PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DETECCIÓN DE PERSONAS Y VEHÍCULOS EN IMÁGENES TOMADAS DESDE UN UAV

Tesis para optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Jhon Jhojan Rivera Vicente

ASESOR: Pedro Crisóstomo Romero

Lima, Diciembre del 2015

ANEXOS

ANEXO 1

CODIGO DESARROLLADO EN OPENCV

Este programa elaborado en OpenCV, se encarga de generar el alineamiento de todos los fotogramas de una secuencia, los cuales se tomarán dependiendo de la tasa de muestro que defina en el parámetro Separacion_fotogramas

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include "opencv2/stitching/stitcher.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <cv.h>
using namespace cv;
using namespace std;
RNG rng(12345);

static void help(char** argv)
{
cout << "\n Con este programa hecho en lenguaje C, mediante la librería de OpenCV, se generará las imágenes resultantes luego de alinear y restar los fotogramas. Este programa debe estar en la misma carpeta donde se ubican las imágenes obtenidas de la extracción de fotogramas hecho en matlab.\n"
    << "Usage: " << argv[0] << " <image_mask> (example mask: example_%02d.jpg)\n"
    << "Image mask define el nombre de las imágenes que se van a procesar y van a ser leídos como una secuencia. \n"
    << "Usando the mask example_%02d.jpg leerá imágenes etiquetadas como 'example_00.jpg', 'example_01.jpg', etc."
    << endl;
}

int main(int argc, char** argv)
{
  char file_name[100],file_output[100];
  int aux=1,aux2=0;
  int j=1;
  Mat gray_image1, gray_image2, resta, rest, umbralizado, element, opening, rest2;

if(argc != 2)
  {
      help(argv);
      return 1;
  }
```

```
//******************Inicio de lectura de la secuencia de imágenes almacenadas******************//

String first_file = argv[1];
VideoCapture sequence(first_file);

if (!sequence.isOpened())
   {
     cerr << "Error al abrir secuencia!\n" << endl;
     return 1;
   }
Mat image;                              // Se crea la matriz que almacenará el fotograma
Int Separación_fotogramas;              // Indica la tasa de muestreo de fotogramas
Sequence >> image;                      // Se lee el primer fotograma seleccionado

for (;;)                                // Inicia la lectura de imágenes y proceso de alineamiento
{
        std::cout<< "# de frame: \t"<< aux <<std::endl;
        if (image.empty())
        {
                cout << "fin de secuencia" << endl;
                break;
        }
        cvtColor( image, gray_image1, CV_RGB2GRAY);         // Primer fotograma a procesar
        std::cout<< "Numero de fotograma a trabajar(fotograma previo): \t"<< aux <<std::endl;
        for (i=1;i<=Separacion_fotogramas;++i)
        {
                sequence >> image;
                aux=aux+1;
        }
        if (image.empty())
        {
                cout << "fin de secuencia" << endl;
                break;
        }
        cvtColor( image, gray_image2, CV_RGB2GRAY);         //segundo fotograma a procesar
        std::cout<< "Numero de fotograma a trabajar(fotograma actual): \t"<< aux <<std::endl;
        if (gray_image1.empty() || gray_image2.empty())         //Verifica si aún hay fotogramas
        {
                cout << "fin de secuencia" << endl;
                break;
        }

//**********PASO 1: Detección de los puntos característicos, mediante SURF Detector********//
int minHessian = 600;
SurfFeatureDetector detector( minHessian );
printf("Se aplicó SURF\n" );
std::vector< KeyPoint > keypoints_object, keypoints_scene;      //creación de vectores
detector.detect( gray_image1, keypoints_object );
detector.detect( gray_image2, keypoints_scene );


//*************************Dibujamos los puntos característicos (Opcional) **************************//
//Mat img_keypoints_1;
//Mat img_keypoints_2;
//drawKeypoints(gray_image1, keypoints_object, img_keypoints_1, Scalar(0,255,0,0),
                        DrawMatchesFlags::DEFAULT);
```

```
//drawKeypoints(gray_image2, keypoints_scene, img_keypoints_2, Scalar::all(-1),
                        DrawMatchesFlags::DEFAULT);
//********************* PASO 2: Calculo de los descriptores (Feature vectors)*******************//
SurfDescriptorExtractor extractor;
Mat descriptors_object, descriptors_scene;
extractor.compute( gray_image1, keypoints_object, descriptors_object );
extractor.compute( gray_image2, keypoints_scene, descriptors_scene );
printf("se calcularon los extractores\n" );

//************ PASO 3: Emparejamiento de los descriptores de ambos fotogramas*************//
FlannBasedMatcher matcher;
std::vector< DMatch > matches;
matcher.match( descriptors_object, descriptors_scene, matches );
double max_dist = 0; double min_dist = 100;
std::cout<< "número total emparejamientos \t"<< matches.size() << std::endl;

//******************** Distancias entre los puntos característicos***********************************//
for( int i = 0; i < descriptors_object.rows; i++ )
{       double dist = matches[i].distance;
        if( dist < min_dist ) min_dist = dist;
        if( dist > max_dist ) max_dist = dist;
}
        printf("-- Max dist : %f \n", max_dist );
        printf("-- Min dist : %f \n", min_dist );


//**Usar solo las distancias que sean menores a 3*min_dist, para determinar los correctos
emparejamientos*****//
std::vector< DMatch > good_matches;
for( int i = 0; i < descriptors_object.rows; i++ )
{       if( matches[i].distance < 3*min_dist )
        {       good_matches.push_back( matches[i]); }
        }
        Mat img_matches;

//************************Dibujamos los emparejamientos (Opcional) ******************************//
drawMatches(gray_image1,keypoints_object,gray_image2,keypoints_scene,good_matches,
        img_matches,Scalar::all(-1),Scalar::al (-1),vector<char>(),
            DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

//*************************************************************************************************************//

std::vector< Point2f > obj;
std::vector< Point2f > scene;
for( int i = 0; i < good_matches.size(); i++ )
{
                                //** obtener puntos caracterisctios de emparejamientos correctos
        obj.push_back( keypoints_object[ good_matches[i].queryIdx ].pt );
        scene.push_back( keypoints_scene[ good_matches[i].trainIdx ].pt );
}
printf("Buenos emparejamientos\n" );
std::cout << "good matches\t" <<good_matches.size()<<std::endl;

//****** PASO 4: Calculo de la matriz de Homografía, entre los fotogramas previo y actual*****//
```

```
Mat H = findHomography( obj, scene, CV_RANSAC );

//*********PASO 5: Uso de la matriz de Homografía, para alinear ambos fotogramas***********//
cv::Mat result;
warpPerspective(gray_image1,result,H,gray_image2.size());      // Transformación de fotograma
subtract(gray_image2,result,rest);
subtract(result,gray_image2,rest2);
rest=rest+rest2;
imwrite("final.bmp",rest);
imshow("Resta entre el fotograma previo transformado y el fotograma actual",rest);

//*************PASO 6: Imprimimos los resultados de todos los pares de fotogramas*************//
// Se almacenan todos los resultados de los alineamientos entre todos los pares de fotogramas
//en una carpeta.
snprintf(file_output,80,"/home/jhon/Desktop/OpenCV/opencv-2.4.9/samples/cpp/
        Nombre_carpeta/Name_%03d.bmp", j);
imwrite(file_output,rest);
j=j+1;


}
waitKey(0);
return 0;
}
```

// Fin del programa que se encarga de generar y almacenar las imágenes resultantes del alineamiento y resta entre cada transformada de fotograma previo y fotograma actual.

ANEXO 2

CODIGO DESARROLLADO EN MATLAB

Primera etapa de detección y seguimiento de blobs

Este programa permite detectar todos los blobs que se encuentran en las imágenes resultantes del alineamiento y resta de los fotogramas previos y actuales. Posteriormente se procede a almacenar todas las posiciones de todos los blobs.

```
% Se procede a extraer todas las imágenes resultantes almacenadas en el fichero
file=dir('C:\Users\Jhon\drive3\Nombre_carpeta\*.bmp');
imagenes=imageSet('drive3\Nombre_carpeta\');
filtro=fspecial('gaussian',5,1.5);
se=strel('disk',1);
a=0; contador=0;
aux=0;                          %permite que el los frames diferentes al primero se lean
frameini(1).x=0;
frameini(1).y=0;
frameini(1).idx=0;
frameni(1).Area=0;
DistanciaObjetos=30;            % Distancia máxima entre centroides, definido por nosotros
dist=360;                       %inicializamos la distancia entre 2 centroides

% se define una estructura que posee tres campos, la cual almacenará las características de
%los blobs detectados, además “g”, representa el parámetro de número de fotogramas,
%indicándonos en que fotograma se encuentra el blob; e “inicio” representa el número de blobs
%que se espera encontrar.
% se procede a colocar valores de ceros ’0 ’, de tal manera que tenga un valor inicial.
blob=struct('Coordenadas',struct('x',0,'y',0,'area',0));
for inicio=1:300
        for g=1:1000
        blob(inicio).Coordenadas(g).x=0;
        blob(inicio).Coordenadas(g).y=0;
        blob(inicio).Coordenadas(g).area=0;
        end
end
%  Se procede a leer todas las imágenes contenidas en el fichero
%%%%%%%%%%%%%%%% SEGMENTACIÓN DE IMAGEN %%%%%%%%%%%%%%%%%
for i=1:length(file)
        ima=read(imagenes,i);
        ima=imfilter(ima,filtro);
        imshow(ima);
        im=im2bw(ima,0.08);             %Umbral para vehículos 0.08, personas 0.05
        [alto, ancho]=size(im);         % Se  extrae las dimensiones de la imagen
        im=imclearborder(im);           % Se inicia el uso de operaciones morfológicas
        BW=bwmorph(im,'clean');
        BWb=bwmorph(BW,'bridge',7);
        BWfill=bwmorph(BWb,'fill');
        Barea=bwareaopen(BWfill,30);
        Bf=imfill(Barea,'holes');
        distancia=bwdist(Bf) <=1;
        bds=bwareaopen(distancia,300);
```

```
        bds=imclose(bds,se);
        mascara=zeros(alto,ancho);              %Crea una matriz con dimensiones extraídas

        % Se procede a contar todos los blobs de una imagen resultante
        s = regionprops(bds,'Centroid','BoundingBox','Area');
        bds=bwlabel(bds);
        numobj=numel(s);
        % se hace uso de los criterios de eliminación de los blobs que no cumplan con lo
        %acordado en el método y solo se quedan los blobs que pasen dichos criterios y se
        %almacenan en una matriz.
        for k=1:numobj
                arearect= s(k).BoundingBox(3)* s(k).BoundingBox(4);
                relacion=s(k).Area/arearect;

                if( s(k).Area>350 && ~(s(k).BoundingBox(3)>3.5* s(k).BoundingBox(4)) &&
                ~(s(k).BoundingBox(4)>3.5* s(k).BoundingBox(3)))
                        mascara=mascara+(bds==k);
                end
        end
% Algunas operaciones morfológicas
mascara=bwdist(mascara) <=1;
mascara=bwmorph(mascara,'bridge',3);
mascara=bwmorph(mascara,'fill');
mascara=imfill(mascara,'holes');
mascara=imerode(mascara,se);
% se crea las imágenes que contienen solo blobs que pasaron los criterios de %eliminación.
fileoutput=sprintf('to%03d.bmp',i);
imwrite(mascara,fileoutput);

%%%%%%%%%%%%%%% SEGUIMIENTO DE BLOBS %%%%%%%%%%%%%%%%%%%

% Se inicia analizando la primera imagen resultante obtenida después de la etapa de
%segmentación de imagen.
if(contador<1)
        r=regionprops(mascara,'Centroid','Area','BoundingBox');
        region=bwlabel(mascara);
        numBlob1=numel(r);
        for h=1:numBlob1
                frameini(h).x=r(h).Centroid(2);
                frameini(h).y=r(h).Centroid(1);
                frameini(h).idx=h;
                frameini(h).area=r(h).Area;
                blob(h).Coordenadas(1).x=r(h).Centroid(2);
                blob(h).Coordenadas(1).y=r(h).Centroid(1);
                blob(h).Coordenadas(1).area=r(h).Area;
        end
        contador=contador+1;                    % Solo se una vez y es para la primera imagen
        previo=frameini;
        numPuntos=numBlob1;
        numero_objetos=numBlob1;        % Indicador de numero de blobs en la secuencia
        f=2;
        b=2;
end
% Se inicia analizando las distancias entre pares de imágenes resultantes
if(aux>0)
r=regionprops(mascara,'Centroid','Area','BoundingBox');
region=bwlabel(mascara);
numBlob2=numel(r);
        for j=1:numBlob2
        framesec(j).x=r(j).Centroid(2);
```

```
            framesec(j).y=r(j).Centroid(1);
            framesec(j).idx=j;
            framesec(j).area=r(j).Area;
            end
% Se inicia el cálculo de distancias entre blobs de pares de imágenes
b=1;
for Operaciones=1:numBlob2              % Blobs de la segunda imagen (Imagen posterior)
   for numDist=1:numPuntos              % Blobs de la primera imagen (imagen inicial)
    dist_calculada=((frameini(numDist).y-framesec(Operaciones).y)^2 +(frameini(numDist).x-
    framesec(Operaciones).x)^2)^0.5;

          if(dist_calculada<=dist && dist_calculada<= DistanciaObjetos)
             dist=dist_calculada;
             tx=frameini(numDist).x;
             ty=frameini(numDist).y;
             m=numel(blob);

             for objetos=1:numero_objetos
                for cuadros=1:f
                   if(blob(objetos).Coordenadas(cuadros).x == tx)
                   blob(objetos).Coordenadas(cuadros+1).x=framesec(Operaciones).x;
                   blob(objetos).Coordenadas(cuadros+1).y=framesec(Operaciones).y;
                   blob(objetos).Coordenadas(cuadros+1).area=framesec(Operaciones).area;
                   end
                end
             end
          end
          % se termino de calcular las distancias de un blob con los blobs de otra imagen
          if(numDist==numPuntos)
             if(dist>=32)

          blob(numero_objetos+1).Coordenadas(f).x=framesec(Operaciones).x;   % Coordenadas
          del centroide del nuevo BLOB
                blob(numero_objetos+1).Coordenadas(f).y=framesec(Operaciones).y;
                blob(numero_objetos+1).Coordenadas(f).area=framesec(Operaciones).area;
                numero_objetos=numero_objetos+1;            % indicador de que se CREO y
ALMACENO un nuevo BLOB
             end
          end
   end
dist=360;         %Se inicializa el valor de dist con 360, pues ya se terminó de analizar el blob
                  %de una imagen con los blob de la siguiente imagen
   a=0;
end
f=f+1;              %nos permite cargar los valores al paso de la 3era imagen (imagen posterior
                     %de imagen 2 )
frameini=framesec;       %Recien ahora podemos trabajar con imagen inicial y posterior
numPuntos=numBlob2;     %Para que los parametros pasen de la imagen posterior y se
                         %carguen a la imagen inicial,
end
aux=1;

end

% Una vez realizado este proceso para todas las imágenes resultantes obtenidas de la
%segmentación de imagen, se generará una variable conteniendo todos los blobs detectados a
%lo largo de la secuencia de imágenes.

nu=numero_objetos;
```

Segunda etapa de detección y seguimiento de blobs

En esta etapa eliminaremos aquellas posiciones iguales a cero “0”, puesto que estas coordenadas solo indica que el blob detectado no se encontraba presente en la imagen.

```
% Creamos una nueva estructura donde se almacenarán las coordenadas distintas de cero.

bl=struct('autos',struct('x',{},'y',{},'frame',{}, 'area',{}));
sec=1;
numero_objetos=nu;
% eliminaremos aquellas coordenadas iguales a 0
for b=1:numero_objetos
   no_mas=2; a=1;frame=1;
   while(frame<=f && no_mas~=1)
         if(no_mas>1)
           if(blob(b).Coordenadas(frame).x ~=0)
                  bl(sec).autos(a).x=blob(b).Coordenadas(frame).x;
                  bl(sec).autos(a).y=blob(b).Coordenadas(frame).y;
                  bl(sec).autos(a).area=blob(b).Coordenadas(frame).area;
                  bl(sec).autos(a).frame=frame;
                  a=a+1;
                  if(blob(b).Coordenadas(frame+1).x==0)
                        no_mas=1;
                  end
            end
          end
          frame=frame+1;
   end
   sec=sec+1;
end
```

Tercera etapa de detección y seguimiento de blobs

Esta etapa se encarga de contar los blobs detectados, los cuales se encuentran presentes más de 3 veces en la secuencia de imagen, ya que se estimó que un objeto detectado no puede aparecer en un fotograma, desparecer en otro y finalmente aparecer en un siguiente fotograma. Con esto logramos obtener el número de vehículos y personas moviéndose en una secuencia de imágenes tomadas desde un UAV(cámara en movimiento)

```
objetos_finales=0;
%contar los autos
sec=sec-1;  % disminuimos en uno, puesto cuando sale del FOR anterior se le aumentó

for obj=1:sec
    aa=length(bl(obj).autos);
    if(aa>3)
        objetos_finales=objetos_finales+1;
        mayor=bl(obj).autos(1).area;

    end
end
```