixel movement using lucas kanade on rectified windows
      if( subpixelaccuracy )
         T2 = I2b( r+v(1)-patch_r:r+v(1)+patch_r, c+v(2)-patch_r:c+v(2)+patch_r );
         gEx_rc = gEx(r-patch_r:r+patch_r, c-patch_r:c+patch_r );
         gEy_rc = gEy(r-patch_r:r+patch_r, c-patch_r:c+patch_r );
         Et_rc = T2-T;
         A = [ gEx_rc(:), gEy_rc(:) ];  b = -Et_rc(:);
         AtA = A'*A;  detAtA = AtA(1)*AtA(4)-AtA(2)*AtA(3);
         if( abs(detAtA) > eps )
            invA = ([AtA(4) -AtA(2); -AtA(3) AtA(1)] / detAtA) * A'; veps = (invA * b)';
            lambdas = eig(A'*A); subrel = abs(min(lambdas)/max(lambdas));
            if( subrel > .0001 ) v = v + veps; end
         end
      end

      % get reliability
      %Dsort = sort(D(:)); rel = 1 - Dsort(1)/Dsort(2);
      x=T(:); rel = sum(x.*x)/n - (sum(x)/n)^2; % variance
```

```
        % record reliability and velocity
        reliab(r-big_r,c-big_r) = rel;
        Vx(r-big_r,c-big_r) = v(2);
        Vy(r-big_r,c-big_r) = v(1);
    end;
end;


%%% resize all to get rid of padding
I1b = arraycrop2dims( I1b, siz );
I2b = arraycrop2dims( I2b, siz );
Vx = arraycrop2dims( Vx, siz );
Vy = arraycrop2dims( Vy, siz );
reliab = arraycrop2dims( reliab, siz );

%%% scale reliab to be between [0,1]
reliab = reliab / max([reliab(:); eps]);
Vx(reliab<thr) = 0;  Vy(reliab<thr) = 0;

%%% show quiver plot on top of reliab
if( show )
    reliab( reliab>1 ) = 1;
    figure(show); show=show+1;  clf; im( I1 );
    hold('on'); quiver( Vx, Vy, 0,'-b' ); hold('off');
    %figure(show); show=show+1;  clf; im( I2 );
    %figure(show); show=show+1;  clf; im( reliab );
end

% since the convolutions are so small just call conv2 everywhere
% see xeucn.m for more general version
function C = xeuc2_small( B, I, shape )
    sizB = size(B);
    B = rot90( B,2 );
    I_mag = localsum( I.*I, sizB, shape );
    B_mag = B.^2;  B_mag = sum( B_mag(:) );
    C = I_mag + B_mag - 2 * conv2(I,B,shape);
```

**tipomov.m**

```
function [promx, promy, ang] = tipomov(Vx, Vy)

% Cálculo de diferencias entre vectores
Vxx = round(Vx);
Vyy = round(Vy);
Vxx = Vx - Vxx;
Vyy = Vy - Vyy;

% Cálculo de desplamientos lineales o ángulos de rotación, según sea el caso
if ((Vxx<=0.01)&(Vyy<=0.01))
```

```
   [promx, promy] = desplaza( Vx, Vy );
   ang=0;
else
   [ang] = rotad( Vx, Vy );
   promx=0; promy=0;
end
```

**desplaza.m**

```
function [promx,promy] = desplaza( Vx, Vy )
promx=0; promy=0;

% Búsqueda de índices en Vx, medición del vector indx, suma del vector indx
[i,j,indx]=find(Vx);
lx=length(indx);
indx=sum(indx);

% Cálculo y despliegue de la magnitud del desplazamiento en 'x'
if (indx==0)
   fprintf ('Desplazamiento detectado en X: 0 ');
else
   promx=indx/lx;
   fprintf ('Desplazamiento detectado en X: %i ',promx);
end
% Búsqueda de índices en Vy, medición del vector indy, suma del vector indy
[k,l,indy]=find(Vy);
ly=length(indy);
indy=sum(indy);

% Cálculo y despliegue de la magnitud del desplazamiento en 'y'
if (indy==0)
   fprintf ('Desplazamiento detectado en Y: 0 \n');
else
   promy=indy/ly;
   fprintf ('Desplazamiento detectado en Y: %i \n',promy);
end
```

**rotad.m**

```
function [ang] = rotad( Vx, Vy )
ang=0; ds=0;

% Búsqueda de índices de Vx, cálculo de los límites de la matriz
[i,j]=find(Vx);
d1=min(i);
d2=max(i);
d3=min(j);
d4=max(j);
```

```
% Obtención de coordenadas del centro del campo de flujo
dy=round(d1+((d2-d1)/2));
dx=round(d3+((d4-d3)/2));
ds=round((d4-d3)/10);
sum1=0; sum2=0; ci=0; cd=0;

% Obtención de información para determinar la dirección de cálculo de la
% pendiente
for y=d3:dx
   sum1=sum1+abs(Vx(dy,y));
   sum2=sum2+abs(Vy(dy,y));
end
aa=sum1+sum2;
sum1=0; sum2=0;
for y=dx:d4
   sum1=sum1+abs(Vx(dy,y));
   sum2=sum2+abs(Vy(dy,y));
end
bb=sum1+sum2;

% Toma de decisión sobre la dirección de cálculo de flujo, obtención del
% valor de los ángulos
if (aa>bb)
   x=round(dx-ds);
   for y=x:-1:d3
      a=dy+Vy(dy,y);
      b=y+Vx(dy,y);
      angs=atan((dy-a)/(dx-b))*180/pi;
      ang=ang+angs;
      ci=ci+1;
   end
else
   x=round(dx+ds);
   for y=x:d4
      a=dy+Vy(dy,y);
      b=y+Vx(dy,y);
      angs=atan((dy-a)/(dx-b))*180/pi;
      ang=ang+angs;
      cd=cd+1;
   end
end

% Cálculo del ángulo promedio y despliegue
if ci==0
   ang=ang/cd;
else
   ang=ang/ci;
end
fprintf ('Rotacion detectada de: %i grados \n', ang);
```