Visión por computador (2016-2017)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 1

Ignacio Martín Requena

21 de octubre de 2016

Índice

Índice de figuras

1. Apartado A

1.1. Enunciado

Implementar una función de convolución (ejemplo, void my_imGaussConvol(Mat& im, Mat& maskCovol, Mat& out)) debe ser capaz de calcular la convolución 2D de una imagen con una máscara.
Ahora supondremos que la máscara es extraída del muestreo de una Gaussiana 2D simétrica. Para ello implementaremos haciendo uso de las siguientes funciones auxiliares (2 puntos):

1).- Calculo del vector máscara: Usar la función getGausssianKernel() . Verificar el resultado mostrando los vectores obtenidos 2).- Calcular la convolución de una imagen con una máscara gaussiana, usar filter2d(). Verificar el resultado mostrando resultado sobre distintas imágenes con distintos valores de sigma.

1.2. Comentarios sobre el desarrollo

En primer lugar leemos la imagen a la cual le calcularemos la convolución:

```
/********************** Lectura y declaracion de las imagenes usadas**********************/
Mat im = imread(imageName_freq_bajas.c_str(),IMREAD_COLOR);
Mat im2 = imread(imageName_freq_altas.c_str(),IMREAD_COLOR);//usada para las frecuencias bajas
if( im.empty() || im2.empty()){
    cout << "error de lectura"<< endl;
}
```

Figura 1.1: Lectura de la imagen

Una vez cargada la imagen procedemos a calcular su convolución 2D. El procedimiento seguido ha sido:

- **Calculamos la máscara para las frecuencias bajas.**

  Con la función *getGaussianKernel* que nos proporciona OpenCv determinamos los valores de sigma y delta. En nuestro caso el valor de sigma viene determinado por un iterador que va aumentando en 20 unidades a fin de observar posteriormente los cambios de la imagen en función del parámetro sigma

- **Hacemos una llamada a la función my_imGaussConvol.**

  Esta función recibe como parámetros de entrada una imagen, una máscara calculada previamente y una imagen de salida donde se almacenará la imagen convolucionada.

  Los pasos para calcular la imagen convolucionada han sido:

  - En primer lugar damos la vuelta a la máscara con la función *flip()*. Realmente este paso para nuestra máscara no sería necesario ya que al ser gaussiana esta es simétrica.

- Con la función *filter2D* de OpenCV realizamos el filtado gaussiano de la imagen original con la máscara gaussiana.
- filter2D hace el filtrado en una única componente (en este caso la componente Y), por tanto debemos realizar un filtrado también en el sentido del eje X. Para ello trasponemos la imagen con el primer filtro pasado y volvemos a realizarle un segundo filtro igual al anterior
- Por último volvemos a trasponer la imagen final para que quede bien orientada

- Acabamos mostrando la imagen por pantalla en cada iteración de nuestro bucle for

1.3. Salidas obtenidas

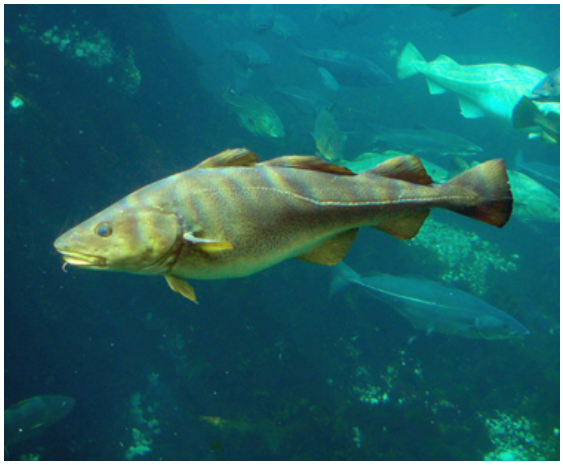

Figura 1.2: Imagen con sigma = 1

Figura 1.3: Imagen con sigma = 21

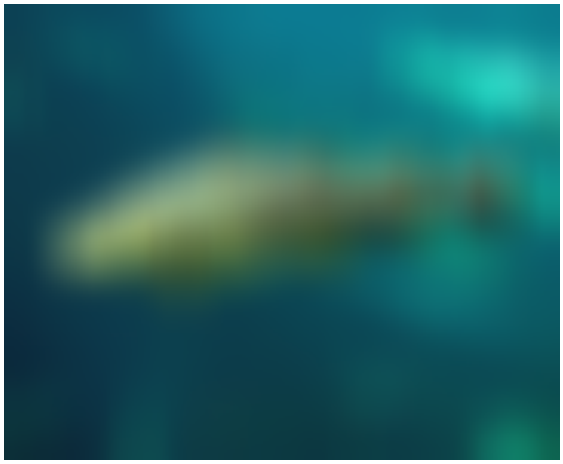

Figura 1.4: Imagen con sigma = 41

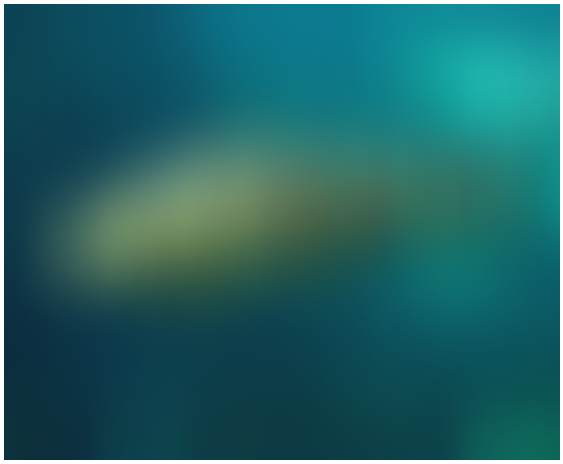

Figura 1.5: Imagen con sigma = 61

Como podemos ver a mayores valores de sigma mayor número de frecuencias altas desechamos. Esto es debido principalmente a que con valores de sigma altos lo que hacemos es aumentar el tamaño de la máscara s y por tanto tengamos en cuenta un mayor numero de vecinos a la hora de hacer el filtro gaussiano.

2. Apartado B

2.1. Enunciado

Mezclando adecuadamente una parte de las frecuencias altas de una imagen con una parte de las frecuencias bajas de otra imagen, obtenemos una imagen híbrida que admite distintas interpretaciones a distintas distancias (ver hybrid images project page).

Para seleccionar la parte de frecuencias altas y bajas que nos quedamos de cada una de las imágenes usaremos el parámetro sigma del núcleo/máscara de alisamiento gaussiano que usaremos. A mayor valor de sigma mayor eliminación de altas frecuencias en la imagen convolucionada. Para una buena implementación elegir dicho valor de forma separada para cada una de las dos imágenes (ver las recomendaciones dadas en el paper de Oliva et al.). Recordar que las máscaras 1D siempre deben tener de longitud un número impar.

Usar la convolución que hemos implementado en el apartado A para elegir los sigmas más adecuados para la selección de frecuencias en parejas de imágenes (ver fichero de datos).
1. Implementar una función que genere las imágenes de baja y alta frecuencia.
2. Escribir una función para mostrar las tres imágenes (alta, baja e híbrida) en una misma ventana. (Recordar que las imágenes después de una convolución contienen número flotantes que pueden ser positivos y negativos)

2.2. Comentarios sobre el desarrollo

Una imagen híbrida es la combinación de dos imágenes, una que posee mayormente frecuencias bajas y otra que posee mayormente frecuencias altas.

La característica más notable de este tipo de imágenes es que a tamaños de imagen grandes y observándola relativamente cerca podemos percibir fácilmente las frecuencias altas, mientras que con tamaños de imagen pequeños o viéndola desde la lejanía nuestro ojo percibe prácticamente sólo frecuencias bajas. Por tanto, una vez obtenida la imagen híbrida y ajustados los valores de sigma de cada una de las dos imágenes deberíamos experimentar este efecto.

El cálculo de la imagen hibrida se ha realizado siguiendo los siguientes pasos:

- En primer lugar, a partir de la imagen que usaremos para las frecuencias bajas calculamos su convolución aplicando un kernel gaussiano (ver apartado A).

```
//Calculo frecuencias baja
mask_bajas = getGaussianKernel(sigma_bajas,6*sigma_bajas + 1,CV_32F); //kernel para imagen de frecuencias bajas
my_imGaussConvol(im, mask_bajas, freq_bajas); //calculo frecuencias bajas para imagen de frecuencias altas
```

Figura 2.1: Código frecuencias bajas

El valor de sigma en este caso depende de la imagen de entrada, por lo que se ha ajustado para cada una de las imágenes posibles un valor de sigma correcto para poder percibir el efecto explicado anteriormente.

- A continuación calculamos las frecuencias altas de una imagen.

  Para ello en primer lugar calculamos su filtro gaussiano y una vez obtenido restamos a la imagen original los valores de frecuencia bajas obtenidos en el filtro gaussiano:

```
/*Calculo de las frecuencias altas*/
void calculo_freq_altas(Mat &im, Mat &freq_bajas, Mat &out){
    out = (im/2 - freq_bajas/2)+127;

}
```

Figura 2.2: Cálculo frecuencias altas

Como vemos, el calculo es una simple resta de cada valor de la imagen original con la de frecuencias altas y su posterior normalización para que los valores finales estén dentro del rango [0-254]

- Una vez calculada la imagen de alta frecuencia tenemos lo necesario para obtener nuestra imagen hibrida de la siguiente forma:

```
/*Ajuste de las frecuencias de la imagen hibrida*/
void calculo_img_hibrida(Mat &altas, Mat &bajas, Mat & hibrida){
    hibrida = (altas + bajas)/2;
}
```

Figura 2.3: Cálculo frecuencias altas

Cuya operación no es más que una suma de los valores calculados de la imagen de alta y la de baja frecuencia y su posterior nacionalización

- Para terminar concatenamos las imágenes de alta y baja frecuencia junto con la híbrida con la función *hconcat* de OpenCV.

2.3. Salidas obtenidas

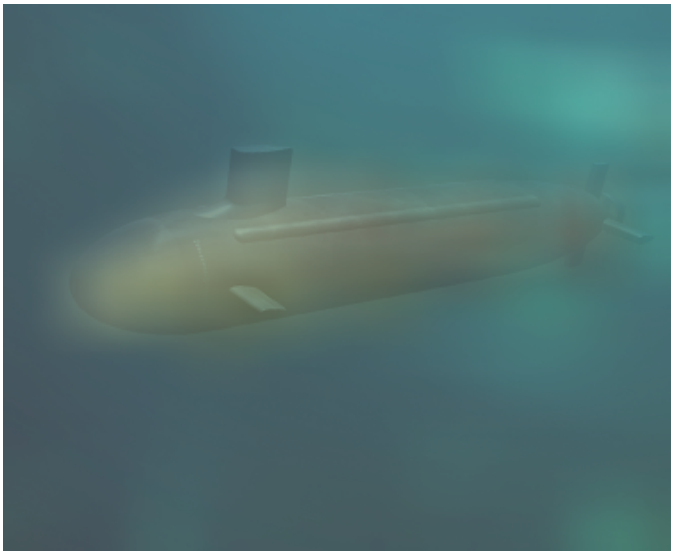

Figura 2.4: Imagen frecuencias bajas

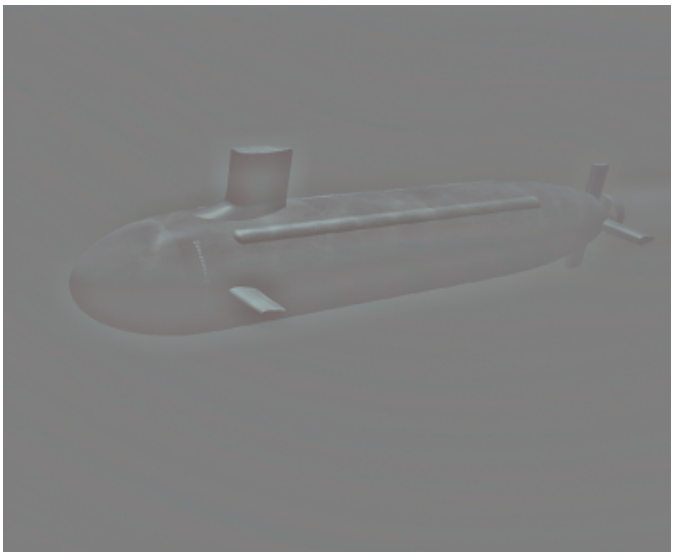

Figura 2.5: Imagen frecuencias altas

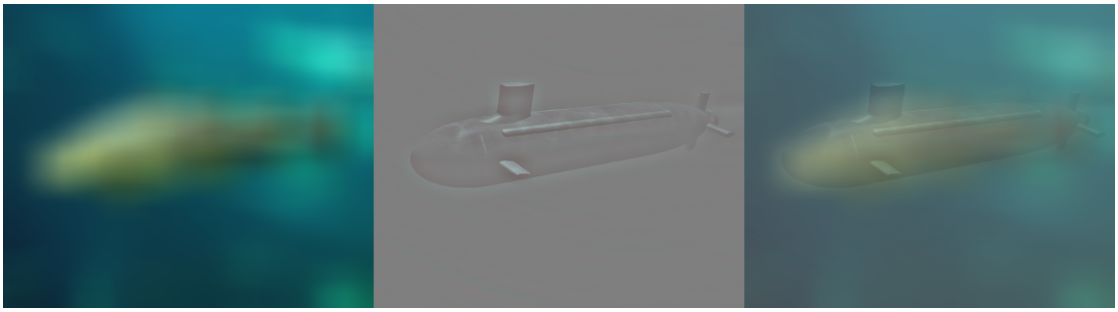

Figura 2.6: Concatenación de la imagen de frecuencias bajas, altas e imagen híbrida

Como podemos observar la imagen híbrida cumple la propiedad comentada al inicio del apartado

3. Apartado C

3.1. Enunciado

Construir una pirámide Gaussiana de al menos 5 niveles con las imágenes híbridas calculadas en el apartado anterior. Mostrar los distintos niveles de la pirámide en un único canvas e interpretar el resultado.

3.2. Comentarios sobre el desarrollo

La representación en forma de pirámide de una imagen híbrida nos da una idea de si los valores de sigma elegidos en el calculo de las imágenes de alta y baja frecuencia son los adecuados o no sin necesidad de estar continuamente alejándose y acercándose del monitor para comprobarlo.

Esta imagen piramidal se compone un número determinado de niveles. En cada nivel se aloja una imagen híbrida resultado de reducir su tamaño a la mitad de la que le precede, así que lo lógico es calcular todas las reducciones de las imágenes según el número de niveles. Esto se ha hecho de la siguiente forma:

```
for(int i=0; i < niveles; i++){
    pyrDown(img_reducidas[i], aux2, Size(aux2.cols / 2, aux2.rows / 2));
    img_reducidas.push_back(aux2);
}
```

Figura 3.1: Cálculo de la reducción del tamaño de la imagen híbrida

En un vector de tipo Mat vamos almacenando la reducción a la mitad del tamaño de la imagen anterior tantas veces como número de niveles deseemos en nuestra pirámide.
Una vez calculadas estas imágenes debemos ajustar el tamaño de las mismas para que "cuadre "con el resto de imágenes ya que es imposible concatenar imágenes que tienen diferente número de filas (para concatenaciones en horizontal) o columnas (para concatenaciones en vertical). Para solventar este problema hacemos lo siguiente:

- A los niveles que difieren en número de columnas con respecto a la imagen de nivel 1 (consideramos la imagen original el nivel 0 de la pirámide) les añadimos tantas columnas como le falten para que tenga las mismas que la de nivel 1 y rellenamos estas filas añadidas de negro:

```
Mat img_negra, nivel_completo;
vector<Mat> reducidas_con_negro;
for(uint i=2; i<img_reducidas.size(); i++){ //añado los trozos en negro para que los escalones tengan la misma anchura que la imagen de primer nivel
    Mat img_negra(img_reducidas[i].rows, img_reducidas[1].cols - img_reducidas[i].cols, img_reducidas[i-1].type());
    img_negra = Scalar(0);
    hconcat(img_reducidas[i], img_negra, nivel_completo);
    reducidas_con_negro.push_back(nivel_completo);
}
```

Figura 3.2: Ajuste tamaño de los niveles 2 a 5

- Concatenamos los niveles de todas las imágenes una vez generadas y ajustadas:

```
Mat piramide = reducidas_con_negro.back();
reducidas_con_negro.pop_back();

//concateno todos los trozos de imagen con negro
while(!reducidas_con_negro.empty()){
    Mat ultimo = reducidas_con_negro.back();
    reducidas_con_negro.pop_back();
    vconcat(ultimo, piramide, piramide);
}

vconcat(img_reducidas[1], piramide, piramide); //añado la del nivel uno ya que no hace falta ajustar sus dimensiones
```

Figura 3.3: Concatenación vertical de los niveles 1-5

- Por último, ajustamos el número de filas de la imagen concatenada en el punto anterior para que tenga el mismo tamaño que el de la original y concatenamos con esta:

```
//Reajuste de las dimensiones de la piramide para concatenarla con la original
Mat nivel_entero_negr(img_reducidas[0].rows-piramide.rows, img_reducidas[1].cols, img_reducidas[1].type());
nivel_entero_negr = Scalar(0);
vconcat(piramide, nivel_entero_negr,piramide);
hconcat(img_reducidas[0], piramide, piramide);
```

Figura 3.4: Concatenación original con reducidas

3.3. Salidas obtenidas

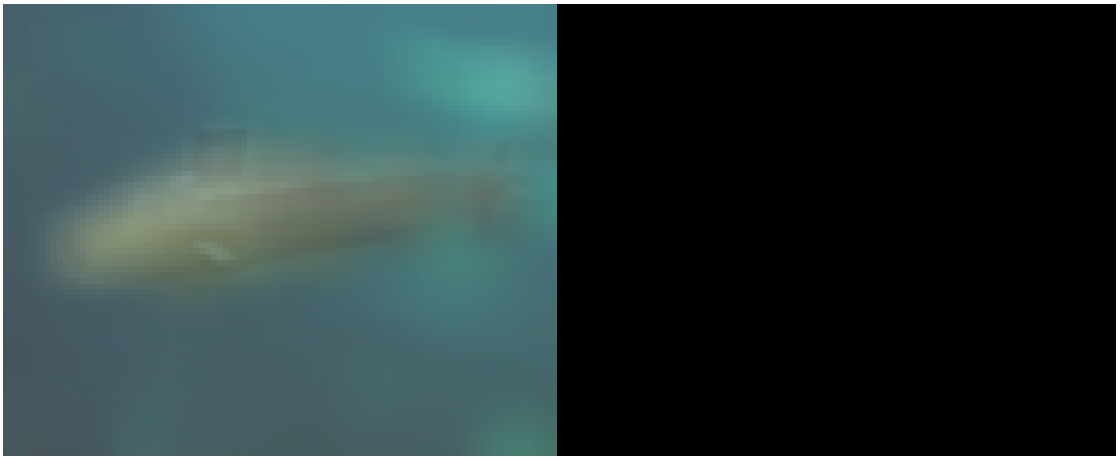

Figura 3.5: Nivel 2 de la pirámide con el ajuste de dimensión

Figura 3.6: Nivel 3 de la pirámide con el ajuste de dimensión

Figura 3.7: Nivel 4 de la pirámide con el ajuste de dimensión

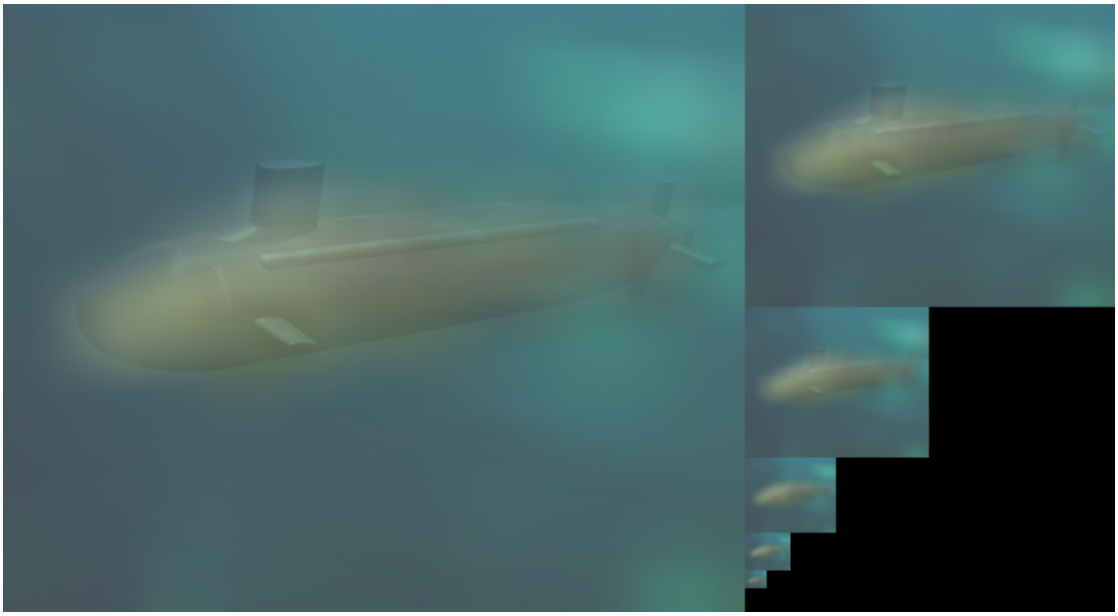

Figura 3.8: Resultado final de la pirámide

Como podemos comprobar, por debajo del nivel 2 vemos mayormente un pez (imagen con bajas frecuencias) y en los niveles 1 y 2 observamos mejor el submarino (imagen con frecuencias altas).