Captura y Compresión de Vídeo de alta Definición con DSP

Queremos optar el Google Lunar X Price y por tanto debemos enviar vídeo de alta definición (**1920x1080 pixeles y 30fps**) en tiempo real desde un robot lunar hasta la nave lunar que lo reenviará a la tierra. Hay un enlace WIFI entre el robot y la nave con un ancho de banda ideal de 11 Mbps, 1'38 MBps. Por lo tanto es necesario realizar en el robot la captura y la compresión del vídeo, para solventar el cuello de botella que supone la comunicación WIFI (poco ancho de banda).

En nuestro caso no existirá una FPGA que controle el CCD (algo que es típico), sino que el mismo DSP debe realizar el barrido de lectura analógica de la matriz del CCD y además realizar la compresión.

Realizar una estima de las prestaciones que necesitamos para poder realizar:

1. **El barrido de lectura analógica del CCD (velocidad de los conversores A/D).**
2. **Manejar imágenes tamaño HD (tamaño de la memoria).**
3. **Coste computacional asociado a la DCT (MACs).** Nos olvidamos del resto de etapas (enteras) que tiene la compresión.
4. **Decidir el tipo de aritmética que nos interesa.**

Buscar un DSP comercial que nos permita desarrollar estas dos tareas. Si fuese necesario utilizar dispositivos externos, como conversores A/D y memoria, documentar el modo de interconexión.

Notas:

- Se habilita un hilo para dudas sobre esta práctica.
- No exigiremos que el DSP esté calificado para espacio.

Se adjunta el código fuente de la DCT para bloques de 8x8 píxeles.

```
/*************************************************************************
 *    Name:         Dct
 *    Description: Does dct on an 8x8 block, does zigzag-scanning of
 *        coefficients
 *    Input:      64 pixels in a 1D array
 *    Returns:    64 coefficients in a 1D array
 *    Side effects:
 *    Date: 930128      Author: Robert.Danielsen@nta.no
 *
 *************************************************************************/
int Dct( int *block, int *coeff)
{
  int        j1, i, j, k;
  float      b[8];
  float        b1[8];
  float        d[8][8];
  float f0=(float).7071068;
  float f1=(float).4903926;
  float f2=(float).4619398;
  float f3=(float).4157348;
  float f4=(float).3535534;
  float f5=(float).2777851;
  float f6=(float).1913417;
  float f7=(float).0975452;

  for (i = 0, k = 0; i < 8; i++, k += 8) {
```

```
    for (j = 0; j < 8; j++) {
      b[j] = (float)block[k+j];
    }
    /* Horizontal transform */
    for (j = 0; j < 4; j++) {
      j1 = 7 - j;
      b1[j] = b[j] + b[j1];
      b1[j1] = b[j] - b[j1];
    }
    b[0] = b1[0] + b1[3];
    b[1] = b1[1] + b1[2];
    b[2] = b1[1] - b1[2];
    b[3] = b1[0] - b1[3];
    b[4] = b1[4];
    b[5] = (b1[6] - b1[5]) * f0;
    b[6] = (b1[6] + b1[5]) * f0;
    b[7] = b1[7];
    d[i][0] = (b[0] + b[1]) * f4;
    d[i][4] = (b[0] - b[1]) * f4;
    d[i][2] = b[2] * f6 + b[3] * f2;
    d[i][6] = b[3] * f6 - b[2] * f2;
    b1[4] = b[4] + b[5];
    b1[7] = b[7] + b[6];
    b1[5] = b[4] - b[5];
    b1[6] = b[7] - b[6];
    d[i][1] = b1[4] * f7 + b1[7] * f1;
    d[i][5] = b1[5] * f3 + b1[6] * f5;
    d[i][7] = b1[7] * f7 - b1[4] * f1;
    d[i][3] = b1[6] * f3 - b1[5] * f5;
  }
  /* Vertical transform */
  for (i = 0; i < 8; i++) {
    for (j = 0; j < 4; j++) {
      j1 = 7 - j;
      b1[j] = d[j][i] + d[j1][i];
      b1[j1] = d[j][i] - d[j1][i];
    }
    b[0] = b1[0] + b1[3];
    b[1] = b1[1] + b1[2];
    b[2] = b1[1] - b1[2];
    b[3] = b1[0] - b1[3];
    b[4] = b1[4];
    b[5] = (b1[6] - b1[5]) * f0;
    b[6] = (b1[6] + b1[5]) * f0;
    b[7] = b1[7];
    d[0][i] = (b[0] + b[1]) * f4;
    d[4][i] = (b[0] - b[1]) * f4;
    d[2][i] = b[2] * f6 + b[3] * f2;
    d[6][i] = b[3] * f6 - b[2] * f2;
    b1[4] = b[4] + b[5];
    b1[7] = b[7] + b[6];
    b1[5] = b[4] - b[5];
    b1[6] = b[7] - b[6];
    d[1][i] = b1[4] * f7 + b1[7] * f1;
    d[5][i] = b1[5] * f3 + b1[6] * f5;
    d[7][i] = b1[7] * f7 - b1[4] * f1;
    d[3][i] = b1[6] * f3 - b1[5] * f5;
  }
  /* Zigzag - scanning */
  for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
      *(coeff + zz[i][j]) = (int)(d[i][j]);
    }
  }
  return 0;
}
```

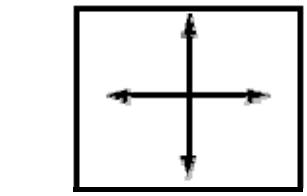
Introducción sobre compresión de video

La compresión de video surge de la necesidad de transmitir imágenes a través de un canal que contenga un ancho de banda aceptable. A continuación se examinarán cuales son los métodos más utilizados que permiten obtener este resultado, y las diferentes normas que se utilizan hoy día.

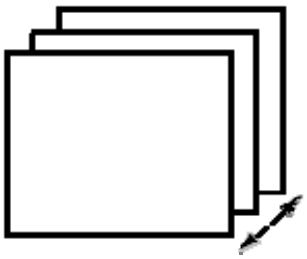
Estos métodos de compresión, recurren a los procedimientos generales de compresión de datos, aprovechando además la redundancia espacial de una imagen (áreas uniformes), la correlación entre puntos cercanos y la menor sensibilidad del ojo a los detalles finos de las imágenes fijas (JPEG) y, para imágenes animadas (MPEG), se saca provecho también de la redundancia temporal entre imágenes sucesivas.

La figura muestra que cuando las imágenes individuales son comprimidas sin referencia a las demás, el eje del tiempo no entra en el proceso de compresión, esto por lo tanto se denomina codificación intra (intra=dentro) o codificación espacial. A medida que la codificación espacial trata cada imagen independientemente, esta puede emplear ciertas técnicas de compresión desarrolladas para las imágenes fijas. El estándar de compresión ISO (International Standards Organization) JPEG, está en esta categoría.

Se pueden obtener grandes factores de compresión teniendo en cuenta la redundancia entre imágenes sucesivas. Esto involucra al eje del tiempo, por lo que este proceso se denomina codificación inter (inter=entre) o codificación temporal (la siguiente figura muestra esto).



Codificación intra o espacial, explora la redundancia dentro de la imagen



Codificación inter o temporal, explora la redundancia entre imágenes

La codificación temporal permite altos factores de compresión, pero con la desventaja de que una imagen individual existe en términos de la diferencia entre imágenes previas. Si una imagen previa es quitada en la edición, entonces los datos de diferencia pueden ser insuficientes para recrear la siguiente imagen.

Codificación intra o espacial

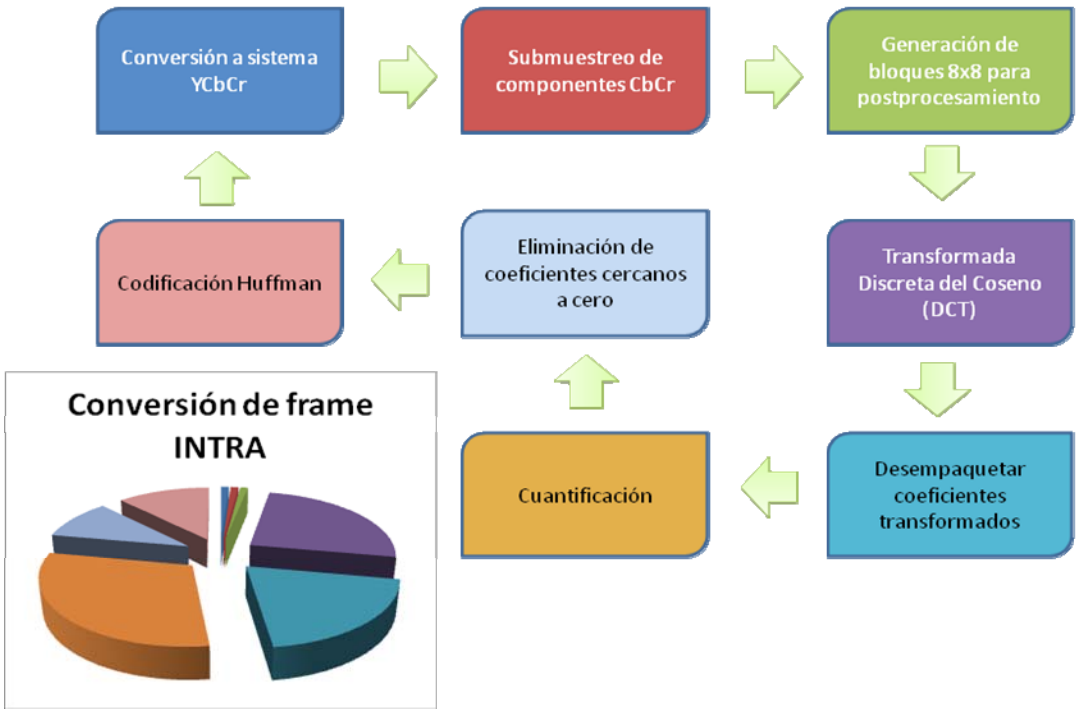
Un análisis de las imágenes de televisión revela que existe un alto contenido de frecuencias espaciales debido al detalle en algunas áreas de la imagen, generando una cantidad pequeña de energía en tales frecuencias. A menudo las imágenes contienen considerables áreas en donde existen pixeles con un mismo valor espacial. Además, como el promedio de brillo de la imagen se caracteriza por componentes de frecuencia de valor cero, siempre será una buena opción trabajar con histogramas y componentes en frecuencia de imágenes, ya que tendremos unas estructuras más manejables dado que la mayoría de transformaciones y algoritmos de codificación utilizan dicho desarrollo. Así que, si lo que deseamos es deshacernos de gran parte de información irrelevante presente en una imagen, simplemente deberemos omitir los componentes de alta frecuencia de la misma.

Una disminución en la codificación se puede obtener, tomando como ventaja que la amplitud de los componentes espaciales disminuye con la frecuencia. Si el espectro de frecuencia espacial es dividido

en subbandas de frecuencia, las bandas de alta frecuencia se pueden describir en pocos bits, no solamente porque sus amplitudes son pequeñas sino porque puede ser tolerado más ruido.

Este codificador debe emplearse siempre que se desee extraer información presente en imágenes, como detección de bordes, formas o estructuras. Este apartado se detalla más adelante y todas y cada una de las aplicaciones que en él se desarrollen pueden ser utilizadas junto a este codificador en tiempo real.

A continuación se describe un diagrama de bloques conceptual con los pasos a seguir para codificar de cada uno de los frames, así como su porcentaje en tiempo de procesamiento medido en las pruebas de rendimiento efectuadas por los desarrolladores del estándar pertenecientes a la UIT (ver referencia oficial del estándar para más información):



Estructura y tiempos

Este proceso se representa de forma circular porque es necesario aplicarlo a cada una de las imágenes tomadas. Traducido a algoritmo de programación podría verse como un bucle infinito en el que se realizan secuencialmente cada una de las acciones detalladas anteriormente.

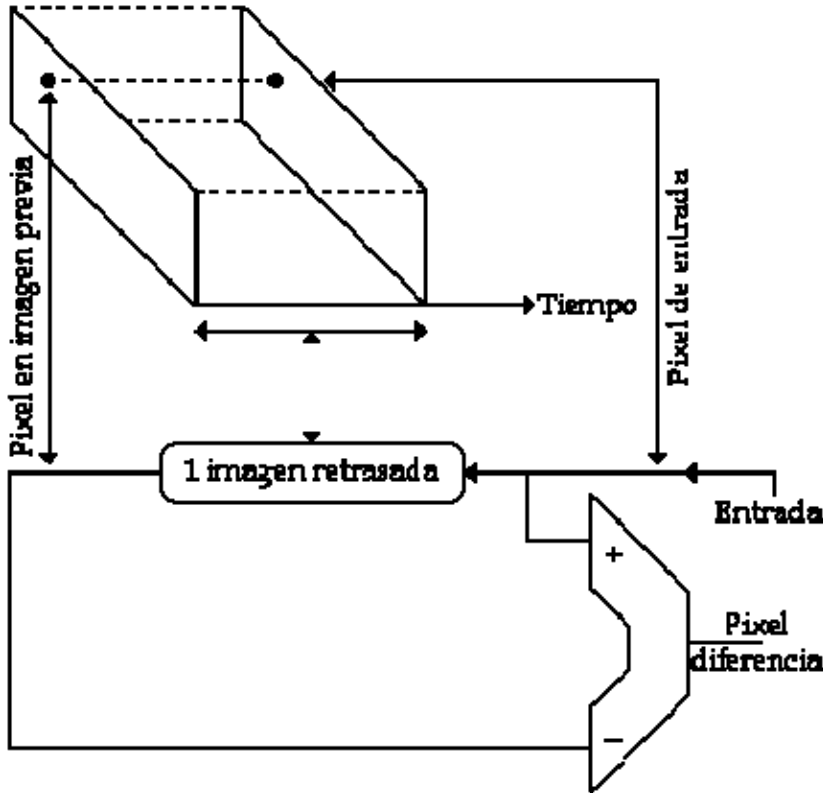
Codificación inter o temporal

La codificación inter aprovecha la ventaja que existe cuando las imágenes sucesivas son similares. En lugar de enviar la información de cada imagen por separado, el codificador inter envía la diferencia existente entre la imagen previa y la actual en forma de codificación diferencial. La figura siguiente muestra este principio. El codificador necesita de una imagen referencia, la cual fue almacenada con anterioridad para luego ser comparada entre imágenes sucesivas y de forma similar se requiere de una imagen previamente almacenada para que el decodificador desarrolle las imágenes siguientes.

Los datos que se generan al hacer la diferencia entre dos imágenes, también se pueden tratar como una nueva imagen, la cual se debe someter al mismo tratamiento de transformadas utilizado en la compresión espacial.

Un sistema básico de codificación inter se muestra en la figura siguiente. Desafortunadamente existe la posibilidad de transmitir errores, si se utiliza una secuencia ilimitada de imágenes previstas. Por

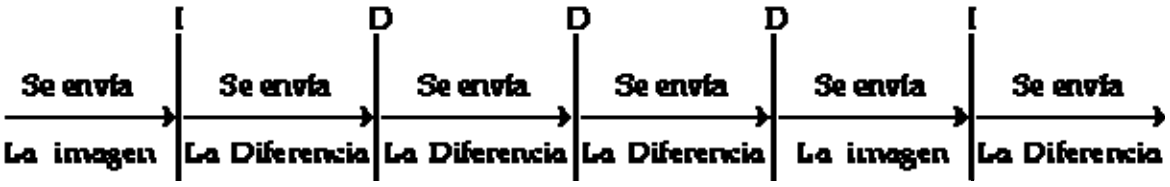
esto es mejor utilizar un número limitado de imágenes previstas para de este modo garantizar una menor acumulación de errores.



Sistema de codificación inter

La próxima figura muestra el recorrido de una imagen original, llamada imagen I o intra, la cual es enviada entre imágenes que han sido creadas usando una diferencia entre imágenes, llamada imágenes P o previstas (en el diagrama siguiente aparecen tramas D que simbolizan “diferencias” con respecto a la imagen de referencia I).

La imagen I requiere grandes cantidades de información, mientras que las imágenes P requieren una cantidad menor. Esto ocasiona que el flujo de transmisión de datos sea variable hasta cuando llegan a la memoria intermedia, la cual genera a su salida una transmisión de datos de forma constante. También se puede observar que el predictor necesita almacenar datos de menor proporción puesto que su factor de compresión no cambia de una imagen a otra.



I = Imagen codificada intra.
D = Imagen codificada diferencialmente.

Una secuencia de imágenes constituida por una imagen I y las siguientes imágenes P hasta el comienzo de otra imagen I, se denomina grupo de imágenes GOP (Group Of Pictures). Para factores de compresión altos se utiliza un número grande de imágenes P, haciendo que las GOPs aumenten de tamaño

considerablemente; sin embargo un GOP grande evita recuperar eficazmente una transmisión que ha llegado con errores.

La estructura sintáctica que forma un GOP puede resumirse en la siguiente figura:

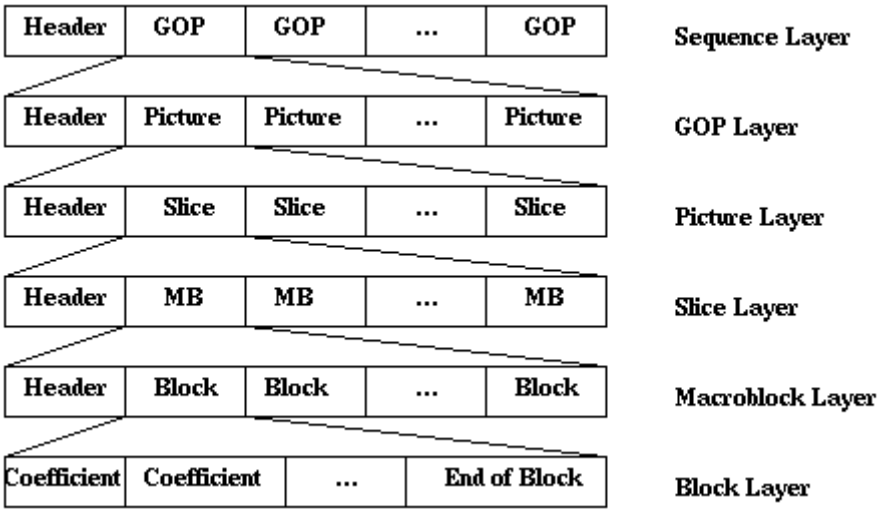


Figura 45: Estructura de un GOP

Como podemos ver, un *GOP* engloba varias *imágenes* (pictures) y éstas a su vez están formadas por *rodajas* (slices). Cada *rodaja* equivale a una pseudocapa agrupada en *macrobloques* de tamaño constante (MB), formados por *bloques* (Block) que poseen los *coeficientes* que llevan la información de cada zona de una rodaja. Cada *subcapa* tiene una *cabecera* que sirve para localizar de forma absoluta cada *subsegmento* de la imagen con respecto a la imagen general.

Codificación bidireccional

Cuando un objeto se mueve, este oculta lo que hay detrás de él, pero esto va cambiando a medida que se va moviendo, permitiendo observar el fondo. El revelado del fondo exige nuevos datos a ser transmitidos, ya que el área del fondo había sido ocultada anteriormente y la información no pudo ser obtenida desde una imagen previa.

Un problema similar ocurre si se hace una toma panorámica con una cámara de video ya que aparecen nuevas áreas ante el observador y nada se sabe acerca de ellas. El proceso de codificación bidireccional ayuda a minimizar este problema ya que deja información para ser tomada de imágenes anteriores y posteriores a la imagen observada. Si el fondo ya ha sido revelado, y este será presentado en una imagen posterior, la información puede ser movida hacia atrás en el tiempo, creando parte de la imagen con anticipación.

La figura muestra en qué se basa la codificación bidireccional. En el centro del diagrama un objeto se mueve revelando su fondo, pero éste no se conoce hasta la siguiente imagen. Entonces se toman los datos de las imágenes anteriores y posteriores, o incluso se utiliza el promedio de los datos, descubriendo de esta forma el fondo. En la figura se muestra una codificación bidireccional. Primero se toma una imagen I y, con la ayuda de una imagen P se pueden obtener imágenes B, las cuales son llamadas también imágenes bidireccionales.

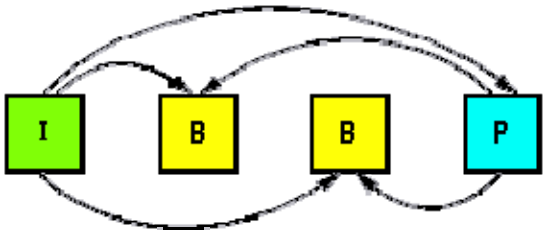


Figura 46: Codificación bidireccional

Las **imágenes I** se codifican como si fuesen imágenes fijas utilizando la norma JPEG, por tanto, para decodificar una imagen de este tipo no hacen falta otras imágenes de la secuencia, sino sólo ella misma. No se considera la redundancia temporal (compresión *intraframe*). Se consigue una moderada compresión explotando únicamente la redundancia espacial. Una imagen I siempre es un punto de acceso en el flujo de bits de vídeo. Son las imágenes más grandes.

Las **imágenes P**: Están codificadas como predicción de de la imagen I ó P anterior usando un mecanismo de compensación de movimiento. Para decodificar una imagen de este tipo se necesita, además de ella misma, la I ó P anterior. El proceso de codificación aquí explota tanto la redundancia espacial como la temporal.

Las **imágenes B**: Se codifican utilizando la I ó P anterior y la I ó P siguiente como referencia de compensación y estimación de movimiento. Para decodificarlas hacen falta, además de ellas mismas, la I ó P anterior y la I ó P siguiente. Estas imágenes consiguen los niveles de compresión más elevados y por tanto son las más pequeñas.