Trabajo Final de Master

Título: Diseño, implementación y optimización del sistema de compresión de imágenes sobre el ordenador de a bordo del proyecto de nanosátelite Eye-Sat.

Autor

Lorenzo Ortega Espluga

Director- CNES

Roberto Camarero Rodríguez

Directora-Universidad Zaragoza

Paloma García Dúcar

Marzo 2016 – Septiembre 2016

Resumen

Eye-Sat es un Proyecto de nano satélites, dirigido por el CNES (Centre National d'Etudes Spatiales) y desarrollado principalmente por estudiantes de varias escuelas de ingeniería del territorio francés.

El objetivo de este pequeño telescopio no solo radica en la oportunidad de realizar la demostración de distintos dispositivos tecnológicos, sino que también tiene como misión la adquisición de fotografías en la bandas de color e infrarrojo de la vía Láctea, así como el estudio de la intensidad y polarización de la luz Zodiacal.

Los requerimientos de la misión exigen el desarrollo de un algoritmo de compresión de imágenes sin pérdidas para las imágenes "Color Filter Array" CFA (Bayer) e infrarrojas adquiridas por el satélite. Como miembro de la comisión consultativa para los sistemas espaciales, CNES ha seleccionado el estándar CCSDS-123.0-B como algoritmo base para cumplir los requerimientos de la misión. A este algoritmo se le añadirán modificaciones o mejoras, adaptadas a las imágenes tipo, con el fin de mejorar las prestaciones de compresión y de complejidad.

La implementación y la optimización del algoritmo será desarrollada sobre la plataforma Xilinx Zynq® All Programmable SoC, el cual incluye una FPGA y un Dual-core ARM® Cortex™-A9 processor with NEON™ DSP/FPU Engines

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

1 INTRODUCTION

1.1 CNES

Bajo la supervisión del ministerio de investigación y defensa, el CNES (Centro Nacional de Estudios Espaciales) es un establecimiento público para objetivos industriales y comerciales. La agencia fue creada en 1961 bajo la dirección del general De Gaulle, para diseñar, desarrollar y proponer al gobierno estrategia espacial.

Así, el CNES es responsable de desarrollar los programas espaciales franceses. A través de eso, el gobierno francés pretende preservar la soberanía nacional en la ciencia, así como incentivar y tomar parte en las más avanzadas investigaciones del espacio. Como tal, el CNES está desarrollando sistemas espaciales mediante los últimos avances tecnológicos, los cuales garantizan un acceso al espacio de manera independiente.

Buscando como introducir la tecnología espacial para el uso cotidiano, no solo por medio de soluciones actuales, si no también anticipándonos a futuros desarrollos. Además, la agencia sabe cómo tratar con colaboradores tanto en el ámbito industrial como empresarial, lo cual, es esencial en las actuales políticas espaciales.

Altamente involucrada en los programas de la agencia espacial europea (ESA), donde Francia es uno de los mayores contribuidores, CNES desarrolla un gran rol sobre la política espacial de la unión Europea mediante el desarrollo de proyectos y de misiones encomendadas.

1.2 Objetivo de las prácticas

1.2.1 Contexto

Dirigida por el CNES pero desarrollado por estudiantes, Eye-Sat es un proyecto de nano-satélites para una misión de astronomía. El propósito del proyecto [1] consiste en la continuación del proyecto en ambos, segmento suelo y segmento espacio en lo denominado fase C/D. Las prácticas serán llevadas a cabo en el servicio de **« informatique bord et équipements »** el cual está bajo la dirección de « techniques véhicule, architecture & intégration » (TV/IN). Este servicio está a cargo del desarrollo de equipo de procesamiento a bordo: procesadores, bus de datos, codificadores, compresores de datos….

1.2.2 Objetivo

Dentro de contexto Eye-Sat, el cual se muestra en la Figura 1, y con el objetivo de maximizar las capacidades de la misión, CNES desea estudiar la implementación de algoritmos de compresión de imágenes sin pérdidas dentro del procesador de abordo. Los métodos seleccionados deberán ser de baja complejidad, además de satisfacer buenos resultados de compresión. Las posibles imágenes serán captadas por un mismo captor el cual podrá adquirir imágenes infrarrojas o imágenes CFA “Color Filter Array”, en concreto en matriz de Bayer.

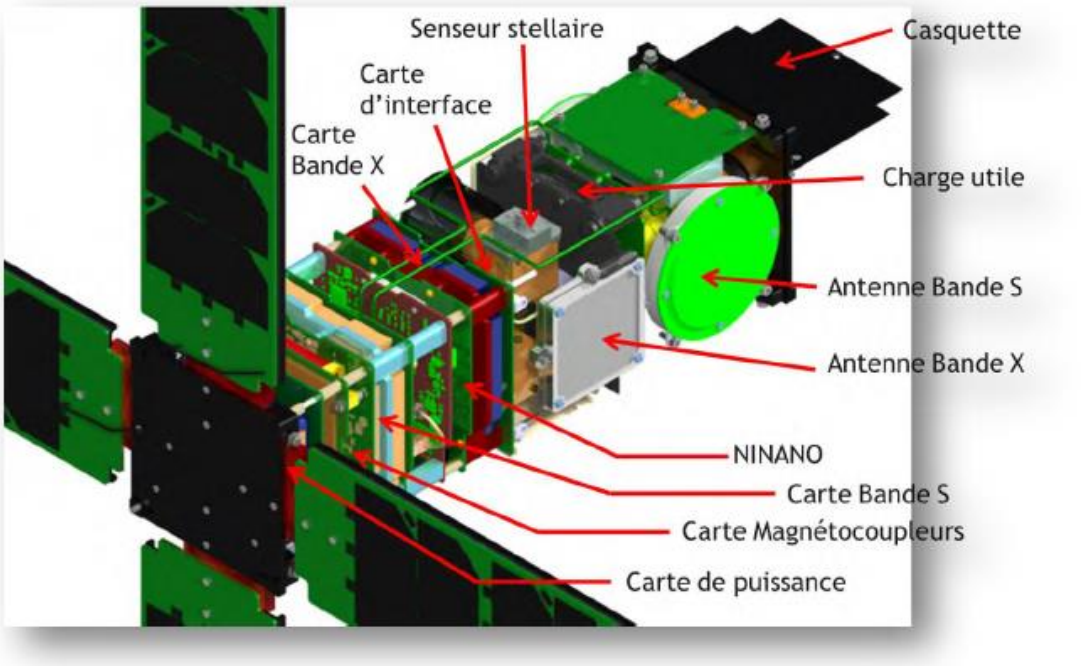


Figura 1- Eye-Sat nanosatellite

1.2.3 Pasos del Proyecto

1. Revisión del estado del arte sobre métodos de compresión de imagen sin pérdidas, así como de procesamiento sobre arquitecturas vectoriales.

2. Uso y manejo de herramientas de visualización de imágenes y de cálculo de rendimientos. ENVI y MATLAB.

3. Estudio preliminar y selección de algoritmos según sus rendimientos. Estudio de posibles cambios dentro de los algoritmos para una mejor adaptación a las imágenes objetivo (Matriz de Bayer).

4. Análisis del rendimiento en la eficiencia de compresión y la complejidad del algoritmo.

5. Implementación del algoritmo seleccionado, así como compilación del código dentro el procesador misión (ARM® heart of ZYNQ® component). Realizar los test de rendimiento.

6. Optimización del código, si fuera necesario, sobre la arquitectura especifica seleccionada (NEON ™: SIMD vector processor)

7. Obtención de una función final optimizada y testeada de compresión dentro del microprocesador ARM®.

8. Escritura de la memoria y presentación oral dentro del CNES.

2 Algoritmo de compresión

2.1 Introducción

El objetivo de la compresión de imagen consiste en la reducción de la redundancia (información innecesaria) de los píxeles de la imagen para almacenarla o transmitirla de manera eficiente. Para llevar a cabo este proceso se puede realizar los pasos descritos por la Figura 2.

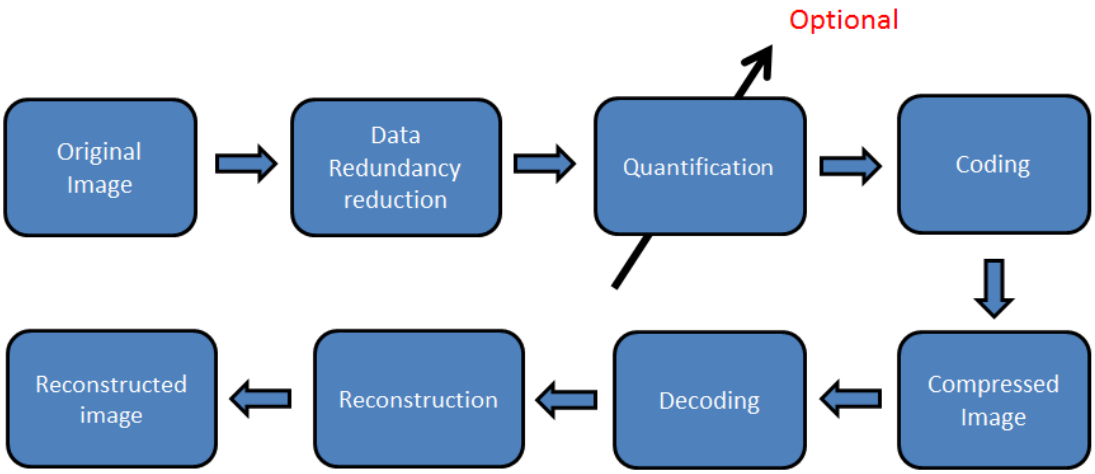


Figura 2- Diagrama de bloques para la compresión de imágenes

Como se observa, para realizar una compresión de una imagen, lo primero que se debe llevar a cabo es una reducción de la redundancia. Dentro de este campo podemos llevar a cabo distintas técnicas como técnicas predictivas o técnicas de trasformadas ortogonales. En caso de llevar a cabo una compresión con pérdidas deberemos introducir un cuantificador, en caso contrario, los datos obtenidos serán introducidos a un codificador entrópico. El codificador entrópico mide la frecuencia de los datos entrantes y realiza un código equivalente con el fin de reducir el número de datos a transmitir.

No cabe duda que esta serie de pasos tienen su proceso inverso y que en caso de no utilizar un cuantificador, la imagen obtenida al final del diagrama de bloque será equivalente a la imagen original.

2.2 Imágenes

Respecto a las imágenes que van a ser tomadas en la misión, el sensor tomara imágenes infrarrojas además de imágenes en las bandas RGB. Las imágenes tendrán una dimensión de 2048x2048 píxeles. Además se tomarán imágenes en negro para llevar a cabo la calibración de los instrumentos. Las imágenes serán tomadas con una frecuencia de 15 segundos, así que a efectos prácticos podemos considerar secuencias de imágenes en vez de imágenes independientes. Por otro lado, una de las mayores particularidades del proyecto reside en la toma de las imágenes RGB. En concreto, dichas imágenes se tomara con patrón de matriz de Bayer, el cual está ilustrado en la Figura 3.

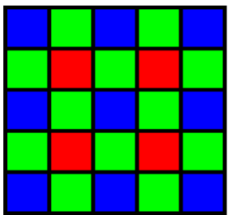


Figura 3- CFA Matriz de Bayer

Como se puede llegar a observar, dicho patrón está compuesto por la mitad de píxeles verde, una cuarta parte para píxeles de color azul y otra cuarta parte de píxeles de color rojo. El principal motivo de llevar a cabo este tipo de capturas (con patrón de matriz de Bayer) es la reducción del coste de la cámara, ya que en vez de llevar integrados 3 filtros distintos, solo lleva uno con el patrón de la matriz de Bayer.

El hecho de que haya el doble de píxeles de color verde que de color rojo o azul tiene que ver con que el color verde tiene una mayor relación con la luminancia y por ello nos aportará una mayor información.

Por último, he de destacar que existe una técnica denominada demosaicing, la cual por medio de combinaciones lineales de los píxeles de la matriz de Bayer es capaz de obtener las tres imágenes RGB. Esta operación por supuesto se llevará a cabo en el segmento tierra, ya que implica una disminución del cómputo dentro de la placa de satélite y una disminución del número de píxeles a comprimir.

2.3 Algoritmo

A lo largo del proyecto [1], se llevó a cabo el desarrollo de numerosos algoritmos con el fin de encontrar aquel algoritmo que se adaptase a las especificaciones del proyecto de manera más rigurosa. Estos algoritmos pueden ser divididos principalmente en dos clases. El primer tipo consiste en desarrollar algoritmos de compresión de imágenes sin pérdidas. La segunda clase de algoritmos de compresión, se basa en el hecho de que cada imagen será parte de una secuencia de imágenes y por tanto quizá se pudiera explotar la redundancia de la dimensión tiempo, es decir, utilizar técnicas de compresión video.

Tras realizar la implementación de diferentes algoritmos descartamos el uso de las técnicas de compresión video debido a que para realizar una buena estimación de movimiento la complejidad de los algoritmos aumenta. Además el hecho de que el satélite pudiera tener un movimiento rotacional, además del transversal, puede provocar una disminución de las prestaciones de los algoritmos de compresión video testeados.

Definitivamente optamos por la utilización de algoritmos de compresión de imágenes sin pérdidas. En concreto, el algoritmo implementado está basado en el estándar CCSDS-123.0-B, el cual usa un sistema de compresión predictiva, un mapping y un codificador entrópico.

El diagrama bloque base, el cual puede describir nuestro algoritmo está ilustrado en la Figura 4. Dicho diagrama se divide principalmente en tres bloques. Un predictor cuya función consiste en eliminar la redundancia entre píxeles. Un mapping, el cual se usa para que los residuos creados por el predictor sean adaptados para el codificador entrópico. Por último el codificador entrópico, el cual tiene como objetivo codificar de manera inteligente cada uno de los símbolos entrantes, mediante el conocimiento de la probabilidad de cada uno de los símbolos.

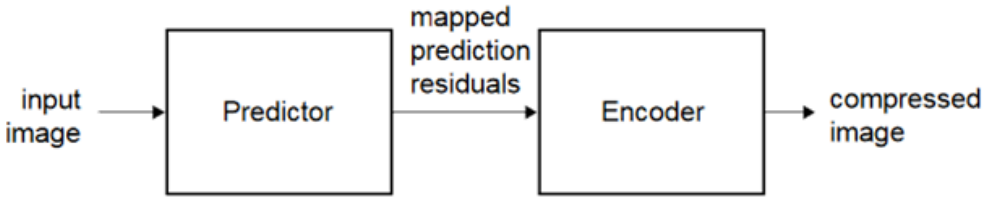


Figura 4- Diagrama de bloque del algoritmo de compresión.

Como se hizo mención con anterioridad, el algoritmo seleccionado para la misión está basado en la estándar CCSDS-123.0-B, de hecho, el codificador entrópico utilizado se trata del codificador entrópico adaptativo en bloque que propone dicho estándar [2]. Respecto al predictor, la implementación es completamente innovadora, teniendo en cuenta las limitaciones de cómputo que tenemos. De todas maneras, el predictor se basa en el uso de bandas espectrales para la estimación de bandas espectrales posteriores. Por último, el mapping también es independiente al estándar y al igual que el predictor se ha elaborado con el fin de reducir el número de operaciones.

2.3.1 Predictor

El predictor tiene 3 modos de utilización los cuales pueden utilizarse dependiendo el tipo de imagen a codificar. Todos ellos se basan en la estimación de píxeles por medio de píxeles adyacentes o píxeles de bandas de frecuencia anteriores. El motivo del uso de las bandas precedentes para la predicción puede ser explicado por medio de la Figura 5.

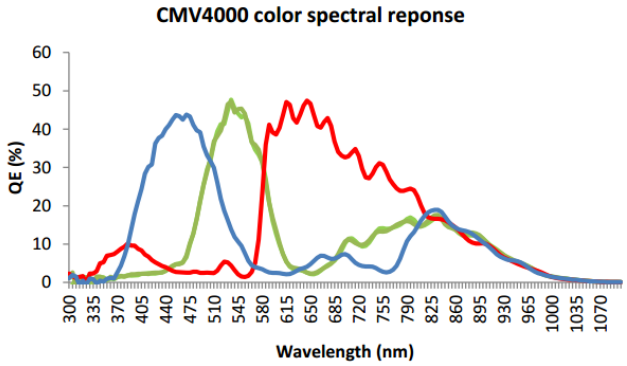
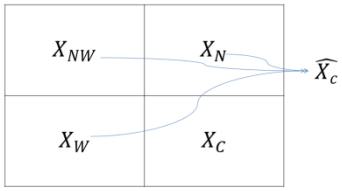


Figura 5-Respuesta espectral de la cámara.

Como se observa en la imagen, podemos deducir dos conceptos, los cuales serán de ayuda a la hora de realizar a cabo el diseño del predictor. El primero de ellos es que la respuesta espectral del verde comparte información común con la respuesta espectral roja y la respuesta espectral de la banda azul. Dicha información será interesante a la hora de estimar los píxeles rojos y azules de la matriz de Bayer. La segunda cuestión interesante que podemos llegar a observar se localiza en la banda infrarroja. Como se observa, las respuestas espectrales del rojo, verde y azul son similares y por tanto podremos considerar dicha imagen, como una imagen en escala de grises.

Para llevar a cabo la explicación de cada uno de los predictores, comenzaremos por la definición de una serie de variables que nos ayudarán a la hora de relacionar e interpretar las ecuaciones.

- $X_N = North\ pixel$
- $X_W = West\ pixel$
- $X_{NW} = North - West\ pixel$
- $X_C = Actual\ pixel$
- $\widehat{X_c} = Estimated\ pixel$
- $R = Residual$



Donde el residuo es definido como:

$$R = \widehat{X_c} - X_C \tag{1}$$

Por supuesto, el objetivo del predictor consiste en obtener un residuo lo más bajo posible. Para ello, $\widehat{X_c}$ debe ser un número entero, lo más parecido a X_C.

2.3.1.1 Primer predictor

El primer predictor será verdaderamente útil para las imágenes de negro, infrarrojo o aquellas imágenes CFA que tenga mucha componente oscura. La imagen será tratada como una única banda. Para obtener los residuos se deben emplear las ecuaciones *2-3-4-5*. Además dichas ecuaciones vienen descritas por medio de la Figura 6.

- *First Sample*

$$X_N = 0, \quad X_W = 0, \quad X_{NW} = 0, \quad \widehat{X_c} = 0, \quad R = X_C \tag{2}$$

- *First Row*

$$X_N = 0, \quad X_W \neq 0, \quad X_{NW} = 0, \quad \widehat{X_c} = X_W, \quad R = X_W - X_C \tag{3}$$

- *First Column*

$$X_N \neq 0, \quad X_W = 0, \quad X_{NW} = 0, \quad \widehat{X_c} = X_N, \quad R = X_N - X_C \tag{4}$$

- *Central Sample*

$$X_N \neq 0, \quad X_W \neq 0, \quad X_{NW} \neq 0, \quad \widehat{X_c} = \frac{X_N}{4} + \frac{X_W}{2} + \frac{X_{Nw}}{4}, \tag{5}$$
$$R = \left(\frac{X_N}{4} + \frac{X_W}{2} + \frac{X_{Nw}}{4}\right) - X_C$$

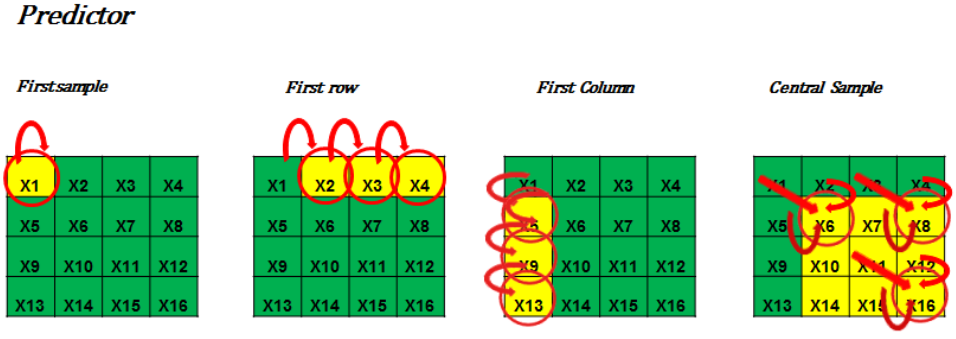


Figura 6- Primer predictor

Como se puede observar, cada residuo se puede obtener con un máximo de 3 sumas debido a que las divisiones se pueden obtener mediante el desplazamiento de bits (lo cual es gratuito en términos de complejidad). Además, como X_N y X_W son X_{NW} y X_C del píxel siguiente, podemos almacenarlos en la caché evitándonos el acceso de dichas muestras a la memoria externa (en la cual se encuentra almacenada la imagen a comprimir). Para resumir podemos concluir que el número de operaciones máxima para realizar la predicción de cada uno los residuos será equivalente a:

$$2\ Read\ Memory\ Access, +3\ Sums + 1\ Write\ memory\ Access \qquad (6)$$

Enfatizo la palabra máxima debido a que para el caso "*First row*" y "*First column*" solo habrá un acceso a memoria y una suma. Para el caso "*First Sample*" solo se exigirá un acceso a memoria.

2.3.1.2 Segundo predictor

El segundo predictor puede llegar a ser visto como el primer predictor con la única diferencia de estar realizando la predicción para dos bandas completamente independientes. Las bandas seleccionadas son una composición de píxeles rojo-verde para la primera banda y azul-verde para la segunda banda. El hecho de utilizar dichas bandas se basa en la idea remarcada mediante la Figura 5, el cual enfatiza que el verde comparte información con las bandas frecuenciales correspondientes al rojo y al azul.

Con respecto al número de operaciones, y puesto que estamos dividiendo el número de datos en dos bandas distintas, llevaremos a cabo un menor número de operaciones, debido a tener dos primeras columnas. De todas formas esta ganancia respecto a la complejidad del sistema es prácticamente despreciable debido a que el mayor número de píxeles corresponde al subconjunto "Central simple".

Por ultimo destacamos que a la hora de realizar la implementación real, si queremos que la complejidad computacional siga constante deberemos tener en cuenta la indexación real, una muestra de dicha indexación está ilustrada en la Figura 8.

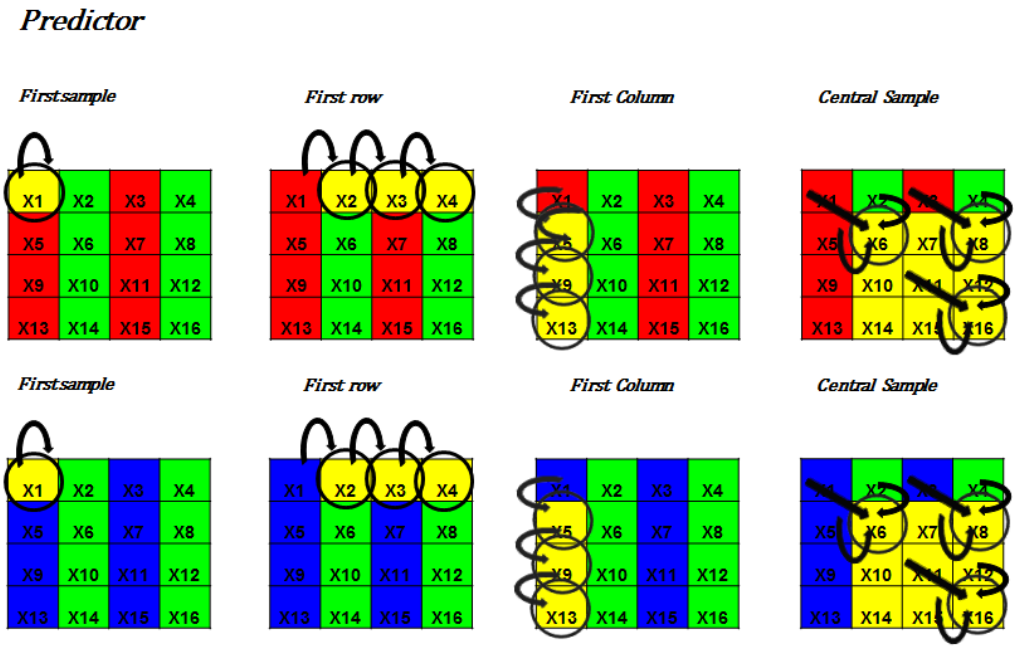


Figura 7- Segundo Predictor

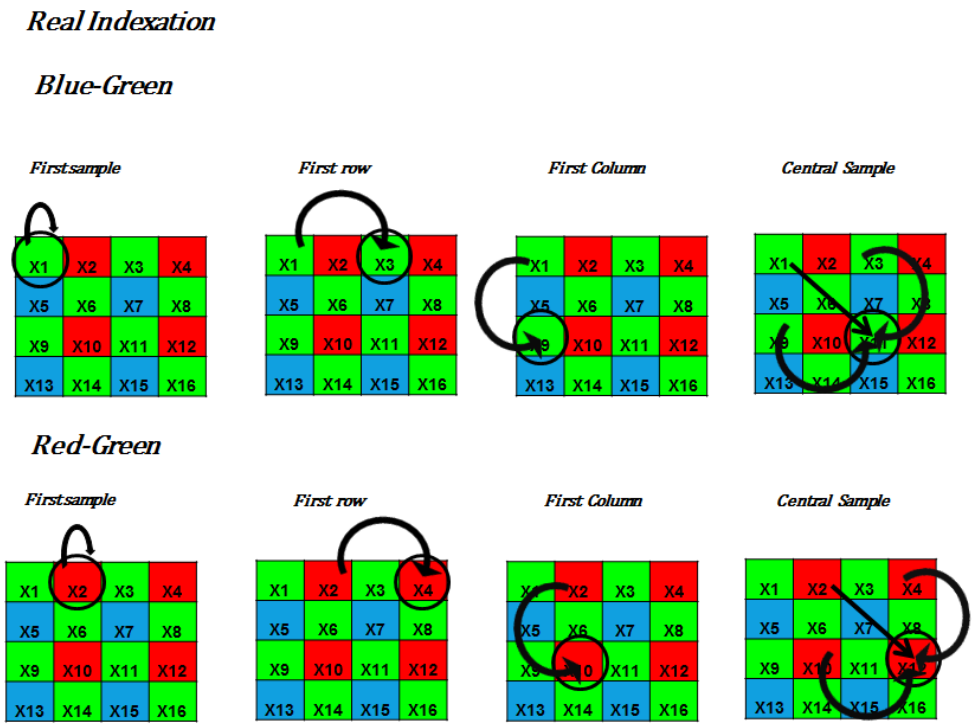


Figura 8- Indexación real predictor 2

2.3.1.3 Tercer predictor

El tercer predictor se basa en la separación de los datos en tres bandas independientes según la tipología del píxel Verde-Azul-Rojo. El tamaño de cada una de estas subbandas será de 1024x2048 para la banda verde y de 1024x1024 para las bandas roja y azul. La figura 9 ilustra los patrones de predicción.

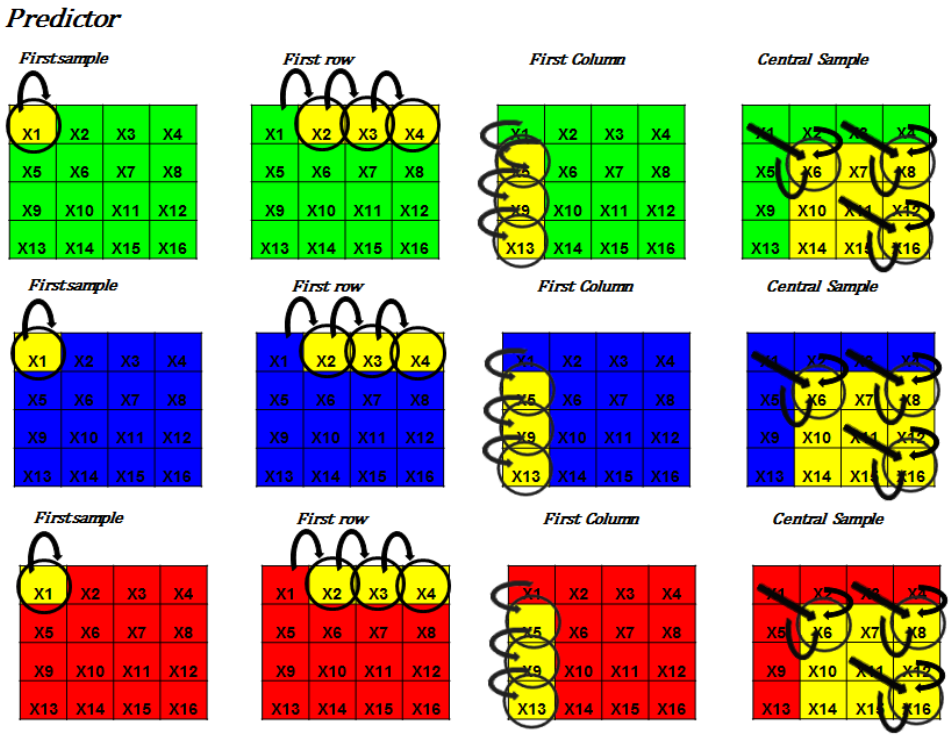


Figura 9- Tercer Predictor

Respecto a la indexación real, la Figura 10 muestra como llevará a cabo dicha implementación para las bandas azul y rojo.

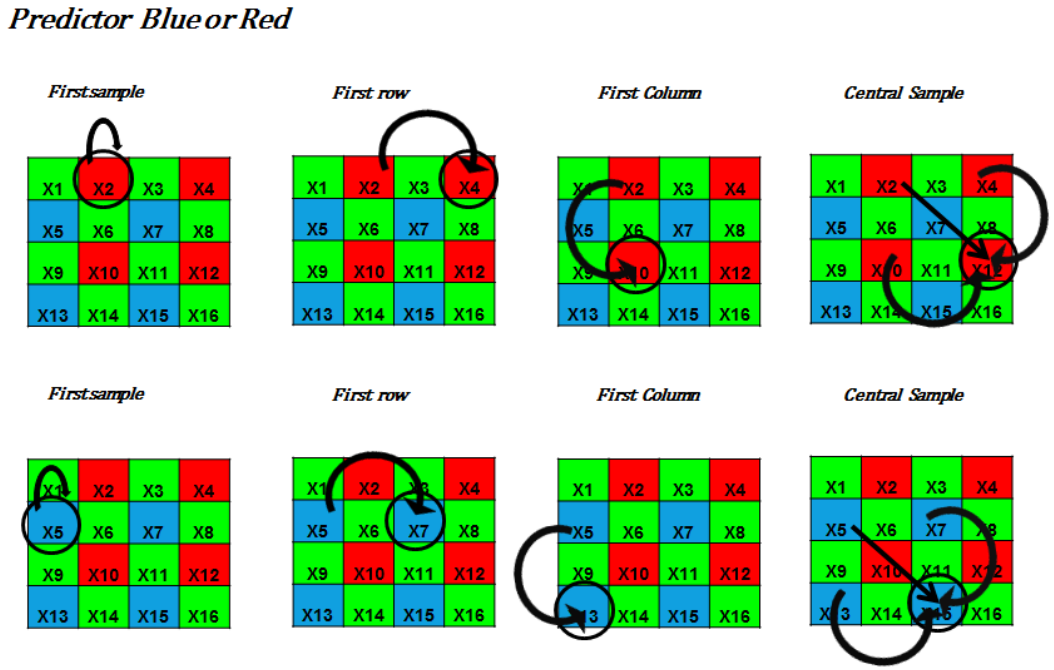


Figura 10- Indexación real Tercer-Predictor Bandas Azul y Roja

Para la banda verde el patrón de indexación se modifica de manera simple. Dicha modificación se muestra en la Figura 11.

Con $X_C = X_{x,y}$

$$X_N = X_{x+1,y-1}, \qquad X_W = X_{x-2,y}, \qquad X_{NW} = X_{x-1,y-1} \tag{7}$$

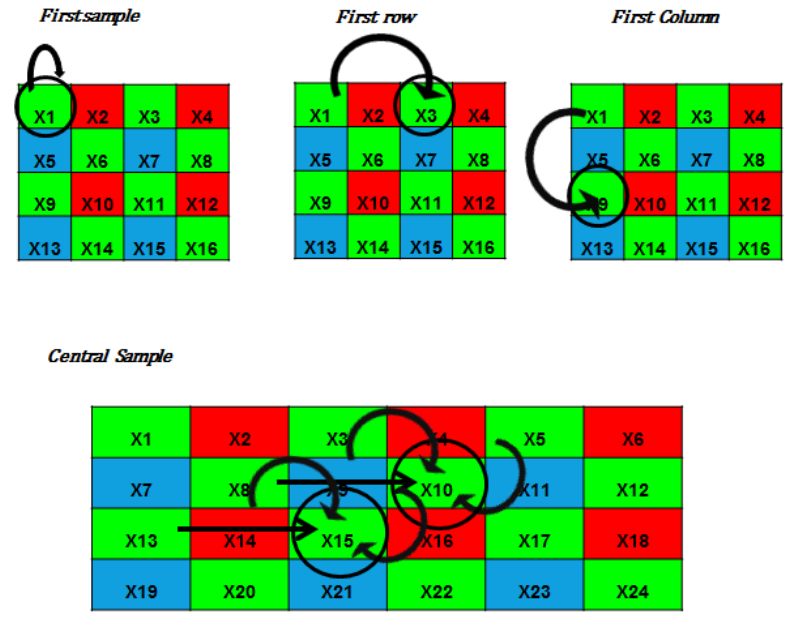


Figura 11-Indexación real Tercer-Predictor Banda Verde

Debemos hace hincapié en el caso de la última columna debido a que $X_N = 0$ o directamente no existe.

Como se especificó con anterioridad el predictor permite el uso de la explotación de la redundancia espectral. Para ello vamos a utilizar los residuos obtenidos en la banda verde para reestimar los residuos obtenidos en las bandas roja y azul. Es decir, vamos a realizar una re-estimación de las bandas azul y roja. Para realizar dicha estimación simplemente llevaremos a cabo una resta entre las bandas de residuos obtenidos en el proceso de predicción. El hecho de que esta técnica tan simple pueda llevar a cabo mejoras se basa en que en la zonas donde hay alto ruido o señal, estarán independientemente de la banda de frecuencias a estimar y por tanto podemos reutilizar dicha información para llevar a cabo la predicción de las bandas residuales correspondientes al rojo y al azul.

La figura 12 nos ilustra cómo se llevará a cabo la operación descrita en el párrafo anterior.

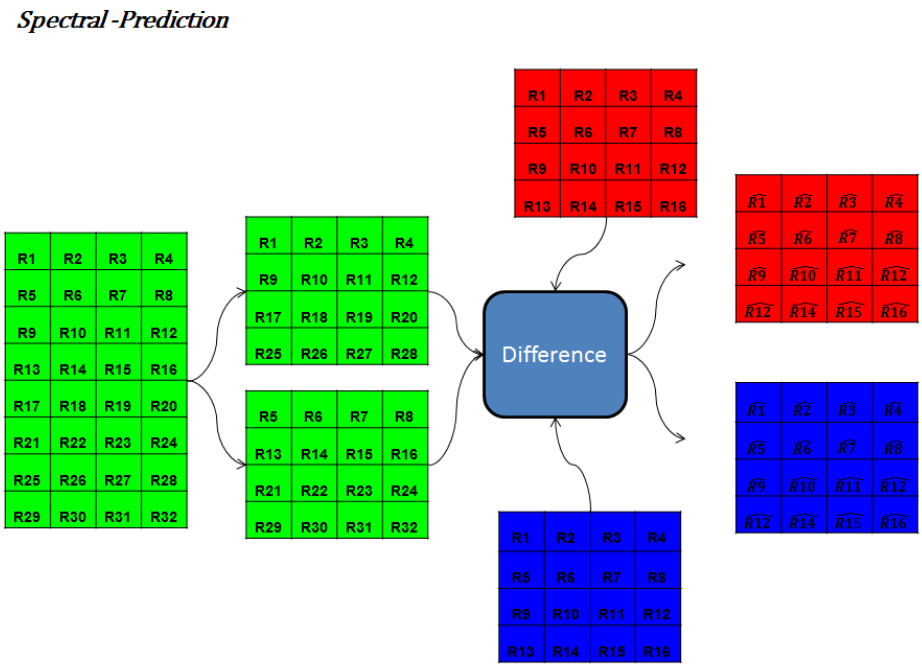


Figura 12- Explotación de la redundancia espectral

2.3.2 Mapping

Con el objetivo de reducir el número de operaciones, dentro del algoritmo hemos llevado a cabo la implementación de un mapping limitado, respecto a la complejidad de cómputo. Como uno se puede imaginar, los residuos pueden ser tanto positivos como negativos, de hecho siguen una distribución probabilística de Rice. Por otro lado nuestro codificador entrópico solo permite la codificación de muestras positivas.

El mapping llevará a cabo las siguientes operaciones: en caso de tratarse de un residuo positivo lo asociará a un número par. En caso de tratarse de un valor negativo lo transformará en un coeficiente impar.

- Números pares : $x \ll 1$

- Números impares $\left((X\ xor\ 0xFFFF) \ll 1\right) + 1$

Como se muestra en las ecuaciones anteriores los números positivos solo utilizan una tabulación en los datos (gratuito a efectos prácticos) y los residuos con valores negativos solo utilizarán una operación binaria (un ciclo de reloj) y una suma.

2.3.3 Segmentación

Uno de los principales problemas que encontramos a la hora de transmitir la información desde el satélite hasta el segmento tierra es la limitación de visibilidad. Al tener un tiempo limitado para transmitir, no existirán mecanismos de retransmisión o ACK. Supongamos que existiera un error en alguno de los paquetes en los cuales se encuentra la imagen comprimida. En la Figura 13 encontramos ilustrado cual sería la problemática.

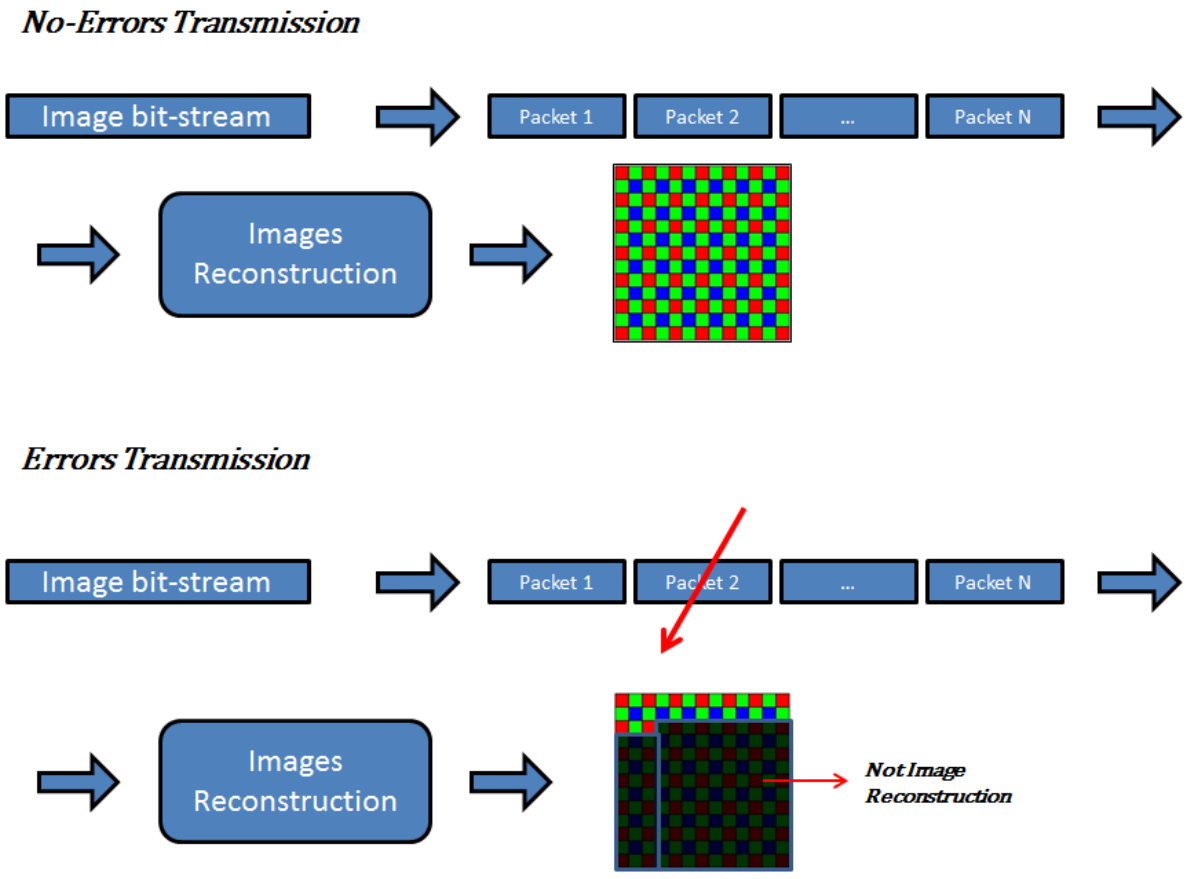


Figura 13-Efecto del Error de transmisión

Cualquier pérdida implicaría no poder reconstruir el resto la imagen. Por tanto si mandamos N paquetes, el ratio de imagen correcta sería de:

$$Arate = 1 - \frac{i}{N} \quad such\ us \quad i = First\ lost\ packet \ \ and \quad N = N°\ packets \tag{8}$$

Si la imagen residuo esta codificada en 20 paquetes y tenemos alguna pérdida en el primer paquete, los 20 paquetes serán enviados en vano.

$$Arate = 1 - \frac{20}{20} = 0 \qquad (9)$$

Para llevar a cabo una contramedida de este suceso, hemos llevado a cabo una técnica de segmentación, la cual es ilustrada en la Figura 14.

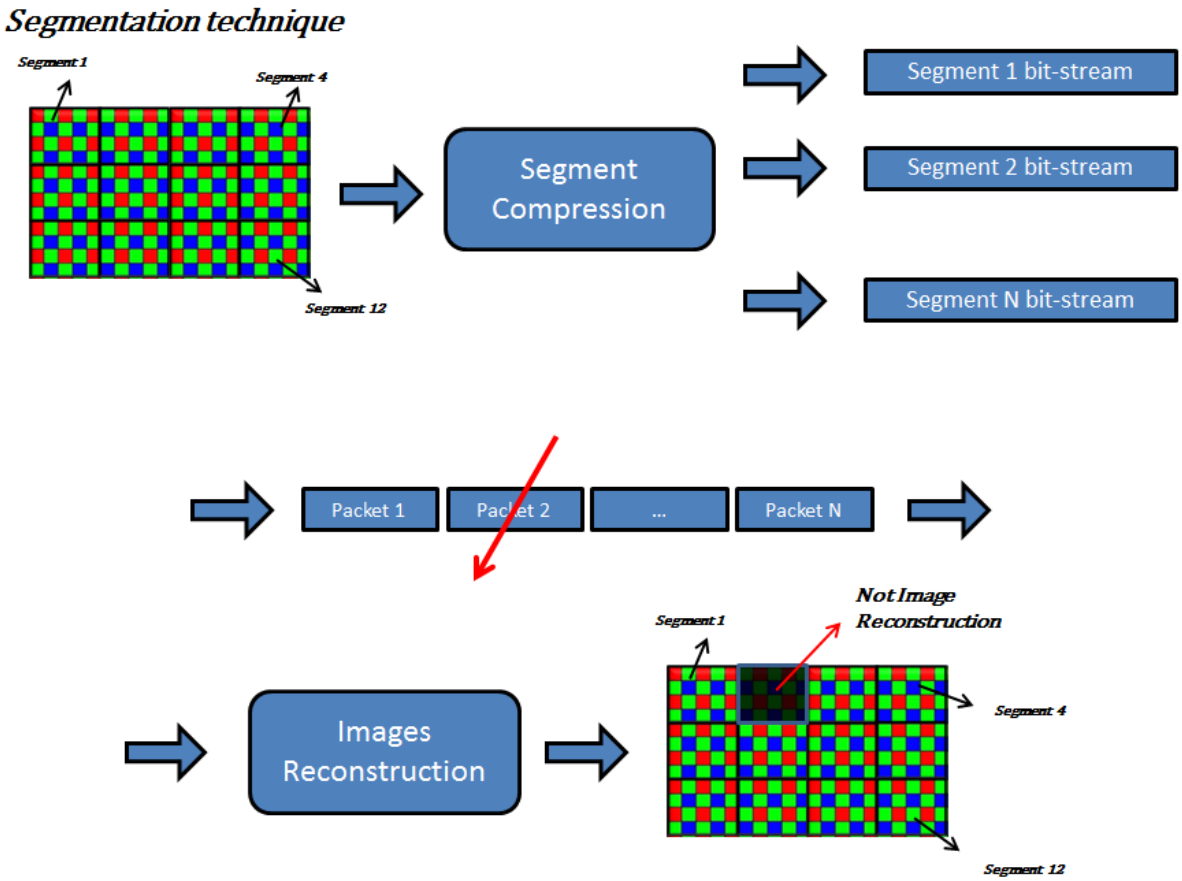


Figura 14- Técnica de Segmentación

El método consiste en dividir la imagen en pequeños segmentos con el fin de aplicar el algoritmo de compresión en cada uno de los segmentos. Cada segmento será enviado como un paquete único y por tanto si existe alguna pérdida durante la transmisión, solo un pequeño trozo de la imagen se perderá, y no toda la imagen.

La nueva tasa de aceptación será de:

$$Arate = 1 - \frac{n_l}{N} \quad such\ us \quad n_l = N°\ of\ lost\ packets \ \ and \quad N = N°\ packets \qquad (10)$$

2.4 Implementación

2.4.1 Introducción

El objetivo del proyecto no solo consiste en la implementación del algoritmo de compresión sin pérdidas, sino que dicho algoritmo debe ser implementado sobre el dispositivo hardware del nano-satélite. La implementación que se ha llevado a cabo se trata de una implementación software en el lenguaje de programación C. La placa del satélite, la cual ejecutará el código fuente se denomina Ninano y está basada en la tecnología "System On Chip" SoC. En concreto la placa utiliza la tecnología Xilinx ZYNQ® All Programmable SoC que incluye un módulo FPGA y un Dual-core ARM® Cortex™-A9 processor con tecnología NEON™ DSP/FPU, es decir un procesador vectorial.

Dentro de la implementación, se deberá tener en cuenta los registros de memoria, posibles notificadores de error, los ratios de compresión y registros modificables por mensajes de tele-comanda para la modificación de los parámetros del algoritmo. Para el desarrollo y testeo utilizaremos la plataforma SDK de Xilinx, el cual nos da variadas funcionalidades al poder interactuar con la tarjeta de manera simple.

2.4.2 Ninano-Board

Esta sección se centra en realizar un pequeño resumen sobre las especificaciones de la tecnología "All-Programable-System-On-Chip" SoC utilizada en el proyecto, es decir, la tecnología ZYNQ®. En concreto la placa utilizada para el proyecto se muestra en la Figura 15.

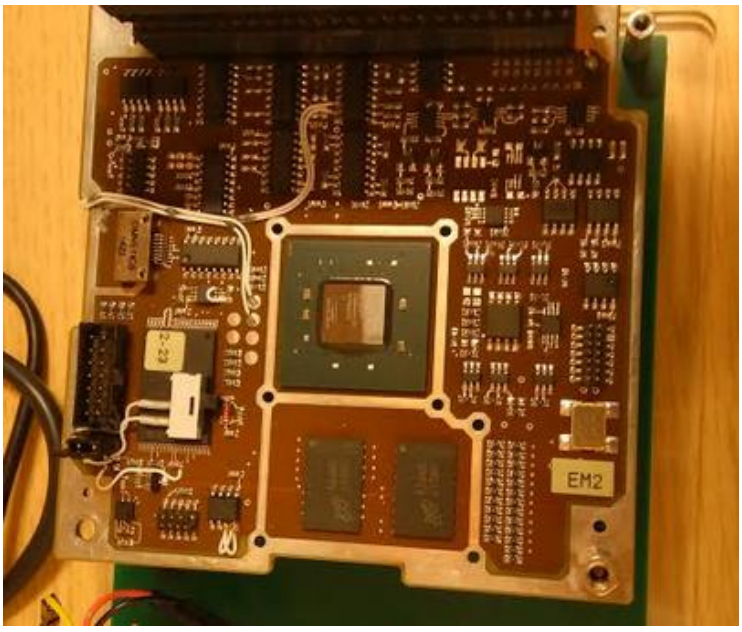


Figura 15- Ninano Board

La tecnología SoC implica que mediante la utilización de un único chip de silicio, seamos capaces de generar las funcionalidades de un sistema entero. Así esta tecnología equivale a distintos dispositivos tecnológicos, los cuales deben ser interconectados para generar dicho sistema. Se trata de una solución de bajo coste, rápida, con métodos seguros de transmisión de información, de pequeño tamaño y de bajo consumo. El único problema de esta tecnología es la falta de

flexibilidad, la cual es suplida con la nueva generación de dispositivos "All-Programable-System-On-Chip". ZYNQ® es un ejemplo de esta tecnología. De hecho, dicha plataforma combina un dual-core ARM® Cortex-A9 processor con una traditional Field Programmable Gate Array (FPGA) logic fabric. En ZYNQ®, el ARM® Cortex-A9 es un procesador capaz de ejecutar sistemas operativos como Linux, mientras que el dispositivo programable está basado en la arquitectura Xilinx 7-series FPGA. La arquitectura es completada por el estándar AXI para interfaces, el cual provee de gran ancho de banda, bajo retardo entre las conexiones de los distintos dispositivos de la placa.

En la Figura 16 se muestra un esquema simplificado de la tecnología ZYNQ®.

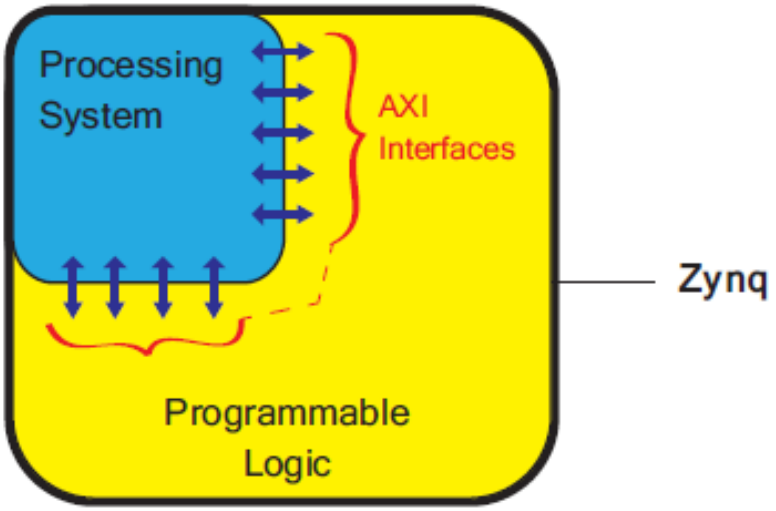


Figura 16-Modelo Simplificado Arquitectura ZYNQ®

2.4.3 Xilinx SDK

El Xilinx Software Development Kit (XSDK) es un entorno de desarrollo integrado para crear aplicaciones embebidas para el dispositivo Xilinx ZYNQ®. El SDK es la primera aplicación IDE para el desarrollo de homogéneos o heterogéneos diseños, además de poder realizar el testeo mediante el sistema de debugger que viene integrado. Se trata de una plataforma basado en eclipse y que nos permite desarrollar códigos para C/C++. La plataforma al igual que Eclipse te permite seleccionar distintas opciones de compilación, linkear librerías, así como realizar la compilación y ejecución de los códigos en hardware.

Como se ha comentado, el XSDK dispone de una plataforma de testeo la cual se denomina el Xilinx® System Debugger, el cual te permite conocer en cada instante lo que tu programa está ejecutando en cada momento. Se puede seleccionar breakpoints que te permite parar tu programa en ejecución, comprobar las variables de estado, comprobar la pila y los registros de memoria del dispositivo.

Por ultimo comentar que el Xilinx® System Debugger viene provisto con una Command-line Interface (XSDB). Dicha interfaz es de fácil uso, interactiva y la cual nos dará funcionalidades como la de escribir dentro de la memoria un archivo binario como el de la imagen fotografiada por la cámara. Además una vez realizada la compresión, gracias a la misma interfaz seremos capaces de leer los datos binarios generados, copiarlos en un archivo de texto y compararlos con los datos obtenidos en simulación. En caso de que los datos sean iguales, querrá decir que nuestro sistema en placa funciona de manera correcta.

2.5 Resultados

A continuación, en la Tabla 1 se muestran los resultados obtenidos para el algoritmo implementado en relación a la tasa de compresión. La información de la tabla está descrita en Bits/Píxel, es decir, el número medio de bits necesarios para codificar cada uno de los píxeles. Si queremos obtener el ratio de compresión debemos dividir el número inicial de bits necesarios para codificar la imagen con el número obtenido tras el algoritmo.

Existen dos subconjuntos de imágenes las cuales se pueden observar en el Anexo. Las primeras son imágenes CFA con mucho colorido, en las cuales pueden darnos una idea de cuál serán las tasas de compresión en el caso de que las imágenes tengan poco ruido y una alta variedad de colores. El segundo subconjunto de imágenes ha sido adquirido por el telescopio Hubble, en dichas imágenes hemos añadido ruido aditivo y multiplicativo. Se tratan de imágenes con escasa variedad de colores y donde el elemento dominador es el negro.

Tabla 1- Resultados de compresión

Bits/Píxels	***Images***	***Algorithms***			
		Efficient Algorithm- Block Encoder			
		Predic 3	***Predic 3 Spec***	***Predic 2***	***Predic 1***
8 bits	***Toulouse 1***	4.072174	4.014709	5.402557	5.779388
	Toulouse 2	3.431503	3.389282	4.805649	5.376251
	Toulouse 3	4.106171	4.025955	5.317749	5.781464
12 bits/píxel +noise	***Hubble***	6.825058	6.956726	6.952332	6.769653
	Hubble1	5.434875	5.558762	5.463791	5.363541
	space	6.678925	6.854645	6.750198	6.241928
	milky	6.86505	6.935089	6.994675	6.809525

Como se observa, cuando el algoritmo utiliza el tercer predictor la tasa de compresión en el primer subconjunto de imágenes consigue una tasa de compresión muy superior a cualquiera de los otros dos predictores. El motivo es que debido al alto colorido de las imágenes, la redundancia entre colores es notable y podemos explotar dicha redundancia para la estimación de los píxeles y aumentar la tasa de compresión. Como se puede observar la tasa de compresión llega a alcanzar en algún caso un ratio de 2.

Respecto al segundo conjunto de imágenes, en las cuales domina la componente oscura y existe una alta componente de ruido el mejor predictor es el primero. Como se observa en algunos casos la tasa de compresión puede ser un poco mayor de 2. Esto es completamente natural debido a la alta componente de ruido, el cual al ser aleatorio es completamente impredecible.

La Tabla 2 muestra los tiempos de ejecución del algoritmo para cada uno de los predictores. Como se comentó con anterioridad la tasa de adquisición para cada una de las imágenes era de

una imagen por cada 15 segundos. Por tanto el objetivo consistía que el algoritmo pudiera ser llevado en un tiempo bastante menor de 15 segundos. En caso contrario, la memoria física podría llegar a ser saturada en algún momento. De esta manera el algoritmo se llevará a cabo y los registros donde se almacena la imagen original podrán ser superpuestos por una nueva imagen a comprimir.

Tabla 2- Tiempo de ejecución del algoritmo

Efficient Algorithm- Block Encoder			
Predic 3	***Predic 3 Spec***	***Predic 2***	***Predic 1***
2s	2s	2s	2s

Como se observan tanto en los ratios de compresión como en los tiempos de cómputo las prestaciones se adecúan a los requerimientos de la misión y por tanto el proyecto queda realizado y testeado.

2.6 Líneas futuras

Dentro de las líneas futuras, las cuales se comenzaron a realizar durante las últimas 3 semanas de las prácticas debido a una buena organización logística. El algoritmo de compresión desarrollado dentro de la plataforma SDK va a ser implementado dentro de un Hypervisor. El motivo del uso de un Hypervisor se debe a que el algoritmo de compresión no va ser el único que será ejecutado por el microprocesador y por tanto necesitamos un software que permita supervisar la compartición de los recursos del procesador.

En concreto se va a utilizar el Hypervisor XtratuM, software creado por ingenieros de la universidad de Valencia, el cual es un software que permite dividir en slot temporales y espaciales los recursos del microprocesador. De esta manera en cada slot se llevará a cabo las operaciones de cada uno de los algoritmos, compresión, adquisición de la imagen, software de vuelo….

Uno de los principales problemas a la hora de utilizar un Hypervisor, es la compilación cruzada, debido a que no podemos realizar la compilación de nuestro programa directamente sobre el hardware que estamos utilizando sino sobre otra plataforma suministrada por el Hypervisor, la cual debe ser configurada mediante una máquina Linux.

Antes de finalizar las prácticas pude ser capaz de reinscribir el código SDK y adaptarlo a las restricciones del Hypervisor. Se comprobaron que las tasas de compresión eran similares a las obtenidas en simulación o en el SDK. Sin embargo, los tiempos de ejecución estaban muy por encima de lo esperado y no cumplían las especificaciones de tiempo demandadas. El fallo es debido a una configuración errónea del Hypervisor, el cual actualmente no ha sido resuelto. Por tanto, entre las líneas futuras del proyecto se encuentra que el ingeniero de sistemas embebidos encuentre la configuración correcta para XtratuM y que el Hypervisor sea capaz de coordinar de manera segura cada uno de los códigos. Por coordinar me refiero a ser capaz de detectar errores, utilizar la información de estado, pasar la información de tele-comanda a cada uno de los algoritmos además de realizar la repartición de los recursos del sistema de manera apropiada.

2.7 Conclusiones

A lo largo de este proyecto [1] y en concreto sobre este resumen de memoria de proyecto se ha presentado la metodología seguida para llevar a cabo el desarrollo y la implementación de algoritmos de compresión de imagen que será utilizado para la misión Eye-Sat. El algoritmo será implementado sobre la placa central del satélite, la cual se denomina Ninano. Ninano es una tecnología System-on-Chip (SoC), la cual usa un Xilinx ZYNQ® All Programmable SoC que incluye un módulo FPGA y un Dual-core ARM® Cortex™-A9 processor con tecnología NEON™ DSP/FPU, es decir, un procesador vectorial. Como el algoritmo de compresión no será el único software ejecutado por el procesador, el proyecto tiene previsto utilizar un Hypervisor para coordinar los recursos del procesador. El Hypervisor seleccionado es XtratuM.

Las imágenes adquiridas en la misión deben ser comprimidas debido a que tenemos un límite de visibilidad entre el segmento suelo y el satélite. Por ello una compresión cercana a 2 nos permitiría envía el doble de imágenes en el mismo periodo de tiempo. Existe tres tipos de imágenes a comprimir, la primea de ellas se trata de imágenes de negro que nos servirá para calibrar los sensores. El segundo subconjunto de imágenes es una secuencia de imágenes infrarrojas adquiridas por un sensor con un filtro espectral adecuado. El tercer subconjunto de imágenes se trata de imágenes RGB que serán adquiridas por el mismo sensor CFA pero sin ningún filtro en particular. El patrón CFA será en matriz de Bayer.

Para desarrollar los algoritmos de compresión para los diferentes tipos de imágenes, hemos tenido en cuenta las características y los requerimientos de la misión. De hecho, los requerimientos más importantes consisten en que el algoritmo de compresión de imágenes sea sin pérdidas, que sea capaz de realizar la compresión dentro de las especificaciones de tiempo y que además tengo una buena tasa de compresión, dentro del hecho que estamos implementado un algoritmo de compresión sin pérdidas.

Como tenemos secuencias de imágenes se han probado, implementado y testeado dos tipos de algoritmos, algoritmos de compresión de imagen sin pérdidas y algoritmos de compresión video sin pérdidas. Los cuales se basan en el estándar de compresión híper-espectral CCSDS-123. Definitivamente después de realizar el testeo optamos por llevar a cabo la implementación final de un algoritmo de compresión de imagen. El motivo principal de descartar los algoritmos de compresión video se debe a la alta complejidad generada por el cálculo de vectores de movimiento, además de no saber muy bien cómo implementar un sistema de segmentación para evitar las retrasmisiones.

El algoritmo de compresión de imagen seleccionado tiene como propiedad principal la baja complejidad respecto a la computación. El codificador entrópico que utiliza el algoritmo se trata del codificador adaptativo por bloque de estándar CCSDS-123, el mapping y el predictor han sido desarrollados enfatizando la baja computación y explotando la redundancia espectral que podremos encontrar en ciertas imágenes, como las adquiridas por el sensor CFA con patrón en matriz de Bayer. El predictor tiene varios modos de utilización dependiendo el tipo de imagen y reduce de manera óptima el número de accesos a memoria y el número de operaciones utilizadas. Un sistema de segmentación es utilizado con el fin de combatir el problema de los errores de transmisión, los cuales podrían generar la pérdida de recursos y de ancho de banda.

Los resultados obtenidos en términos de compresión muestran buenos resultados, consiguiendo en varios ejemplos ratios de compresión superiores a 2. Respecto a los resultados obtenidos en el análisis de tiempo, los resultados del algoritmo son muy alentadores, consiguiendo un gran margen de tiempo y por supuesto adecuándose a las especificaciones demandadas por la

misión. Dichos resultados, al igual que el testeo, fueron obtenidos mediante el uso de la plataforma de desarrollo SDK, la cual nos facilitó un medio para poder comprobar, ejecutar y desarrollar el algoritmo final, el cual será implementado dentro del Hypervisor XtratuM.

Además, y aunque no estuviese fijado dentro del proyecto, se llevó a cabo la implementación del código sobre el Hypervisor, obteniendo éxito en la implementación. Por otra parte y debido a una mala configuración del Hypervisor los tiempos de ejecución no eran coherentes. Entre las próximas actividades a realizar, se deberá encontrar una configuración apropiada para el Hypervisor en la cual los tiempos de cómputo estén dentro del margen y de las demandas de tiempo de la misión.

Para finalizar, es interesante de comentar que aunque las prestaciones de compresión muestren buenos resultados, realmente no sabemos con exactitud que ruido va a estar presente en las imágenes adquiridas y por tanto el rendimiento del algoritmo podría variar dependiendo de las condiciones reales que encontremos *ahí fuera*….

Referencias

[1] L. Ortega, ": [Eye-Sat Student Nanosatellite] Data Compression Manager."

[2] *Lossless Multispectral & Hyperspectral Image Compression Recommendation for Space Data System Standards, CCSDS 123.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 2012*.

Anexo

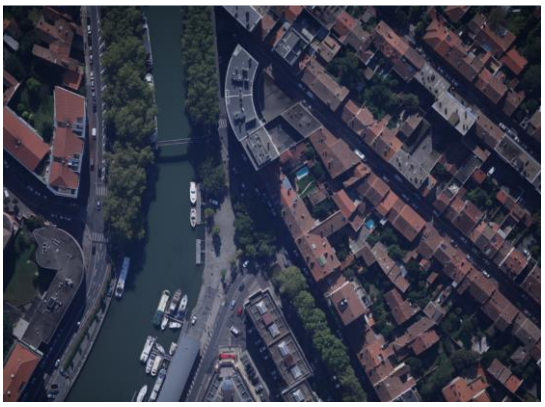


Ilustración 1- Toulouse 1

Ilustración 2- Toulouse 2

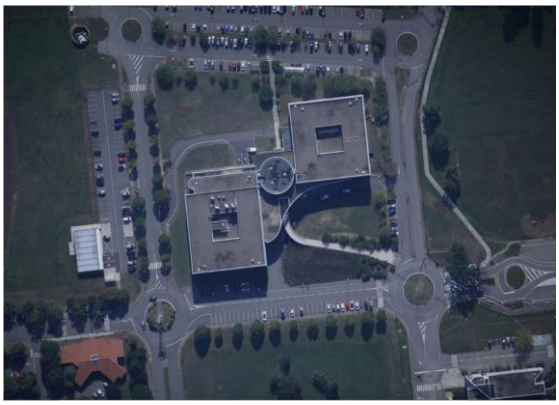


Ilustración 3- Toulouse 3

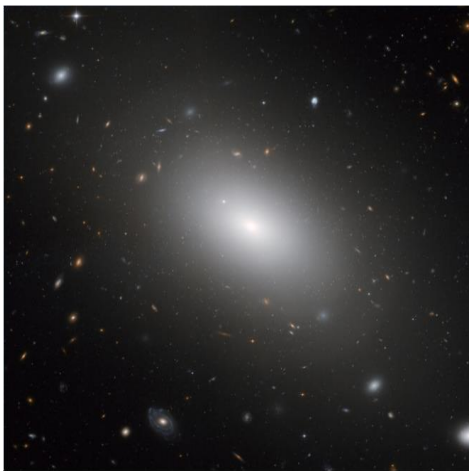


Ilustración 4- Hubble

Ilustración 5- Hubble 1

Ilustración 6- Milky Way

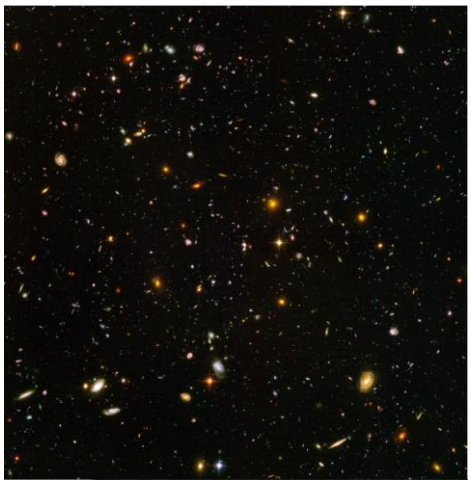


Ilustración 7- Space

Master Thesis

Title: [Eye-Sat Student Nanosatellite] Data Compression Manager.

Author

Lorenzo Ortega Espluga

Director

Roberto Camarero Rodríguez

March 2016 – September 2016

Summary

Eye-sat is triple Cubesat (nanosatellite) project, managed by the CNES (Centre National d'Etudes Spatiales) and developed principally by students of French engineering schools.
The purpose of this small telescope is not only to provide the opportunity for technological demonstrations, but also to obtain global color and infrared images from the Milky Way and to study the intensity and polarization of the solar light scattered by interplanetary dust particles.

The mission requirements demand the development of a low-complexity lossless compression algorithm for CFA (Bayer) images on-board the satellite. As a member of the Consultative Committee for Space Data Systems, CNES has selected the CCSDS-123.0-B Standard to fulfill those mission requirements.

The algorithm software implementation and optimization has been performed on a Xilinx Zynq® All Programmable SoC which includes a programmable logic module (FPGA) and a Dual-core ARM® Cortex™-A9 processor with NEON™ DSP/FPU Engines.

“Information is the resolution of uncertainty”
Claude Shannon

Acknowledgments – Remerciements - Agradecimientos

A todas aquellas personas con sed de conocimiento y deseos de superación, que leen hoy estas páginas y premian el esfuerzo de este trabajo.

Agradezco en primer lugar a mis profesores, tanto de la universidad de Zaragoza como del ENSEEIHT por no solo dotarme de los conocimientos necesarios para llevar a cabo este proyecto, sino por brindarme la oportunidad de formarme como persona.

Superlativas gracias a Roberto Camarero por brindarme la oportunidad de poder realizar este proyecto y ayudarme en cualquier duda planteada, a todos mis compañeros de laboratorio, Clement, Elise, Jeremie, Martin y Thibaud así como a todo el departamento de informática de abordo y equipamiento por todos los buenos momentos, consejos y por no solo por ayudarme con el proyecto, sino también a mejorar mi francés.

Por último, agradecéroslo a vosotros, Papas, vuestro esfuerzo, superación y valentía me han dado la fuerza para continuar adelante, en la búsqueda de lo que de verdad disfruto. Gracias por todas oportunidades, enseñanzas y por vuestro incondicional apoyo.

Summary

Part 1: Internship description and components

1.CNES

1.1.INTRODUCTION

Under the Ministries of Research and Defense supervision, the National Centre for Space Studies (CNES) is a Public Establishment for Industrial and Commercial goals (EPIC). The agency was created in 1961 under the leadership of General De Gaulle, to design, develop and propose the French government space strategy.

Thus, CNES is responsible for developing the national France space programs. Through these, the French government intends to preserve national sovereignty in science and aims to take an active part in the most advanced space research. As such, CNES is developing future space systems, handles the most current technologies and guarantees an independent space access.

Seeking how to introduce space technology at social service to, not only provide current solutions, but also to anticipate future developments. Moreover, the Agency knows how to deal with industrial and commercial partners, which are essential in current space policies.

Heavily involved in European Space Agency (ESA) programs, where France is one of the main contributors, CNES has a great role upon European Union space policy, such as projects development as completed missions.

1.2. MISSIONS

CNES research and development policy is built concerning five main axes:

1.2.1. SPACE ACCESS

CNES mission is to provide Europe space access from the spaceport of Kourou Space Center (French Guiana) through the use of three operational launchers:

- Vega: Launcher for the small satellite market (weight below 1.5 tons), in low orbit (altitude up to 2 000 km)

- Soyouz: Russian-made launcher, intermediate satellites (less than 3 tons mass) and in intermediate orbit (a few thousand kilometers above)

- Ariane 5: Heavy launcher for two heavy satellites (total weight less than 10.5 tons) ,on all orbits.

1.2.2.SCIENCES AND FUTURE

CNES has participated in many projects in connection with space research, in order to increase knowledge, study the elements which constitute it and understand the origin of life in the universe. The main concerned areas are: Astronomy and astrophysics, solar system and the Sun, heliosphere and magnetosphere, fundamental physics, exobiology, microgravity.

1.2.3.EARTH OBSERVATION

Many projects involve Earth observation, for better understanding and therefore better Earth protection. Indeed concerning this topic in one hand, Merlin satellites projects are found, which tracks the evolution of greenhouse gases in the atmosphere; in the other hand Venμs, which handles of monitoring the evolution of Earth vegetation.

1.2.4.DEFENSE

CNES, as under the Defense Ministry ward, find essential to regain the army among its missions. Helios 2 is an example of a satellite, which supplies unique image information to the government.

1.2.5.GENERAL PUBLIC

CNES is also taking action in the "general public" telecommunications sector by adapting space solutions to European citizen's needs:

- Telecommunications: Sample THDSAT telecommunications satellite for broadband internet connection
- Positioning / Navigation: Ex: Galileo as project for GPS navigation
- Location data collect : ARGOS, for the collection of data to protect the environment
- Search and Rescue: Cospas Sarsat: international program to detect and localize distress marine, aeronautical or ground signals, (35 000 lives rescued since 1992)

1.3. DIFFERENT CENTERS

Currently CNES is composed of around 2,500 men and women (35%); engineers, researchers, administrative and operating personnel in four centers in France. These centers are:

1.3.1.PARIS LES HALLES CENTER

Headquarters. Located in Paris, near Les Halles, it handles the administrative functions of the agency. It ensures the development and promotion of CNES policy in conjunction with the ministries. It defines the strategic directions of technical centers and relations with external partners.

1.3.1.PARIS DAUMESNIL CENTER

Located in Daumesnil, this site hosts the Launcher Directorate (DLA) which leads all developments of the Ariane program, commissioned by the European Space Agency (ESA). He follows all project phases, since launch production until their orbit collocation. In parallel, it researches and develops new concepts for launchers and propulsion systems.

1.3.2.GUYANAIS SPATIAL CENTER (CSG)

As Russian spaceport Baikonur (Kazakhstan) or the John F. Kennedy Space Center (United States, FL), the Kourou (French Guiana) is the spaceport Western Europe. It is dedicated to launch the Ariane launcher as well as the Soyuz and Vega launchers. CSG manages operations to prepare the embedded Payload on satellites, launches, track the orbiting satellite and coordinates the necessary resources to receive and process at ground station all the information sent by the launcher.

1.3.3.TOULOUSE SPATIAL CENTER

CST participates in numerous scientific projects, as the number of departments that host is important. It handles observation programs such as, Spot, Helios, Pleiades and Argos. In partnership with industry and scientific research laboratories, it develops complete space systems, from conceptualization to the operational service. To sum up, it leads the orbiting satellite operations and the station-keeping procedures.

1.4.MORE THAN SPACE

To carry out its tasks, CNES surrounds itself with industrial and scientific partners to reinforce and complement their capabilities, both financial and intellectual. French agency has been able to build for more than 50 years a strong and competitive industrial base, handling technical projects and ensuring its partners an attractive place in the global market. His three most significant industrial partners are Thales Alenia Space, Airbus Defence and Space and Arianespace.

Moreover, CNES has developed agreements with academic and space laboratories. Providing for around 500 trainees Thesis project, PhD or post-PhD. This reflects not only a policy of openness and acceptance, but also an active involvement in training our future engineers.

Finally, CNES works hand-in-hand with space agencies around the world (ESA, NASA, JAXA, etc ...). Indeed, most CNES programs are in cooperation either within the ESA or in a multilateral framework. Half of the CNES budget is devoted to the European Agency programs, while the rest is devoted to projects developed in European cooperation or not. Among these works we can mention SPOT (France, Belgium, Sweden), JASON (France, USA), or Pleiades (France, Sweden, Belgium, Spain, Italy and Austria).

2.INTERNSHIP GOAL

2.1.CONTEXT

Managed by CNES but developed by students, Eye-Sat is a nanosatellite project for an astronomy mission. The internship purpose concerns the project continuity upon the detailed design and development in both nanosatellite and ground segment: Phases C / D.
The internship will take place in the service **« informatique bord et équipements »**, which is under direction of « techniques véhicule, architecture & intégration » (TV/IN). This service, composed of 13 agents, is in charge of developing equipments for on-board data processing: processors, data bus, mass memories, encoders, compressors.

2.2.GOAL

Inside the Eye-Sat context, and to maximize the capabilities of the mission, CNES wishes to study the implementation of image compression methods within the board processor. The recommended methods will favor low complexity, provided that performance will be achieved. Among the proposed methods, two predictive compression methods will be studied; both are CCSDS standards for single-band image compression (CCSDS 121.0-B) or multi-band (CCSDS 123.0-B). The adequacy of a video compression standard, as acquired data is a set of image (sequences), will also be studied.

2.3.OBJECTIVES AND CNES EXPECTATION

• State of the art of the best available algorithms, which may adapt to both the sensor and the processing target.

• Performance evaluation of different algorithms (over reference images) to validate their relevance: performance vs complexity (cost calculation) trade-off.

• Compilation and validation of the algorithm chosen on the target (ARM®). Optimization coding phase for the architecture (ARM® NEON ™) may be necessary.

2.4.INTERNSHIP STEPS

1. Literature review (State of the art) on compression methods and processing in vector processing units. A set of algorithms tailored to the mission needs to be chosen and tested during this phase.

2. Use and manage of image viewing tools and performance measurement.

3. Preliminary study of chosen algorithms performances, with possible changes within the algorithms for better data adaptation (Bayer matrix).

4. Analysis of performance in both, compression efficiency and complexity.

5. Implementation of the chosen algorithm (adapted from code and compilation) on the target of the mission (ARM® heart of ZYNQ® component). Perform compression tests

6. Code optimization to fit it on the target architecture (NEON ™: SIMD vector processor)

7. Obtention of an embedded compression function optimized for ARM® and validating (exhaustive testing).

8. Writing of the report and oral result presentation within CNES.

3. EYE SAT

3.1. INTRODUCTION: A STUDENT NANOSATELLITE

EyeSat [22] is a nanosatellite project designed and developed by students from several French universities (ISAE-SUPAERO, IUT de Cachan, University Paul Sabatier Toulouse) and supervised by CNES. Its launch is scheduled for June 2017 as a secondary passenger on a Soyuz flight. Which will be its mission, what will it carry out, why, and for what purpose?

3.2. MISSION, CHALLENGES

3.2.1. SCIENTIFIC CHALLENGES

EyeSat mission goal consist on observing and characterizing the zodiacal light: low light corresponding to the sunlight diffusion by interplanetary dust and which is hardly visible from Earth, even in good conditions. Therefore, EyeSat is the first mission dedicated to this topic and its purpose is improving the knowledge about photometry and polarization in 4 different spectral bands by using 4 filters: blue, green, red and near infrared, upon the rough light.

Figure 1- Zodiacal light

EyeSat data acquisition should be used by Euclid mission, whose launch is scheduled in 2020. Its mission will consist on exploring the dark universe to have a better understanding of the role of dark energy in universe expansion.

3.2.2. TECHNICAL CHALLENGES

EyeSat has other plans than pure science; its second aims consist on a technology demonstration of several equipment devices, which have never been used on a space project. Therefore, depending on results and studies leaded by CNES with industrial partners, it will be possible to qualify in space environment for further space missions, certain technologies never used on space before. Below, a representative 3D view of nanosatellite and all the facilities that will ship:

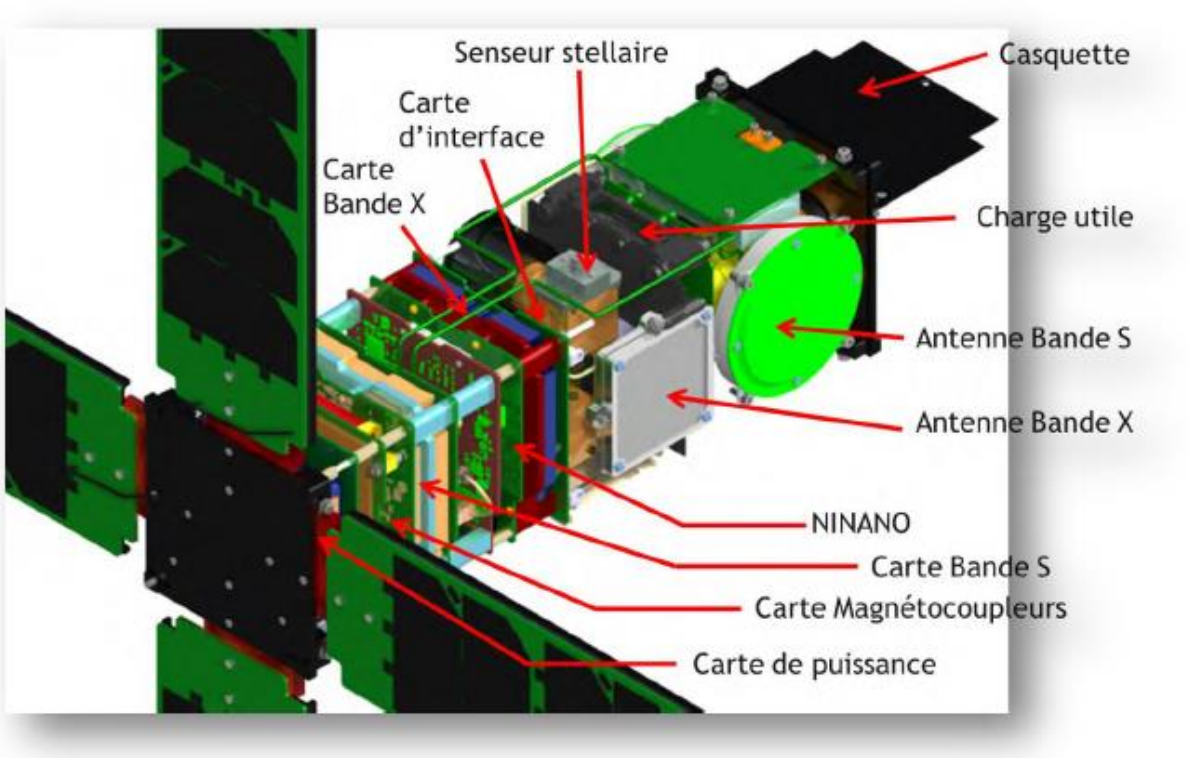


Figure 2- Eye-Sat Nanosatellite

Among these facilities we find:

➢ Power board, batteries and related satellite solar panels, electric source of satellite

➢Magneto-Card, star sensor and GPS (optional) for the direction and guidance of satellite (AOCS: Attitude Control System and Orbit)

➢The satellite is controlled over the three axes by using reaction wheels and the attitude is estimated from the measures of the star tracker

➢ Card and High Frequency antennas: X-band (8-12 GHz) and S-band (2-4 GHz) for communication with the ground: telemetry transmission and telecommand reception.

➢ IRIS System ("camera" of the satellite) and filter wheels for acquistion

➢ Ninano Board: board computer

Up to now, there are five technology demonstration axes, concerning the aforesaid equipment.

➢ IRIS detector, which uses an ever flown (in a space project) color CMOS technology.

➢ The Radiofrequency system using NanoTTC map in S-band for the telecommand (satellite monitoring) and NanoTMHD Band-X card for telemetry (scientific data payload)

➢ Flap deployment mechanism (solar panels) by the use of self-deployed composite hinges and self-locked

➢ The onboard computer, Ninano card, consists on a Xilinx Zynq7030 component, with two ARM Cortex-A9 cores and programmable logic part, offering greater computing power.

➢ Use of LVCUGEN software (Logiciel de Vol Charge Utile GENérique), TSP hypervisor (Time and Space Partitioning) and XtratuM for hardware resources partitioning (memory, CPU, IOs)

3.2.3. HUMAN CHALLENGES

A multitude of contributors are involved the project, up to now we find:

Space professionals:

- Project manager: Alain Gaboriaud

- System responsible: Antoine Ressouche

- Support engineering teams dedicated to the instrument, AOCS mechanisms, structure, thermal, avionics, energy board, flight software, radio frequency, data compression, ground stations. The teams give their time to guide students working on the project.

Students in training, learning within CNES, from:

- IUT de Cachan
- University of Toulouse Paul Sabatier
- ENAC
- ISAE-SUPAERO
- Other schools and universities

The CNES goals concerning the involvement of students on space projects are many:

➢ Encourage the space promotion by offering to students from universities and engineering schools the development of space systems:

- Nanosatellites between 1 and 10 kg, Cubesat type 1U and 3U (1U: 10cm cube)
- Scientific instruments to measure Earth's environment parameters, to produce images of the planet in different spectral bands, to locate vehicles.
- Ground Segment (telemetry receiving stations and remote controls transmit, control centers, mission centers).

➢Propose and validate new technologies (materials, sensors, components, ASIC, propulsion, attitude control, computer, electrical architecture, communication,…)

➢ Educate students in the logic of space project development and implementation (project management, cooperation with universities or schools from other countries, development plan, launch, operation data,...).

➢To promote scientific teaching with a strong experimental dimension in different areas of space (mechanical, thermal, avionics, attitude control, power systems, propulsion, control center, image processing...)

➢Making a link between students and space laboratories for scientific, technological and instrumental activities.

3.2.4. COMMERCIAL AND ECONOMIC CHALLENGES

The project objectives mainly concern scientific and human intentions, therefore the economic challenge is not predominant. If its financial benefits are not assessable, we can however look at the overall cost of the project EyeSat, which it is around € 3 million (external expenditure, HR Students, HR supervisors CNES). This is 50 to 1000 times lower than a conventional satellite mission (depending on the type of satellite)

Like more "classical" CNES projects, the economic benefits are difficult to quantify in contrast to scientific and technical benefits:

- Direct Impacts: produced by the various scientific publications, theses and scientific advances

- Ripple effects: international trade of data, supporting industry, graduate employment, knowledge of new space technologies (including miniaturization).

Part 2: Theory requirements

1.INTRODUCTION TO THE DATA COMPRESSION

Summary

Data compression has been an enabling technology for the information revolution, and as this revolution has changed our lives, data compression has become a more and more ubiquitous, if often invisible, presence. From mp3 players, to smartphones, to digital television and **Space applications**, data compression is an integral part of almost all information technology. This incorporation of compression into more and more of our lives also points to a certain degree of maturation and stability of the technology.

1.1. INTRODUCTION

In the last decade, we have been witnessing a transformation concerning on how we communicate. This transformation comprises since the growing internet, the "boom" and development of mobile communications… until the increasing importance of Images communication upon space researching. Data compression technology is one of the most important aspects concerning multimedia revolution due to It would not be practical upload, download or in general terms transmit images, video, audio… if it were not for data compression algorithms. Multimedia revolution involves a resource quality improvement, which results in increasing storage needs as well as higher bandwidth requirements.

So what is data compression, and why do we need it? Most of you have heard of JPEG and MPEG, which are standards for encoding images, video, and audio. Data compression algorithms are used in these standards to reduce the number of bits required to represent an image or a video sequence or music. So, basically, data compression handles of representing the information in a compact form. To create this compact form we focus on how information has been represented in the digital world and we delete all the information which is considered unnecessary.

Given the explosive growth of data that needs to be transmitted and stored, why not focus on developing better transmission and storage technologies? Of course, this is happening, but it is not enough. There have been significant advances that permit larger and larger volumes of information to be stored and transmitted without using compression; however the huge increase of data generation is so big that we could not store all these data without compression techniques.

1.2. COMPRESSION TECHNIQUES

The task of compression is carried out through two procedures [1], the first one consists on an *encoding* algorithm that takes a message and generates a "compressed" representation (hopefully with fewer bits), and the second, which consist on a *decoding* algorithm that reconstructs the original message or some approximation of it from the compressed representation. These two components are typically tightly tied together since they both have to understand the shared compressed representation.

We distinguish between lossless algorithms, which can reconstruct the original message exactly from the compressed message, and lossy algorithms, which can only reconstruct an approximation of the original message. Lossless algorithms are typically used for text, and lossy for images and sound where a little bit of

loss in resolution is often undetectable. Here we underline that lossless algorithm can be used to compress images, video …, in fact these report has this goal.
Figure 3 shows a common compression scheme, especially a lossless compression scheme since the Original Message and the reconstructed message are exactly the same.

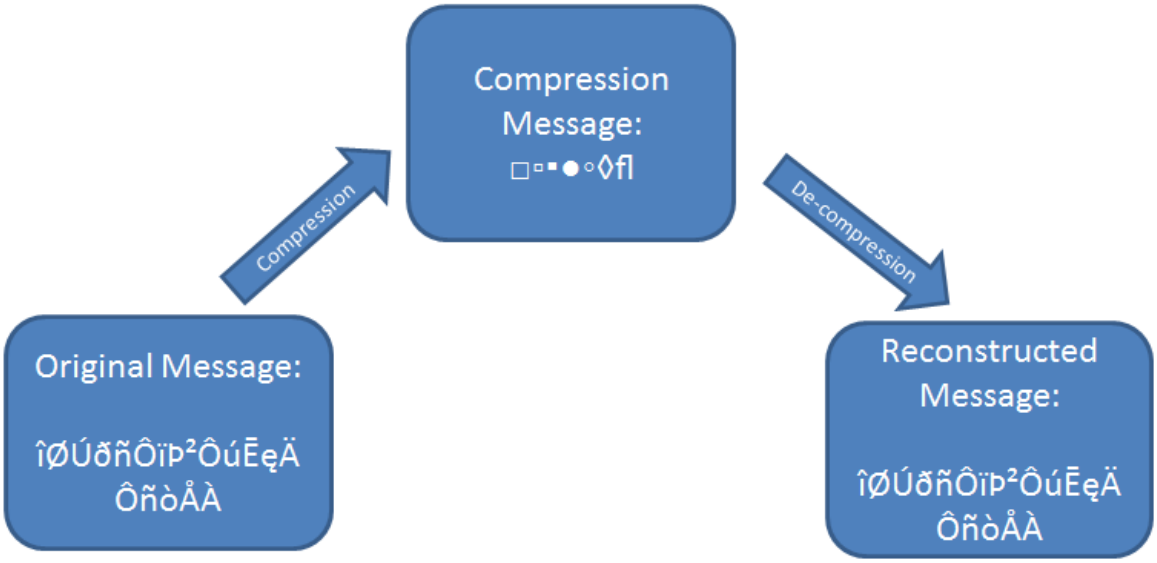


Figure 3- Compression scheme

1.2.1. LOSSY COMPRESSION

Lossy compression techniques involve some loss of information, and data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly. In return for accepting this distortion in the reconstruction, we can generally obtain much higher compression ratios than is possible with lossless compression.

There are many applications where this lack of exact reconstruction is not a problem. For instance, when storing or transmitting speech, the exact value of each sample of speech is not necessary. Depending on the quality required of the reconstructed speech, varying amounts of loss of information for each sample can be tolerated.

Similarly, concerning a video sequence reconstruction, the fact that the reconstruction is different from the original is generally not important as long as the differences become in annoying artifacts. Thus, video is generally compressed using lossy compression.

Once we have developed a data compression scheme, we need to be able to measure its performance. On the one hand to quantify the effectiveness of the compression technique we can use the compression ratio, by the other hand we should measure the quality of the decompressed information and to figure out if it is good enough.

1.2.2.LOSSLESS COMPRESSION

Lossless compression techniques, as their name implies, involve no loss of information. If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossless

compression is generally used for applications that cannot tolerate any difference between the original and reconstructed data.

Text compression is the mainly example for lossless compression since it is very important to reconstruct the original text due to very small differences can result in statements with very different meanings. Consider the next sentences example “Do *not* send money” and “Do *now* send money.” A similar argument holds for computer files and for certain types of data such as bank records.

Concerning the satellite domain, Data obtained often are processed later to obtain different numerical indicators of vegetation, physic calculations, and so on. If the reconstructed data are not identical to the original data, processing may figure out uncorrected results, jeopardizing all the efforts, which have been carried out to capture the pictures.

It is seemed that in many situations data compression is required; in fact; the majority of communication system use in one way or in another a compression system. **This project is one of those examples.**

1.2.3. MODELING

While reconstruction requirements may force the decision of whether a compression scheme is to be lossy or lossless, the exact compression scheme we use will depend on a number of different factors. Some of the most important factors are the characteristics of the data that need to be compressed. A compression technique that will work well for the compression of text may not work well for compressing images. Each application presents a different set of challenges. Independently of this fact,

Independently of this fact, the development of data compression algorithms can be divided into two phases. The first phase is usually referred to as **modeling**. In this phase, we try to extract information about any redundancy that exists in the data and describe the redundancy in the form of a model. The second phase is called **coding**. A description of the model and a “description” of how the data differ from the model are encoded, generally using a binary alphabet. The difference between the data and the model is often referred to as the **residual**.

Let’s use an example for re-organize the ideas.

Consider the following sequence of numbers:

27	28	29	28	26	27	29	28	30	32

The sequence does not seem to follow a simple law as in the previous case. However, each value in this sequence is close to the previous value. Suppose we send the first value, then in place of subsequent values we send the difference between it and the previous value. The sequence of transmitted values would be

27	1	1	-1	-2	1	2	-1	2	2

As it is shown in the sequence, the number of distinct values has been reduced. Fewer bits are required to represent each number, and compression is achieved. The decoder adds each received value to the previous decoded value to obtain the reconstruction corresponding to the received value. Techniques that use the past values of a sequence to *predict* the current value and then encode the error in prediction, or residual, are called *predictive coding* schemes.

In the framework of our project, predictive techniques are highly used in lossless image compression because the prediction error of each pixel value will be relatively low just for the fact that nearby pixels are going to be quite similar.

1.3.3 INTRODUCTION TO INFORMATION THEORY

Although the idea of a quantitative measure of information was known, the person who joined all the abstract concepts into what nowadays is called information theory was Claude Elwood Shannon, who was an electrical engineer at Bell Labs. The basis of Shannon theory starts by defining a quantity called self-information. We are going to suppose that we have an event A, which is a set of results of some random experiment. If we define *P(A)* as the probability of the event A will occur, then the self-information associated to the event A is given by

$$i(A) = \log_b \frac{1}{P(A)} = -\log_b P(A) \tag{1}$$

where usually the base b is 2 (binary language). Recall that $log(1) = 0$, and $-log(x)$ increases as *x* decreases from one to zero. Therefore, if the probability of an event is low, the amount of self-information associated with it is high; however, if the probability of an event is high, the information associated with it is low. Notice that even if we ignore the mathematical definition, the information concept has some intuitive sense.

Another property, which can be directly derived from the mathematical definition of information, is that the information obtained from the occurrence of two independent events is the sum of the information obtained from the occurrence of the individual events. Concerning image adjacent pixels, we cannot consider this hypothesis, as adjacent pixels usually have common information (noise of the sensor, structure and color of the images and so on).

$$\begin{aligned} i(AB) &= \log_b \frac{1}{P(A)P(B)} \\ &= \log_b \frac{1}{P(A)} + \log_b \frac{1}{P(B)} \\ &= i(A) + i(B) \end{aligned} \tag{2}$$

Finally, the last important definition is based on next premise. Suppose a set of independent event *Ai*, which are sets of results of some experiment ***S*,** such that

$$\bigcup A_i = S \tag{3}$$

where *S* is the sample space, then the average self-information associated with the random experiment is given by

$$H = \sum P(A_i) i(A_i) = -\sum P(A_i) \log_b P(A_i) \tag{4}$$

This measure is called the *entropy* associated with the experiment. One of the many contributions of Shannon showed that if we have an experiment where the events or symbols ***Ai*** from a set ***A***, then the entropy is a measure of the average number of binary symbols needed to code the output of the source. Moreover, Shannon showed that the best lossless compression scheme which we can do is to encode the output of a source with an average number of bits equal to the entropy of the source.

The goal of this brief introduction to information theory, it is to provide to the reader the basic knowledge required for understanding how our algorithm accomplished the encoding stage.

2.INTRODUCTION TO THE IMAGE COMPRESSION

Summary

In recent years, data compression theory becomes more and more significant for reducing the data redundancy to save more hardware space and transmission bandwidth. In fact, as we said in the section before, data compression or source coding is the process of encoding information using fewer bits. Concerning this section, we will focus on describing briefly how data compression scheme can be use upon image resources.

2.1. INTRODUCTION

The objective of image compression [2] is to reduce the redundancy (unnecessary information) of the image pixels and to store or transmit data in an efficient form. Figure 4 shows a general image compression block diagram. Underline that Quantification is only used in lossy schemes. Moreover, image compression goal not only consist on reducing the information, but also to provide after reconstructing an image as similar as the original image and concerning a lossless compression scheme, it requires a perfect image reconstruction.

General block diagram can be modified depending on the application and the way the image is acquired. In fact, a different block diagram is employed in our application because of the use of an image sensor including Bayer pattern matrix.

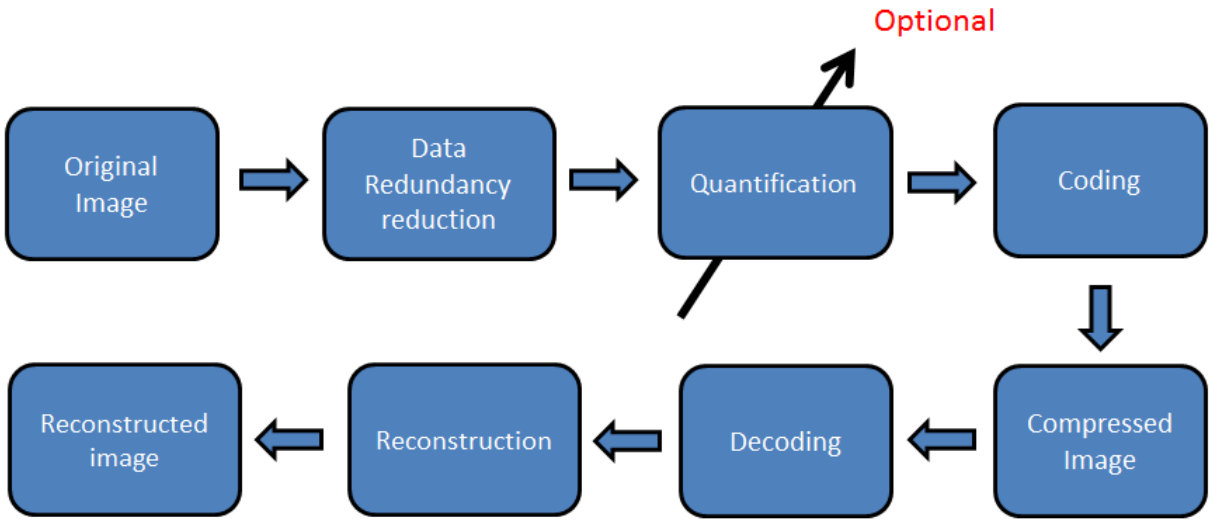


Figure 4- Image Compression Diagram

2.1.1. DIGITAL IMAGE

We cannot start to explain the methodology in image compression without explaining what a digital image is. A digital image, or "bitmap", consists of a grid of dots, or "pixels", with each pixel defined by a numeric value that gives its intensity. In fact, we can describe an image by two main properties:

- **Luminance:** received brightness of the light, which is proportional to the total energy in the visible band.
- **Chrominance:** describe the perceived color tone of a light, which depends on the wavelength composition of light chrominance is in turn characterized by two attributes – hue and saturation.
    - hue: Specify the color tone, which depends on the peak wavelength of the light
    - saturation: Describe how pure the color is, which depends on the spread or bandwidth of the light spectrum

2.2. THE FLOW OF THE IMAGE COMPRESSION CODING

Image compression coding is to store the image into bit-stream as compact as possible and to display the decoded image in the monitor as exact as possible. Now consider an encoder and a decoder, when the encoder receives the original image file, the image file will be converted into a series of binary data, which is called the bit-stream. The decoder then receives the encoded bit-stream and decodes it to form the decoded image. If the total data quantity of the bit-stream is less than the total data quantity of the original image, then this is called image compression. The full compression flow is as shown in Figure 5.

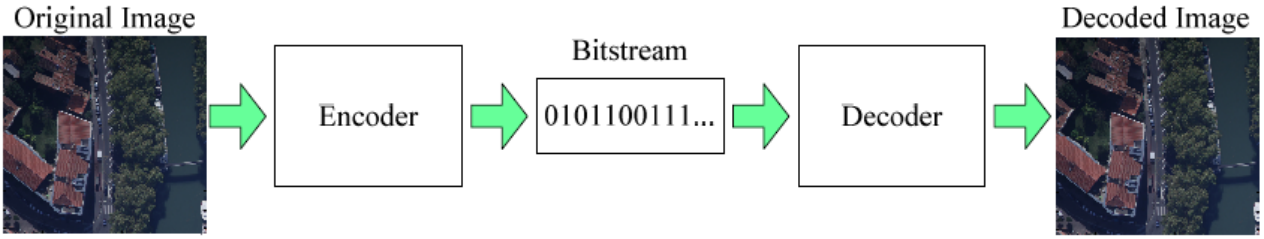


Figure 5- Compression Flow

The compression rate is defined as

$$Cr = \frac{n1}{n2} \tag{5}$$

where *n1* is the data rate of original image and *n2* is that of the encoded bit-stream.

2.2.1. REDUCE THE CORRELATION BETWEEN PIXELS

The reduction of the existing correlation is particularly useful in image compression because the correlation between one pixel and its neighbor pixels is very high, or we can say that the values of one pixel and its adjacent pixels are very similar. Once the correlation between the pixels is reduced, we can take advantage of the Information Theory (statistical characteristics and variable length entropy coding) to reduce the storage quantity.
This decorrelation stage is the most important part of an image compression algorithm; there are a lot of relevant methods available. The more common methods are as follows:

- **Predictive Coding**: Predictive coding is a compression method used for text and image compression. It encodes the difference between the current data estimation derived from past data and actual current data to attain more efficient compression. The degree of efficiency depends very much on the accuracy of the estimation as the difference becomes smaller, the information to be encoded becomes smaller as well. There are several lossless image compression algorithms which have been developed using this predictive coding method
- **Orthogonal Transform**: Orthogonal transformation consists on re-distributing the image data according to a new vector space in order to obtain an energy compaction. These methods are really useful when we are working lossy method as we can delete the coefficients which represent less energy. **Transform**: Karhunen-Loeve Transform (KLT) and Discrete Cosine Transform (DCT) are the two most well-known orthogonal transforms.

2.2.2. QUANTIZATION

The objective of quantization is and to achieve higher compression ratio by reducing the precision of the values to be encoded. For instance, while the original image uses 8 bits to store one element for every pixel; we can use less bits to save a coarser information of the image, and then reduce the storage size. The shortcoming of quantization is that it is a lossy operation, which will result into loss of precision and unrecoverable distortion. We have to highlight that **quantization is the only operation which generate lost,** if this block is omitted, independently of the decorrelation or coding method employed, the compression will be lossless.

2.2.3.ENTROPY CODER

Main objective of entropy coding is to achieve smaller average coding length for every sample of the image. Entropy coding assigns codewords to the corresponding symbols according to the probability of those symbols. In general, the entropy encoders are used to compress the data by replacing symbols represented by equal-length codes with codewords whose length is inversely proportional to the corresponding probability.

2.3. CCSDS COMPRESSION STANDARDS

CCSDS Recommended Standards define specific interfaces, technical capabilities or protocols, or provide prescriptive and/or normative definitions of interfaces, protocols, or other controlling standards such as encoding approaches. Standards must be complete, unambiguous and at a sufficient level of technical detail that they can be directly implemented and used for space mission interoperability and cross support.

2.3.1. CCSDS 123.0-B-1

The Standard CCSDS 123.0-B-1 represents a data compression algorithm applied to digital three-dimensional image data from payload instruments, such as multispectral and hyperspectral imagers, moreover it specifies the compressed data format obtained from the algorithm.

The Standard define the two functional part of a compressor as it is shown in Figure 6

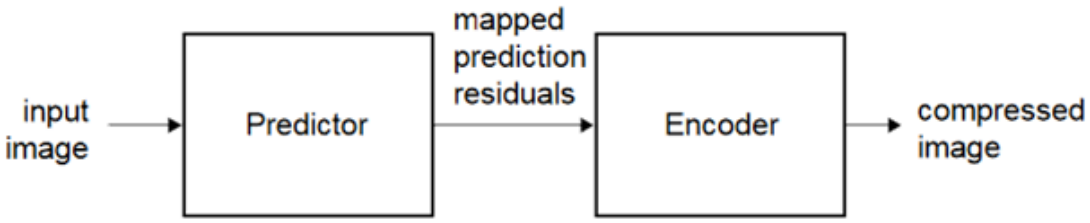


Figure 6- CCSDS-123.0-B-1 Compressor

2.3.2.1. Predictor

De-correlation algorithm is based in a predictive coding scheme, in fact the predicted value sample and the mapped prediction residual, depends on the values of nearby samples for the current spectral band and for P preceding spectral bands, where P is a user-specified parameter. Figure 7 illustrates the typical neighborhood of samples used for prediction; this neighborhood is suitably truncated in the margins of each image.

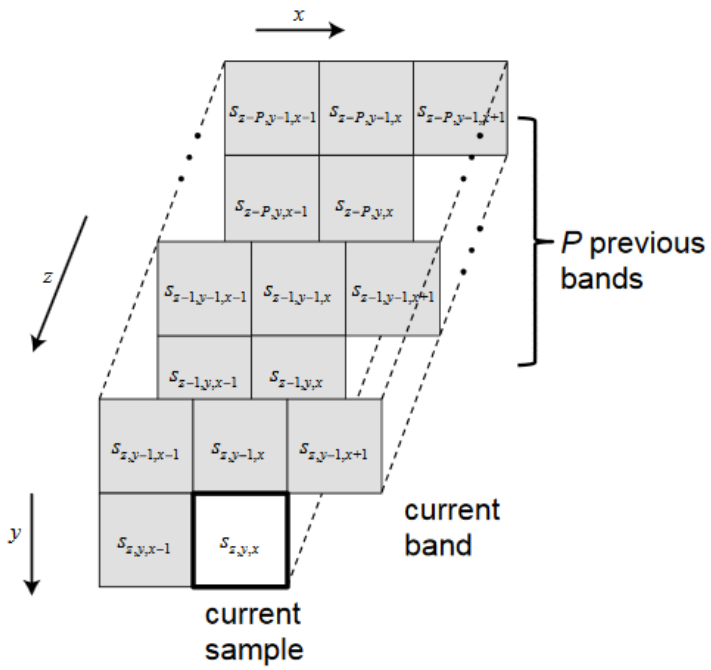


Figure 7- Typical Prediction Neighborhood

2.3.2.2. Encoder

CCSDS 123.0-B-1 encoder consist on in an adaptive entropy encoder, especially it uses Rice's adaptive coding technique. The compressed image will consist of a variable-length header followed by a body. To encode the mapped prediction residual for an image, the user may choose to use a sample-adaptive entropy coding or a block -adaptive approach. Underline that CCSDS 123.0-B-1 encoder is based on the adaptive entropy coder from CCSDS 121.0-B-0 Standard. Figure 8 gives us an approach of the adaptive entropy coder.

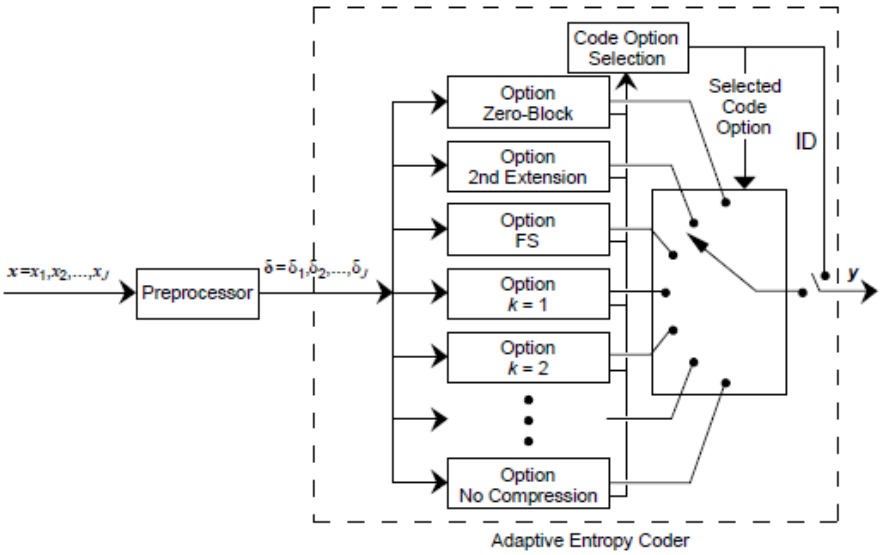


Figure 8- Adaptive Entropy Coder

2.3.2. CCSDS 122.0-B-1

CCSDS 122.0-B-1 Standard for a data compression algorithm applied to two-dimensional digital spatial image data from payload instruments and to specify how this compressed data shall be formatted into segments to enable decompression at the receiving end.

The compression technique described in the Standard can be used to produce both lossy and lossless compression. Figure 9 shows schema of the Standard.

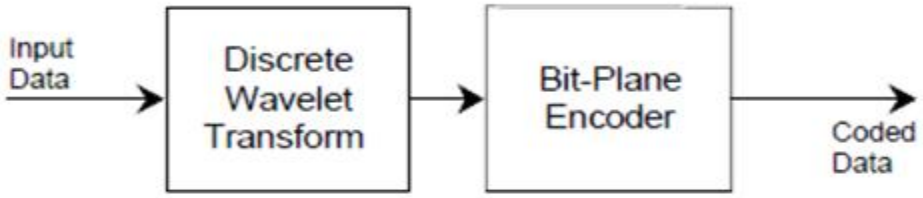


Figure 9- CCSDS 122.0-B-1 Compressor

2.3.2.1. Discrete Wavelet Transform

Standard de-correlation module makes use of a three-level, two dimensional, separable Discrete Wavelet Transform (DWT) with nine and seven taps for low- and high-pass filters, respectively. The two dimensional transform is allowed by repeating a one-dimensional Discrete Wavelet Transform over each dimension. Moreover, CCSDS-122 Standard provides two DWT to be selected. The 9/7 biorthogonal DWT, referred to as '9/7 Float DWT' or simply 'Float DWT', and a non-linear, integer approximation to this transform, referred to as '9/7 Integer DWT' or simply 'Integer DWT'.

Figure 10 illustrates the three-level, two dimensional, separable Discrete Wavelet Transform (DWT).

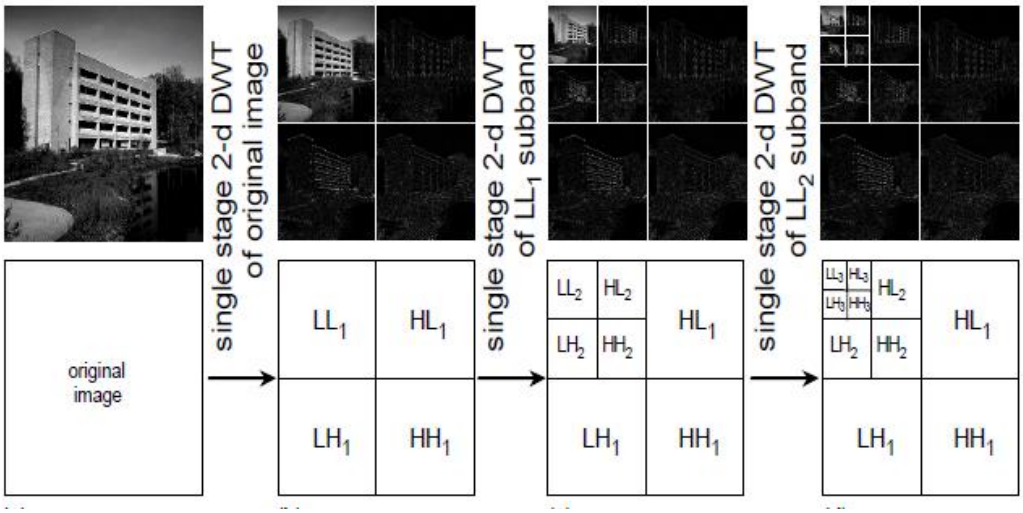


Figure 10- DWT Example

2.4. EXTENSION TO VIDEO COMPRESSION

Video compression can be viewed as image compression with a temporal component since video consists of an image sequence. From this point of view, the only "new" technique introduced is a strategy to take advantage of this temporal correlation. In fact, Video compression algorithms make use of the temporal correlation to remove redundancy.

The previous reconstructed frame is used to generate a prediction for the current frame. The difference between the prediction and the current frame, the prediction error or residual, is encoded and transmitted to the receiver. The previous reconstructed frame is also available at the receiver. Therefore, if the receiver knows the manner in which the prediction was performed, it can use this information to generate the prediction values and add them to the prediction error to generate the reconstruction.

2.4.1. MOTION COMPENSATION

The prediction operation in video coding has to take into account the motion of the objects between frames, which is known as motion compensation. If we try to apply temporal prediction without handle the fact that object tend to move between frames, we can actually increase the amount of information that needs to be transmitted.

The amount of information may increase for the next reason:

.

Consider two frames of a motion video sequence shown in Figure 11. The only difference between frames is an object displacement. Using a basic predictive technique, as the difference between two frames like is shown in Figure 12; the results image will contain more detail than the original image.

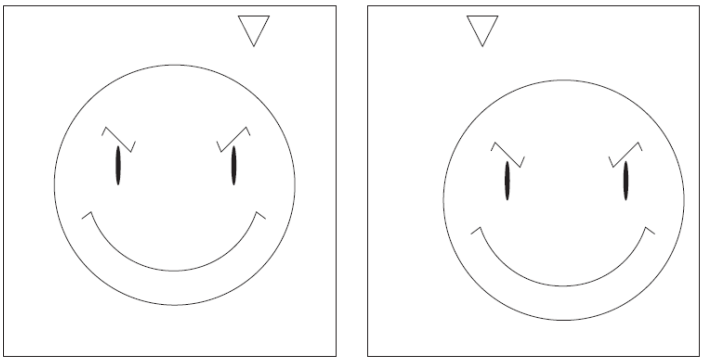


Figure 11- Motion Frames

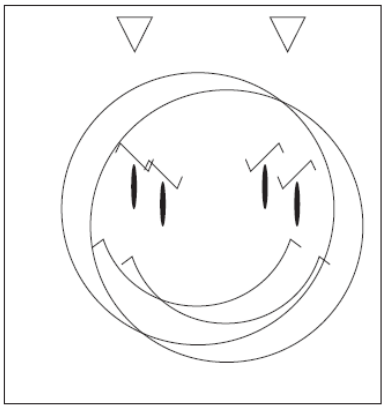


Figure 12- Overlap Frames

Using a proper motion compensation technique we can reinforce the displacement and as a result, improving the temporal prediction approach as it is shown at Figure 13.

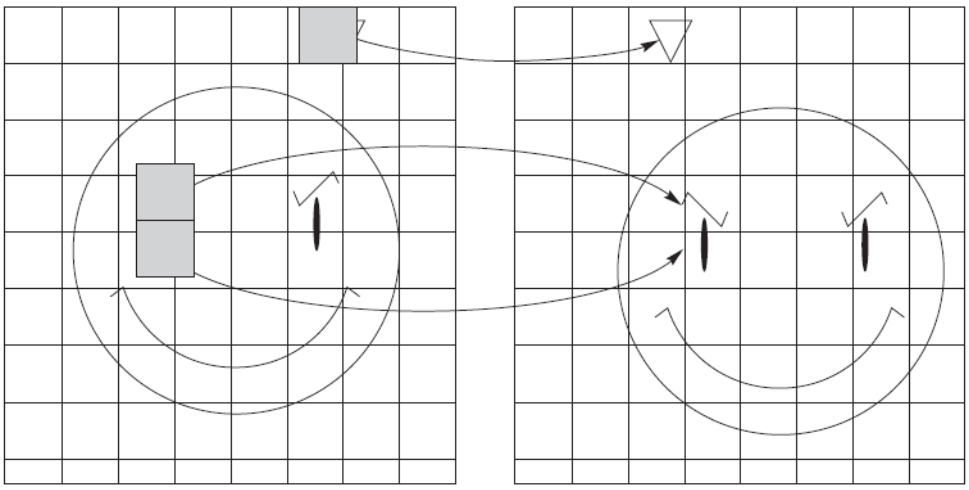


Figure 13- Motion Compensation Technique

3.INTRODUCTION TO THE BAYER PATTERN AND CFA DEMOSAICING

Summary

Today, the majority of color cameras are equipped with a single CCD (Charge- Coupled Device) sensor. The surface of such a sensor is covered by a color filter array (CFA), which consists in a mosaic of spectrally selective filters, so that each CCD element samples only have one of the three color components Red (*R*), Green (*G*) or Blue (*B*). The Bayer CFA is the most widely used one to provide the CFA image where each pixel is characterized by only one single color component. To estimate the color (*R*,*G*,*B*) of each pixel in a true color image, one has to determine the values of the two missing color components at each pixel in the CFA image. This process is commonly referred to as CFA demosaicing, and its result as the demosaiced image.

3.1. BAYER PATTERN

A CFA is an array of alternating color filters that samples only one color band at each pixel location. The most popular CFA pattern is the Bayer pattern Figure 14, which features blue and red filters at alternating pixel locations in the horizontal and vertical directions, and green filters organized in the quincunx pattern at the remaining locations. This pattern results in half of the image resolution being dedicated to accurate measurement of the green color band. The peak sensitivity of the human visual, shown in Figure 15, system lies in the medium wavelengths, justifying the extra green sampling. Because each pixel now has only one color sampled, a demosaicing algorithm must be employed to recover the missing information.

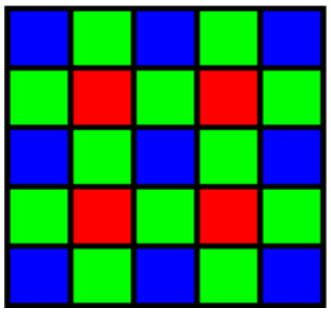


Figure 14- Bayer CFA Pattern

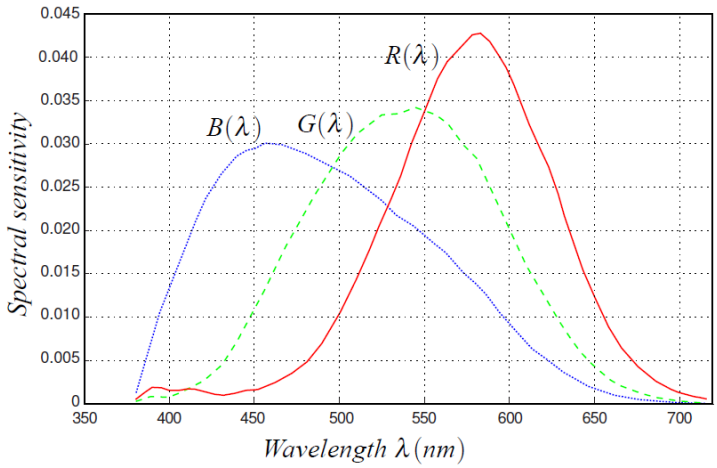


Figure 15- Human visual sensitivity

3.2. CFA DEMOSAICING

CFA demosaicing is the process through which three fully populated color planes are created from the CFA data. Several algorithms exist for this purpose, ranging from simple linear interpolators to high-end nonlinear interpolators that exploit as much spatial and spectral information as possible.

In most imaging systems, image compression is carried out after demosaicing process and compression algorithms usually have focused on compressing three channel color images. Recently, several image compression methods are proposed to directly handle CFA images that are captured before color interpolation. These CFA compression schemes exploit the fact that CFA images do not contain the redundant information introduced by demosaicing

3.2.1. CONFIGURATIONS

When images are transmitted or stored from the image capture module, the connection between two systems is almost always bandwidth-limited and becomes the bottleneck of whole system performance. Thus, efficient image compression methods are employed to send fewer amounts of data to the link and they are commonly applied after demosaicing operation as it is shown in Figure 16.

Recently, an alternative imaging system is proposed as shown in Figure 17. The main difference of this system is that it directly compresses CFA image and demosaicing operation is done after decompression.
By compressing the CFA data, the compressor can deal with only one-third of data compared to the case for compressing three-channel RGB images. Thus, it does not have to fight with the redundancy information that is introduced by color interpolation, which enables more efficient compression. In addition, CFA compression scheme offloads computational burden to the decoder side that usually has much more computation power and memory. Not only reducing the load on the compressor, it also passes whole demosaicing stage to the other side [3] [4] [5].

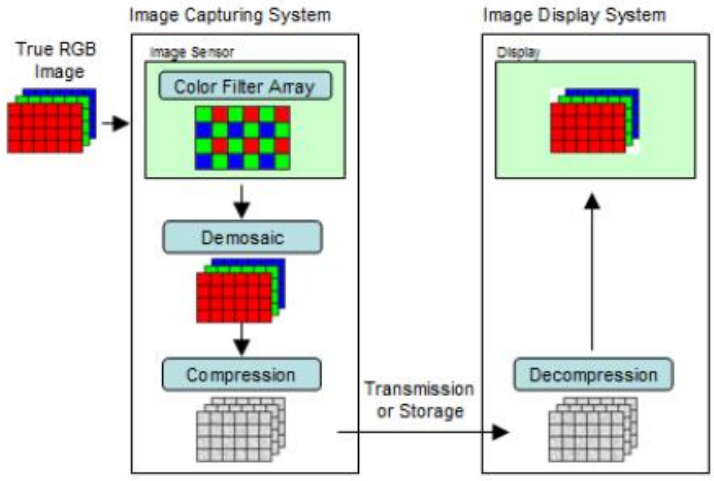


Figure 16- Demosaicing before Compression

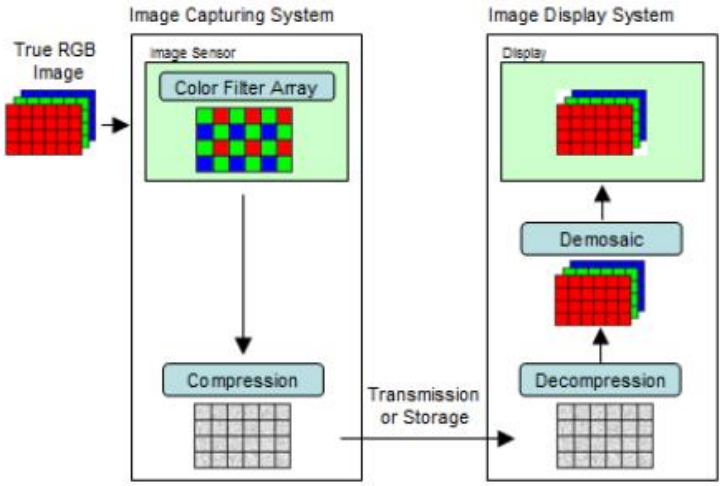


Figure 17- Demosaicing after Decompression

Part 3: Generation of the material (Matlab)

1. GENERATION OF BINARY IMAGES

Summary

Project requirements conduct us to generate and to adapt images for checking the developed algorithms. Moreover, as we do not dispose of an image repository from other missions, we have searched a set of images, which may have similar characteristics as the mission images.

1.1. COLOR IMAGES

Images generation has been one of the most important steps to test and verify the algorithms implemented. To carry out the task, we have used Matlab platform, where images can be seen as numerical matrix and where there are libraries to create binaries files. The steps followed to create binaries files are shown in the Figure 18.

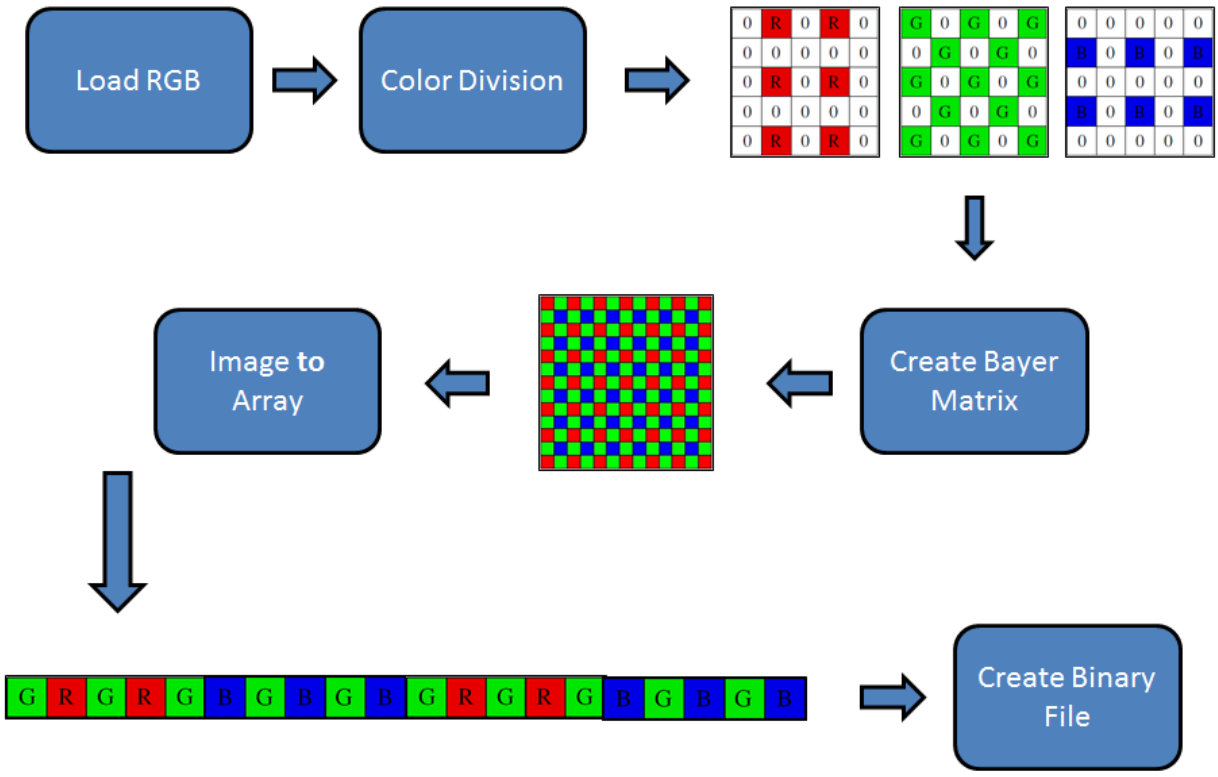


Figure 18- Binary Files Generation Steps

1.1.1. STEPS

1.1.2.1. Load RGB Image

Matlab is able to open most image commercial formats, independently of the compression method. Concerning our case, we have RGB images. These images were composed by 8-bits per pixels and per band.

1.1.2.2. Color Division

From each band, we just need the values corresponding to the Bayer pattern, in order to create the Bayer pattern Matrix; we will set the undefined pixels to zero.

1.1.2.3. Create Bayer Matrix

We will overlap the three bands in one band by adding pixels. This "new" band will be the Bayer Pattern Matrix.

1.1.2.4. Images to Array

To create the data vector, we should re-organize our Bayer Matrix (NxN) into a unidimensional vector ($1xN^2$), in order to make easy the mapping within a binary file.

1.1.2.CREATE BINARY FILE

Matlab provide us a library in order to generate binary files. Specific *"fwrite"* function led us to generate binaries files specifying the number of bytes per data, data format and byte ordering.

1.1.2.1.Examples

Figure 19 shows an example, which has been created in Matlab.

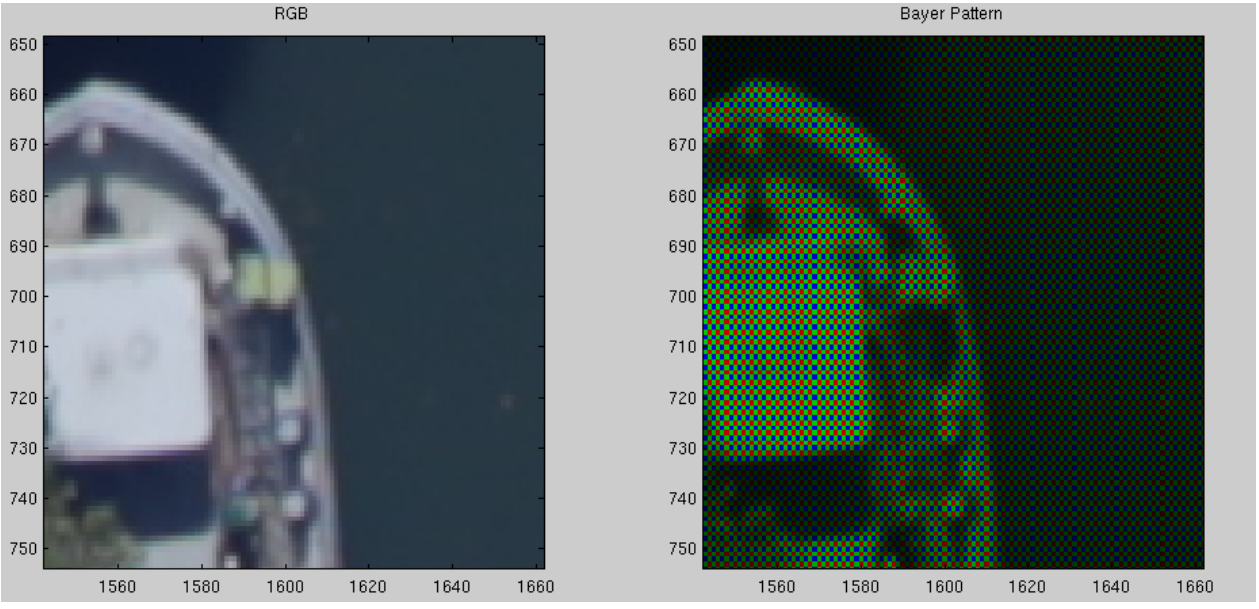


Figure 19- Bayer CFA Example

1.2. INFRARED IMAGES

In our mission, we also have infrared Images to compress. To generate these images we have to take into account the information provided by Figure 20.

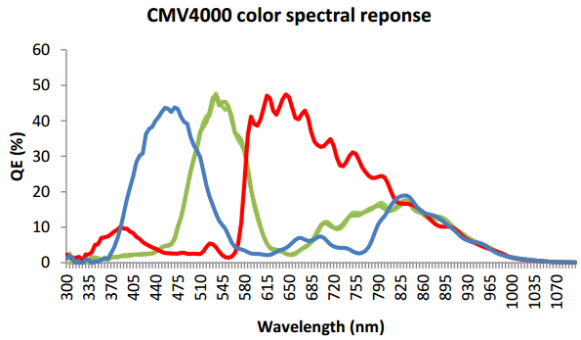


Figure 20- Color Spectral Response

In the infrared frequencies, Red, Green and Blue color responses are quite similar, therefore the total final response captured by the sensor might be pretty close to luminance.
Matlab provide us a way to create the luminance of a pixel, directly from the RGB data by the function *"rgb2gray"*. The schema to generate the image is shown in the Figure 21.

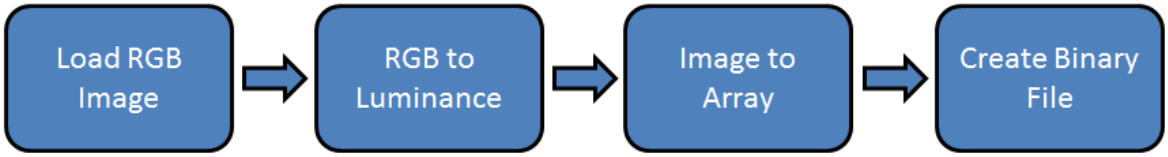


Figure 21- Image Generation Schema

2.GENERATION OF SEQUENCES (MOTION)

Summary

As data acquisition consists on taking image sequences, we have considered video compression techniques as possible algorithms. Frames differences will consist on small movements generated by satellite instabilities. Therefore, we should carry out a series of motions inside the image to simulate the satellite motion.

2.1. INTRODUCTION

Create image sequences and simulate the motion between frames could be a problem taking into account that we do not have any test images or image sequences to estimate or predict the impact of the movement. To simulate the displacements, we are going to apply image processing (bilinear interpolation) upon a bigger image and we will split it into small images.

In this project, only translation movements have been subjects of study as they are the most common motion phenomena, therefore we will develop basic techniques to estimate camera translation movements.

2.2. TRANSLATION MOVEMENT TECHNIQUES

2.2.1. HORIZONTAL AND VERTICAL MOTION (FIXED PIXEL)

Figure 22 and Figure 23 show us four possible movements. Horizontal movements can be combined with vertical movements, creating different permutations to simulate the motion.

2.2.2.1. Horizontal motion

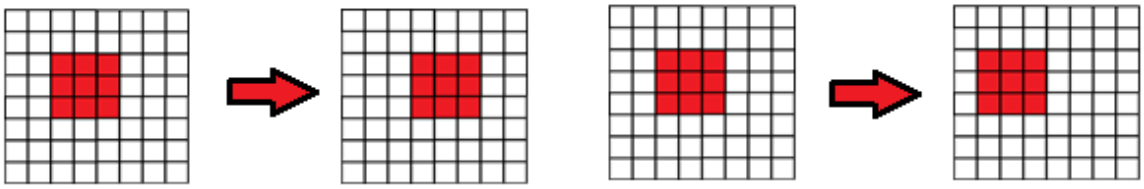


Figure 22- Horizontal Movement

2.2.2.2. Vertical motion

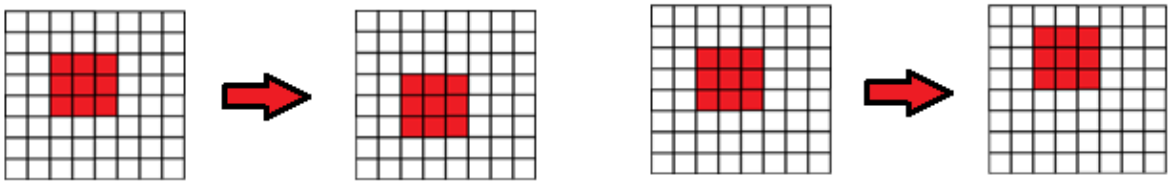


Figure 23- Vertical Movement

2.2.2. HORIZONTAL AND VERTICAL MOTION (INTERPOLATION)

Typically, differences between two consecutive frames are not an integer number of pixels and simulating this phenomenon involves the use of interpolation techniques. Equation 6 shows the mathematical expression of the bilinear interpolation used for our simulations.

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2] \quad (6)$$

As Equation 6 shows us, we can consider a bilinear interpolation as a filter technique, indeed creating motion requires filtering our image by a NxN matrix. Depending on matrix values and matrix sizes, motion distances and directions will result different. Figure 24 illustrates the filter technique.

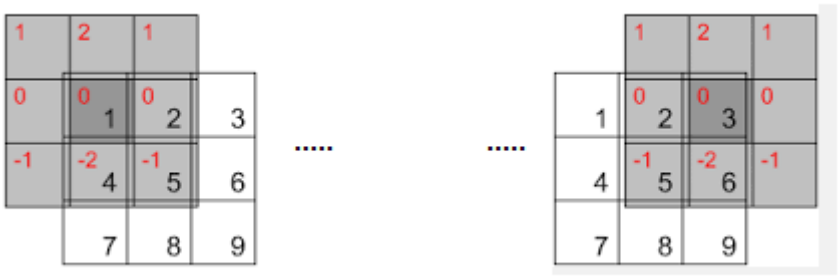


Figure 24- Filter Technique

2.3. BLOCK DIAGRAM

Finally, keeping in mind that the relationship between images sequences and independent frames, creating a motion illusion can be achieved just by a simple "motion generation" block as is illustrated in Figure 25.

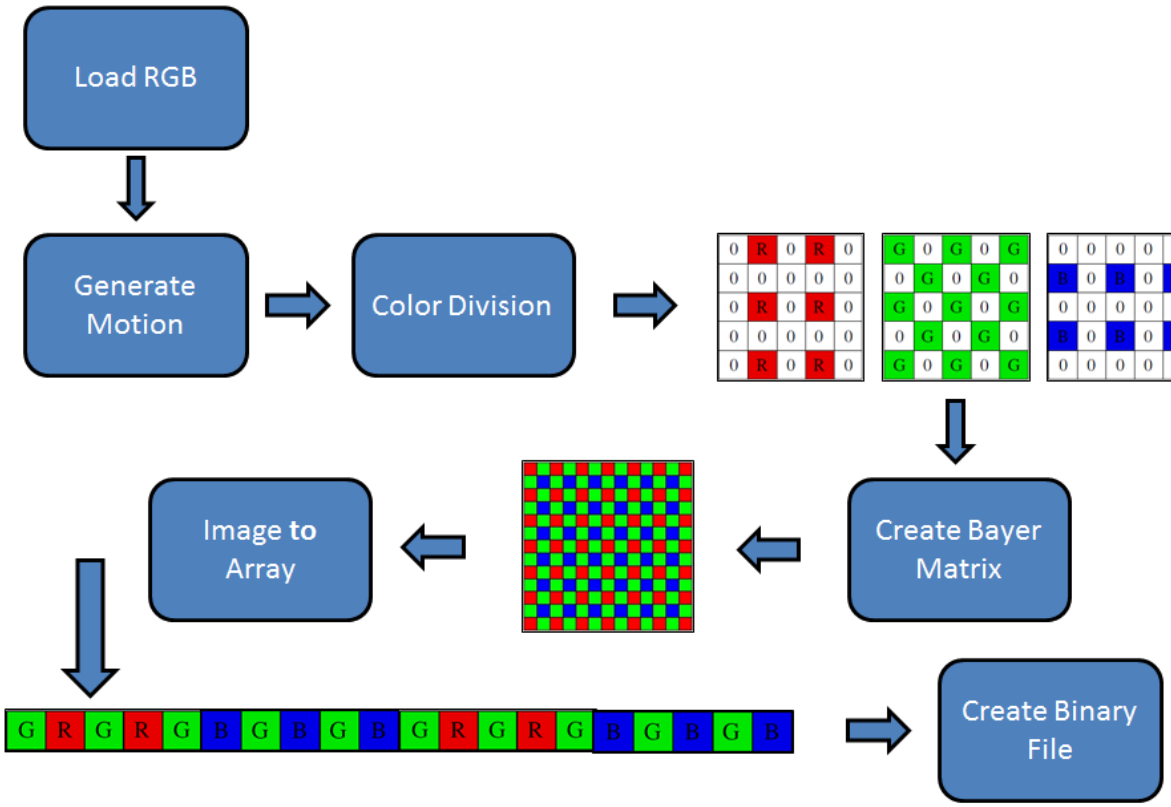


Figure 25- New Binary Files Generation Steps

3. ADAPT TO REAL PIXELS

3.1. INTRODUCTION

Mission requirement does not allow dynamic range reduction or quantization, as lossless compression is a strong requirement. This is important since real RAW format, which will be generated by the camera, will be equivalent to 12 bits per pixel and until now we have just made images of 8 bits. Moreover, we must consider that camera devices are not ideal and they generate both, Gaussian (additive) and Laplacian (multiplicative) noise.
Image repositories of NASA, Hubble telescope, or ESA are in JPEG format, therefore only pre-processed images, by a lossy compression algorithm, are available. Noise component will be reduced because of the compression.

3.2. SOLUTIONS

As we need pixels of 12 bits, dynamic range is extended by a multiplication factor. Multiplication factor will upshift the data by 4 bits (12 bits – 8bits), in other words, by factor of 16 (2^4). Image pixels with small values will be considered noise; therefore the multiplicative factor will be applied according to a threshold. This threshold decision is important as multiplicative noise will affect intensely upon higher pixel values. As we don't know the multiplicative noise, we keep that Image noise, which will be incremented thanks to the (2^4) factor. To introduce Gaussian noise, we can exploit the information about the noise behavior in the Eye-Sat sensor. Thus, a normal random variable with a 2 bits standard variation is introduced. Those 2 bits of Standard variation result from adding the temporal noise of the pixel, which is under 15 electron rms (1 LSB rms), and the spatial noise (pixel to pixel variation), which is under 13 electron rms.

Part 4: Algorithms implemented

1.INTRODUCTION

Summary
The most important step of this project consists on developing compression algorithms; in fact, these algorithms must provide some properties as low-complexity and good compression ratio. Of course both abilities are usually incompatible, so we should make a trade-off to obtain as good performance as possible.

1.1. GOAL

Nowadays must of research in compression algorithms has focused on what is called lossy compression algorithms, where the output data after decompression it is not exactly the same as the input data of the compression system.
However, when losses are not allowed, lossless compression algorithms must be chosen. Lossless compression algorithms provide us the possibility of restoring the data without any loss; in return we won't be able to obtain a high compression ratio.

As the compression algorithm will be deployed in the satellite central processor which is shared with other process, it must be designed under the premise of low complexity. Moreover, it should obtain a minimum value of compression rate. Both values are commonly conflicting because the reduction of the data redundancy involves making a costly preprocessing.

The rapport topic down-below is the detailed explanation of each implemented algorithm, with particular attention upon the technical implementation, and theoretical causes, decisions and problems to develop them. The comparative performances of the algorithms will be illustrated on Part 5.

The following is an index of the compression algorithms which have been developed.

- **Algorithm Bayer matrix with CCSDS 123**
  - **Pattern one**
  - **Pattern two**
- **Interpolated Algorithm with CCSDS 123**
- **Algorithm CCSDS 122**
- **Video Coder algorithm 1**
- **Video Coder algorithm 2 low complexity**
- **Segmented-Final Algorithm**
- **Efficient-Segmented –Final Algorithm**
- **Demosaicing method**

2. ALGORITHM BAYER MATRIX WITH CCSDS 123

Summary

CCSDS 123 is a compression standard, whose prediction block not only exploits the redundancy between adjacent pixels, but also use spectral bands redundancy to predict the image's pixels. As we have a Bayer pattern image, we can consider instead of having just one band, split out the image in two or three separated color bands and estimate the pixels of the other bands with more accuracy.

2.1.INTRODUCTION

As it is known, CCSDS 123 defines a payload lossless data compression that has applicability to multispectral and hyperspectral imagers and sounder. This compression algorithm can be very interesting if we consider that a Bayer pattern image has the information of several spectral bands, especially red, green and blue bands. So, the goal of developing this algorithm is to exploit the functionalities of the Bayer pattern to improve the prediction of the pixels.

2.1.1. CCSDS-123 PREDICTOR

This section is an extract of CCSDS-123 standard [15] and specifies how to calculate predicted samples values $\hat{S}_{x,y,z}$ and mapped prediction residuals $\delta_{x,y,z}$ from the input image samples $S_{x,y,z}$. $\hat{S}_{x,y,z}$ and $\delta_{x,y,z}$ generally depends on the values of nearby sample in the current band a P preceding spectral bands, where P is an user-specified value. Figure 7 shows the most common neighborhood of samples used for prediction, moreover we should underline that neighborhood will be suitable truncated when $y = 0, x = 0, x = N_X - 1, or\ z < P$.

Within each spectral band, the predictor computes the local sum of the nearby samples values as it is shown at [15]. Local sum will use to compute a local difference. Predicted sample values are calculated using the local sum in the current spectral band and a weighted sum of local difference values from the current and previous spectral bands. The weights used in this calculation are adaptively updated following the calculation of each predicted sample value. Each prediction residual, that is, the difference between a given sample value $S_{x,y,z}$ and the corresponding predicted sample value $\hat{S}_{x,y,z}$, is mapped to an unsigned integer $\delta_{x,y,z}$, the mapped prediction residual.

Local sum $\sigma_{x,y,z}$ is a weight sum of nearby samples in the current spectral band Z. Within the Standard a user may choose to perform prediction using neighbor-oriented or columns-oriented local sums for an image. A neighbor-oriented local sum is figured out by carrying out weighted addition of the four neighboring samples values in the spectral band as is illustrated in Figure 26. Column-oriented local sums is calculate by summing four times the neighbor sample value In the previous row, except when $Y = 0$, as this sample it is not available. Figure 26 illustrates also this method.

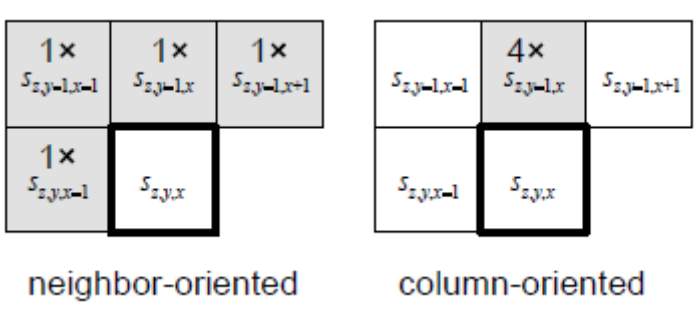


Figure 26- Samples Used to Calculate Local Sums

Local differences will be compute thanks to the local sums, therefore to compute the central local difference $d_{x,y,z}$, we make the difference between the local sum $\sigma_{x,y,z}$ and four times the sample value $S_{x,y,z}$ [15]. Directional local differences $d^N{}_{x,y,z}$, $d^W{}_{x,y,z}$, $d^{NW}{}_{x,y,z}$ are the difference between the local sum $\sigma_{x,y,z}$ and four times a sample value corresponding to the adjacent pixels N, W, NW. Figure 27 illustrates those pixels.

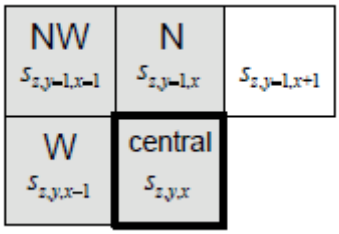


Figure 27- Computing Local Differences in a Spectral Band

Standard provides two more possibilities concerning how compute the prediction samples, Full and Reduce mode. Reduce mode, prediction depends on a weighted sum of the central local differences computed in preceding bands and the direction local differences are not used. Full mode, prediction depends on a weighted sum between the central local difference computed in preceding bands and the directional local differences computed in the current band.

The prediction residual, which is the difference between the sample value and the predicted value, is mapped to an unsigned integer $\delta_{x,y,z}$ as it is show in [15]. The mapping is invertible, therefore $S_{x,y,z}$ can be exactly reconstruct.

Moreover, to obtain the prediction value, each component of the local difference vector $U_z(t)$ is multiplied by a corresponding integer weight value $W_z(t)$.

The predicted central local difference is equal to the inner product of vectors $U_z(t)$ and $W_z(t)$.

$$\hat{d}_z(t) = W^t{}_z(t) U_z(t) \tag{7}$$

With this value we will compute the scaled predicted sample $\tilde{S}_z(t)$ and the predicted sample value $\hat{S}_z(t)$

$$\tilde{s}_z(t) = \begin{cases} \mathrm{clip}\left(\left\lfloor \frac{\mathrm{mod}^*_R\left[\hat{d}_z(t) + 2^{\Omega}\left(\sigma_z(t) - 4s_{\mathrm{mid}}\right)\right]}{2^{\Omega+1}} \right\rfloor + 2s_{\mathrm{mid}} + 1, \{2s_{\min}, 2s_{\max}+1\} \right), & t > 0 \\ 2s_{z-1}(t), & t = 0, P > 0, z > 0 \\ 2s_{\mathrm{mid}}, & t = 0 \text{ and } (P = 0 \text{ or } z = 0). \end{cases} \tag{8}$$

$$\hat{S}_z(t) = \left\lfloor \frac{\tilde{S}_z(t)}{2} \right\rfloor \tag{9}$$

Once we find the predicted sample value, the prediction residual $\Delta_z(t)$ is the difference between the predicted sample value and the current sample. The mapped prediction residual $\delta_{x,y,z}$ is defined at [15].

2.1.2. CCSDS-123 ENCODER

The section describes the encoding step concerning the compressor algorithm, moreover the format of a compressed image. The compressed image consists on a variable-length header followed by a body. Header stores the compression parameters and the body stores the mapped prediction residuals $\delta_{x,y,z}$.

As described in [15], Header shall store the following parameters in the following order:

- Image Metadata – 12 Bytes
- Predictor Metadata – Variable length
- Entropy coder Metadata- Variable length

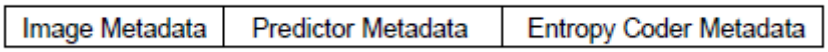


Figure 28- Overview of Header Structure

See availability notes for further information at [15].

To encode the mapped prediction residuals for images, the standard provides two methods. The first one is to use the sample adaptive entropy and the second one uses a block-adaptive entropy coding.

Under the sample adaptive entropy coding approach, each mapped prediction residual is encoded using a variable-length binary codeword. The variable-length codes used are adaptively selected based on the statistics of the preceding samples, statistics will update per iteration. Moreover, separate statistics are maintained for each spectral band, and the compressed image size does not depend on the order in which mapped prediction residuals are encoded.

Under the block-adaptive entropy coding approach, the sequence of mapped prediction residuals is partitioned into short blocks, and the encoding method used is independently and adaptively selected for each block. Depending on the encoding order, the mapped prediction residuals in a block may be from the same or different spectral bands, and thus the compressed image size depends on the encoding order when this method is used.

For additional information; check [15]

2.2. COMPRESSOR

Taking into account what we have said in the paragraph before, we are going to develop the algorithm which is illustrate in Figure 29 through a block diagram.

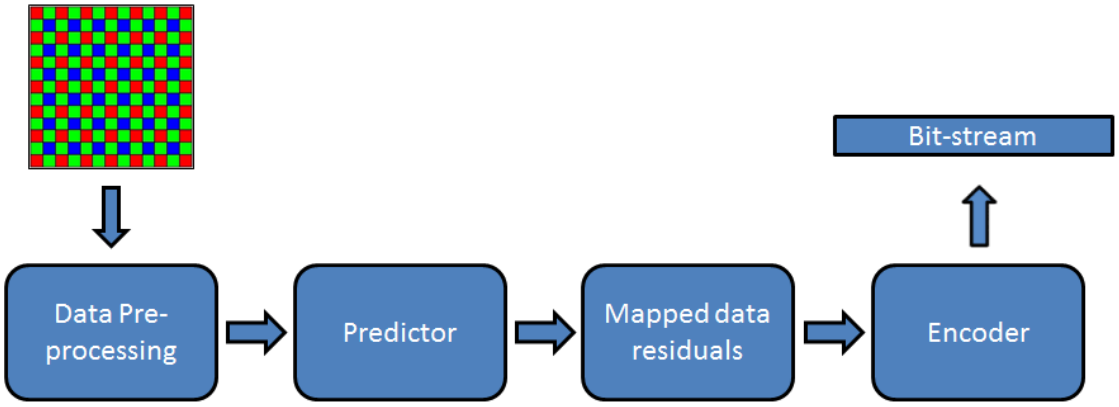


Figure 29 – Compression Diagram Block

2.2.1. DATA PRE-PROCESSING

This block consist on generating two different spectral bands from the Bayer Pattern images, So after this block instead of having a single image, we could consider two images, each one belongs to real data and each one will be a band to introduce within CCSDS 123 Predictor.

So, the idea it is to create two bands from the original data with the goal of letting our predictor be able to estimate pixels as better as possible. We are going to propose two band patterns with an explanation of why we should use it. In Part 5, we will compare the performance between both patterns and we will decide which one give us better performance.

Pattern one

First pattern born from the fact of the Bayer patter disposes twice green samples than reds and blues, moreover the green color have mutual information with red and blue color as it is shown in the Figure 20.

Under these two assumptions is seems logic to create a band of green and use it for predicting the pixels blue and red. To sum up, we can sort the original green data in one band and the non-green data in the other band as it is shown in Figure 30.

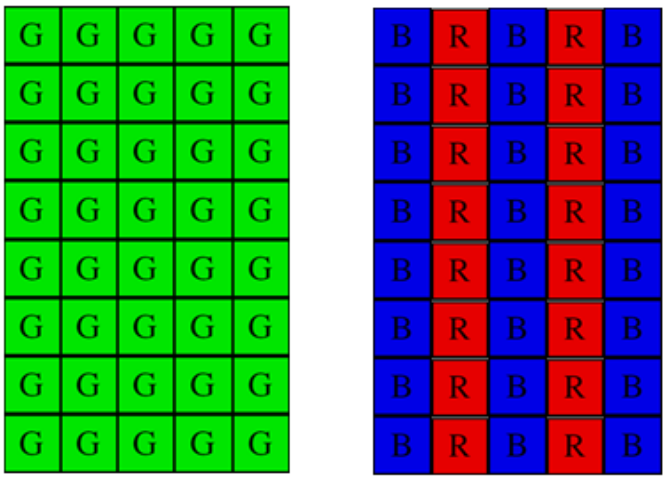


Figure 30- Pattern 1

Pattern two

It is not difficult to find the main problem of the first pattern. As we can see in Figure 31, the second band is a set of red and blue colors, whose frequencies band don't have any mutual information upon our sensor, as it is shown in Figure 20, so although we are going to have information from the green band to predict the non-green band, we are wasting the ability of carrying out a correct prediction over the spatial dimension, especially in horizontal vector component.

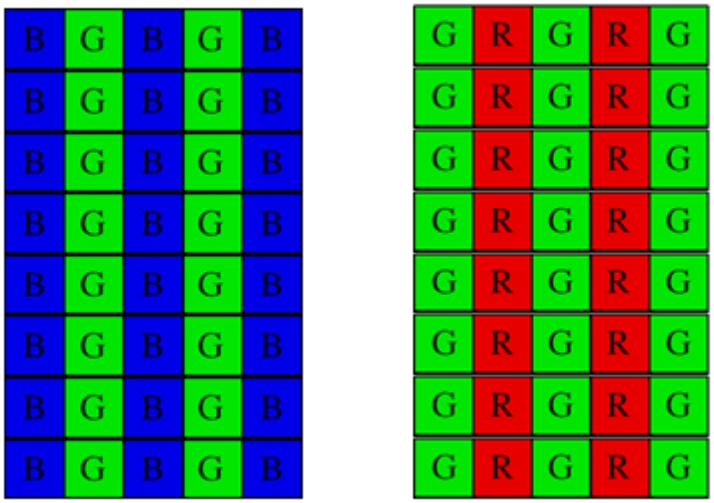


Figure 31- Pattern 2

Using the pattern shown in Figure 32, we can solve the problem before, as red and blue components has always mutual information with green color and bands have adjacent pixels with redundancy in three dimensions, spatial and spectral.

2.2.2. PREDICTOR

CCSDS-123 predictor has been selected to undertake pixels estimation. The goal is forecast pixels as better as possible to obtain the lowest residual and send it to the mapping. Main characteristic upon CCSDS 123 are defined in the section before.

2.2.3. MAPPING

After obtaining the residuals by the pixels prediction, we have to notice how this data is statistically distributed. Usually the probability density function takes a Laplace distribution centered in zero. Laplace function it is defines as:

$$f(x|\theta,\lambda) = \frac{1}{2\lambda} exp\left(-\frac{|x-\theta|}{\lambda}\right) \tag{10}$$

This distribution is defined by location theta and scale Lambda.

Figure 32 illustrates the PDF function located in zero, as our residuals will be.

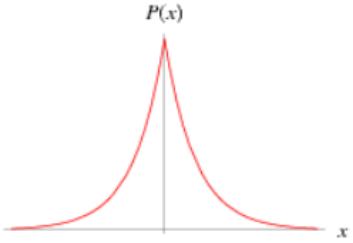


Figure 32 –PDF residuals samples

The prediction residual may be positive or negative, but the variable length codes used by the entropy coder are defined only on nonnegative integer inputs. Thus, each prediction residual is mapped to a nonnegative integer, called the *mapped prediction residual*. This mapping is invertible, so that the decompressor can reconstruct the original sample.

Equations of mapped function are illustrated in [15].

2.2.4. ENCODER

When we talk about the encoder, we are talking about the entropy encoder. In this case, the entropy encoder has two functions. The first one is to create a header with the necessary parameters to decode the bit-stream on the decompression step. The second one handles the way of compressing the data by replacing symbols represented by equal-length codes with the codewords whose length is somewhat inversely proportional to the corresponding probability of the symbols.

2.3. DECOMPRESSOR

Figure 33 illustrates the block diagram concerning decompression stage. First step consists on decoding the compressed bit-stream. Header parameters and mapped prediction residuals shall be stored at this point. Mapped prediction residuals are used by de-mapping stage to obtain the prediction residuals. Both, prediction residuals and Header parameters will be the inputs at Inverses predictor block. The results, the real samples, which will be restore at Data Post-processing stage as the CFA image.
As an optional processing, a bilinear interpolation will use as Demosaicing technique to create the RGB spectral bands.

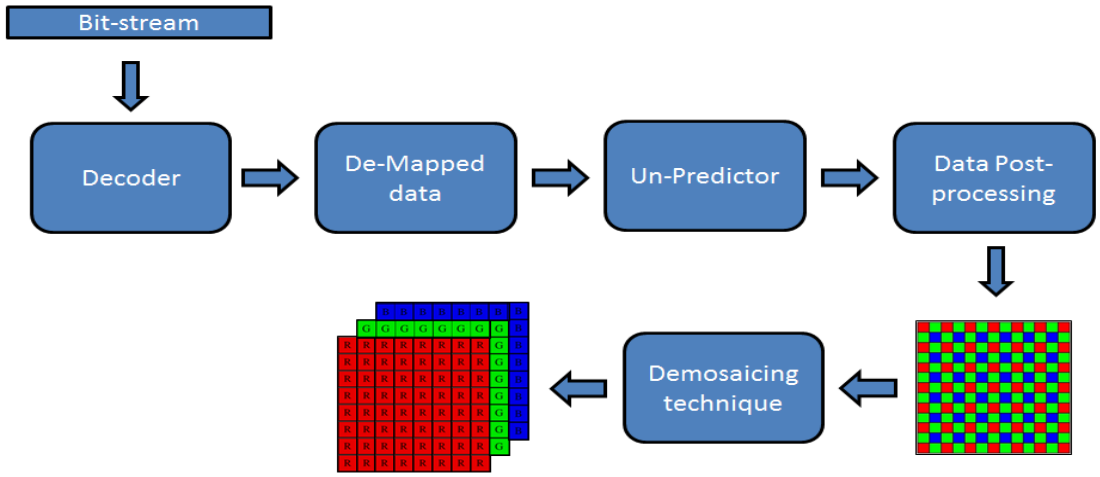


Figure 33 – Decompression block diagram

3. INTERPOLATED ALGORITHM WITH CCSDS 123

Summary

This method tries to go far away from the first method and carry out an invertible interpolation to endow the pixels more redundancy and achieve a better prediction.

3.1.INTRODUCTION

The goal of developing this algorithm is to exploit the characteristics of the Bayer pattern to improve the prediction of the pixels. Moreover, we can extend the functionalities just applying a simple interpolation upon the pixels as we are supplying information from adjacent pixels.

So the idea will be the following, by introducing an interpolation over green pixels, we are going to estimate green pixels corresponding to blue and red positions. After that we will introduce to the predictor two bands, the first one will be the green-estimated pixels and the second one will be the non-green pixels. With this technique we could use more pixels to predict the target pixel and obtain better performance.

New Pattern

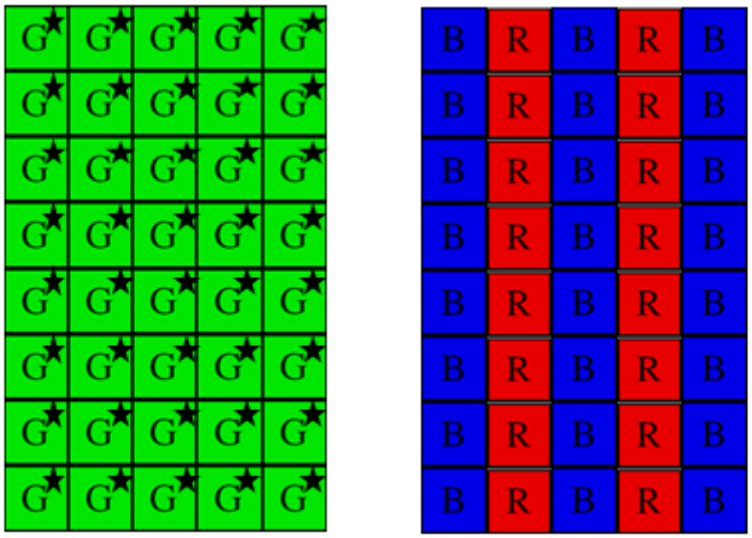


Figure 34- New Pattern

How we can calculate the green-estimated pixels will be explained in the next section.

3.2. COMPRESSOR - DECOMPRESSOR

Only one difference exists between the two algorithms and it concerns the Pre-processing step. Therefore there is no difference between the flow diagram illustrated in figure 30 and the diagram shown in Figure 35.

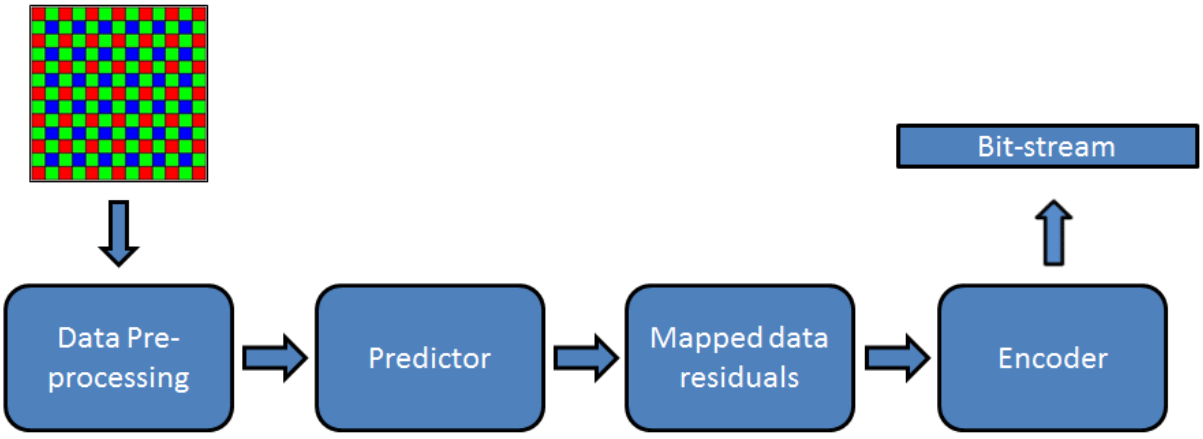


Figure 35- Compression Diagram Block

3.2.1. DATA PRE-PROCESSING

At this step, as only the pre-processing phase differs between algorithms, we will just discuss how to get the estimation data by a filtering technique in this pre-processing stage.

As we said, we should try to achieve an invertible interpolation to endow the pixels more redundancy and achieve a better prediction. The first idea is to take into account as much pixels as possible. In figure 36 we illustrate the following 8 possibilities in order to describe all the possible pixels to estimate in the image. Light green pixels represent the pixels which need to be estimated.

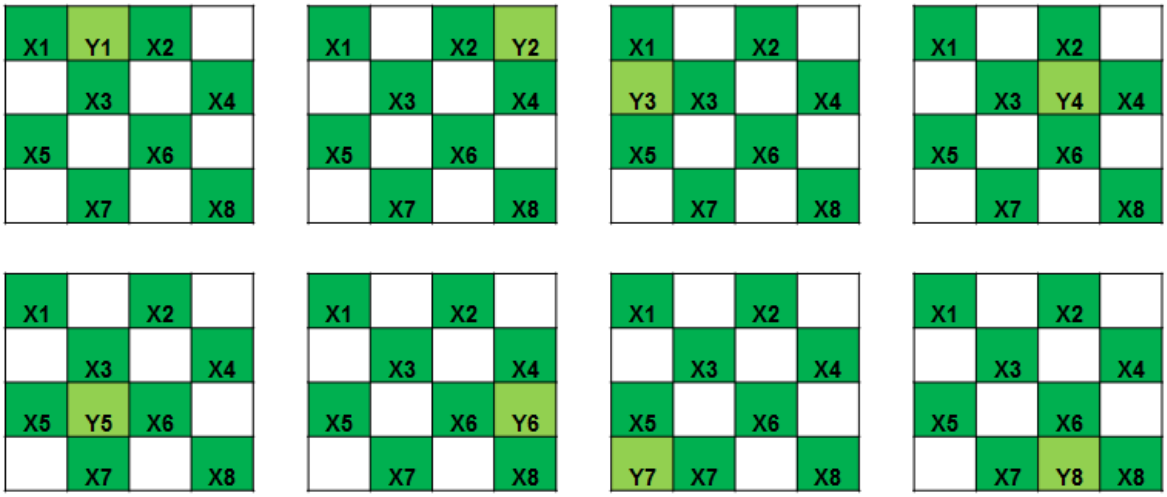


Figure 36- Green estimated Pixels

Green pixels represent real information from the camera. As the first idea was to use as much pixels as possible, we could estimate the pixels by a bi-linear interpolation as it is shown in Figure 37.

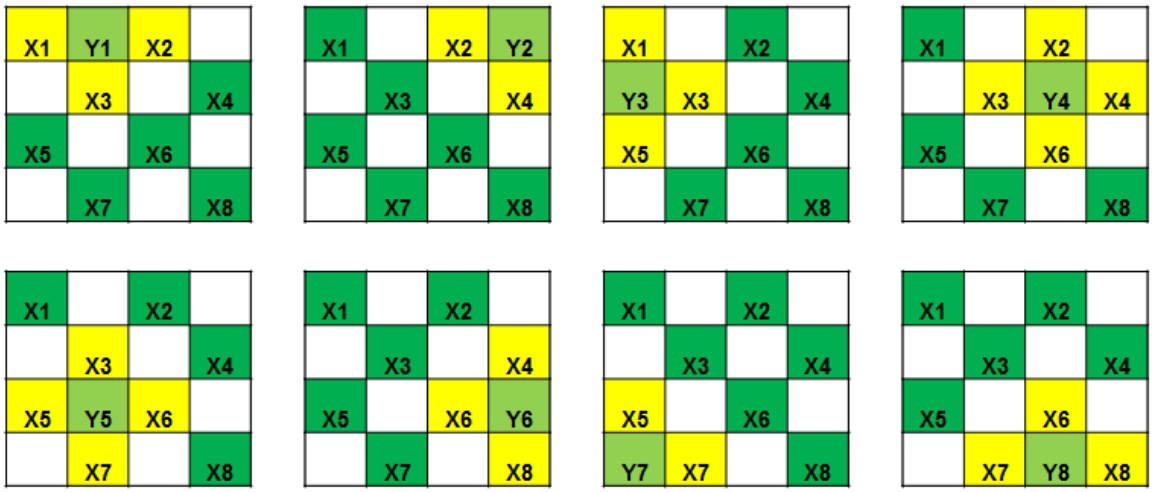


Figure 37- Interpolation Method

To carry out this method we can follow the procedure showed in Figure 38.

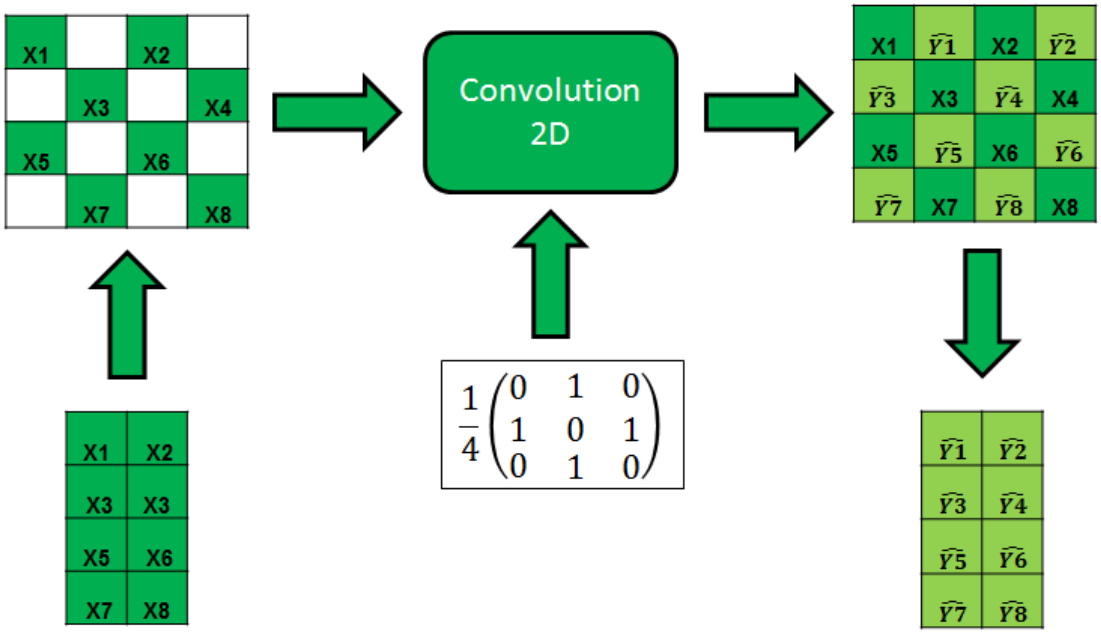


Figure 38- Interpolation Methodology

Because a lossless compression method is required, the data after decompressing should be exactly the same as in the original Bayer pattern picture data. So we have to check if the interpolation we have just developed can be inverted.

To do that, we can consider each pixel of the image as a value within a vector. So an image of NxN pixels can be seen as a vector of $1xN^2$. The linear interpolation we have already applied over the images can be seen as a linear transformation applied into the Vector. Equation 11 describes this procedure.

$$\hat{Y}_{1:N} = f(X_{1:N}) \tag{11}$$

And as every linear transformation, we can considerer a product of Matrix like it is shown in Equation 12.

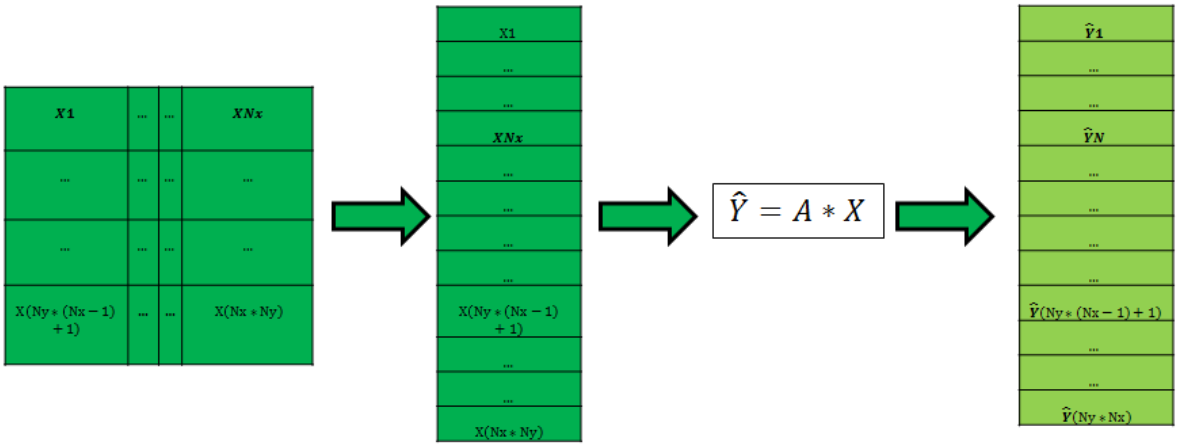
$$\hat{Y} = A * X \tag{12}$$

Algorithm is shown in Figure 40, where A represent the transformation matrix with $Nx = 2$ and $Ny = 4$.

Of course, the goal of using this methodology is not other than knowing if the system is invertible or not. To have an invertible system the matrix A has to be invertible. To know if a matrix is invertible we can come back to the basics rules of algebra and check the determinant.

$$det|A| = 0 \leftrightarrow \ \exists\, A\ such\ us\ \exists\, A^{-1} \tag{13}$$

A is a not invertible matrix due to the determinant is equal to zero. So we can not to use this interpolation to obtain a prediction of the pixels. Although in this part, we are not to illustrate the results of each algorithm (Part 5)), to introduce why we follow to looking for any invertible interpolation, we should advise that this interpolation of the samples provides an improvement of the compression ratio.



$$A = \begin{pmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 \\ 0 & 1/4 & 1/4 & 1/4 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/4 & 0 & 1/4 & 1/4 & 1/4 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \end{pmatrix}$$

Figure 39- Interpolation Algorithm Diagram Block

Next choice consists on a simpler linear system, where to obtain the prediction samples we use the methodology shown in Figure 40, where light green samples represent the predicted samples and yellow samples will use to estimate the prediction.

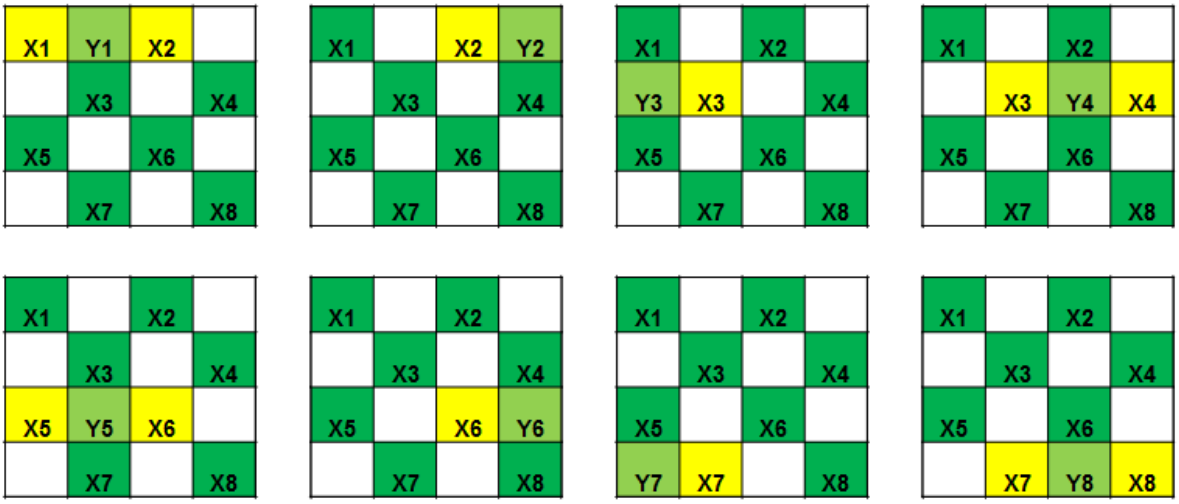
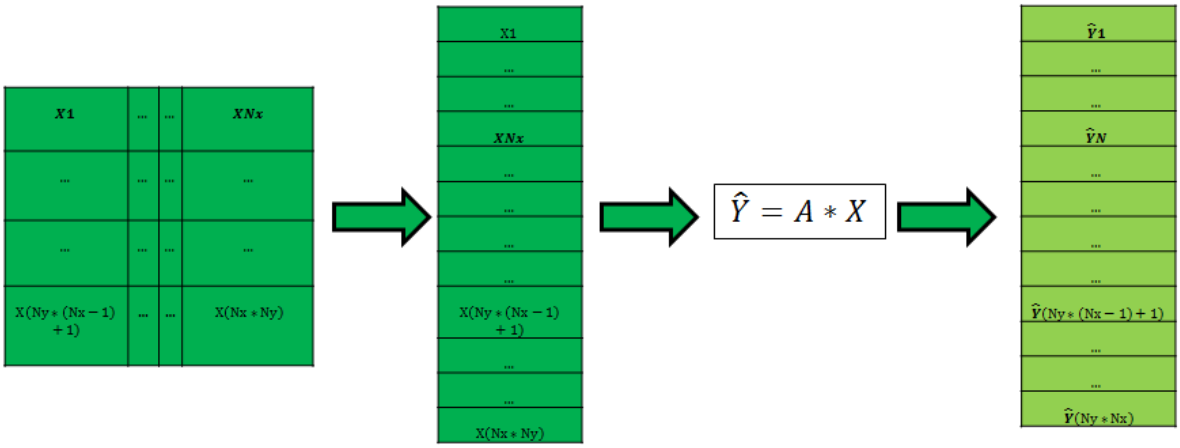


Figure 40- New Green estimated Pixels

In this case the new system has been represented by a matrix like the system show in the Figure 41. This matrix is invertible and as a result we can generate the inverse operation.



$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -1 \end{pmatrix}$$

Figure 41- New Interpolation Algorithm Diagram Block

4. ALGORITHM CCSDS 122

Summary

CCSDS 122 Standard [13] for image data compression is an algorithm applied to two-dimensional digital spatial image data from payload instruments. It also specifies how this compressed data shall be formatted into segments to enable decompression at the receiving end.

4.1. COMPRESSOR

Compressor algorithm performance has been tested by an ESA code. The results will be shown at the next section. As it is known this Standard is able to compress the data in both lossy and lossless methodology. To check the performance, we compare the compression rate achieved by this algorithm with the compression rate of the algorithms tested before. The results of previous predictive algorithms are better than those of this algorithm for all cases, so it has been decided to discard the CCSDS-122 Standard.

5. VIDEO CODER ALGORITHM 1

Summary

As we said, the objective of image compression is to reduce the redundancy of the image and to store or transmit data in an efficient form. Images algorithms allow us to exploit space and spectral redundancy to reduce the data sent. However, mission specifications tell us that the goal of the mission is to send a set of images of the same location, so we could try to exploit the temporal dimension to predict the pixels of new frames and reduce the global compression rate.

5.1. INTRODUCTION

Video compression [17] can be viewed as image compression with a temporal component since video consists of a time sequence of images. From this point of view, the only “new” technique introduced is a strategy to take advantage of this temporal correlation. It is know that video compression algorithms usually have been used in lossy environments. However, recently data compression community has been working upon lossless algorithms. In fact, most of lossy video compression algorithms can be used as lossless compression algorithms just by removing the quantitation matrix. Of course, this technique won’t return the best compression performance as possible but it could be used as a lossless algorithm.

Anyway, we have to remember the context and the scope of our mission: we have to develop a low-complexity algorithm, and video compression standards are not good examples. We have finally decided to create our own-low-complexity algorithm which it is based in motion vectors, spatial and spectral redundancy exploitation.

5.2. COMPRESSOR

As the video compression algorithm, is more complex than image compression algorithms, we are going to start by presenting the general compressor block-diagram. Figure 42 shows the block diagram.

5.2.1. COLOR DIVISION

First part of the algorithm consists on splitting the set of Bayer Images in two smaller set of images, the first set will be created by green pixels and the second one will use the non-green pixels. Reasons to follow this pattern are based on simulation. Under Matlab platform, we have researched the higher performance upon algorithm for different patterns subdivisions.

To carry out this subdivision, we just have to select odd pixels to create the first image and even pixels to generate the second one.

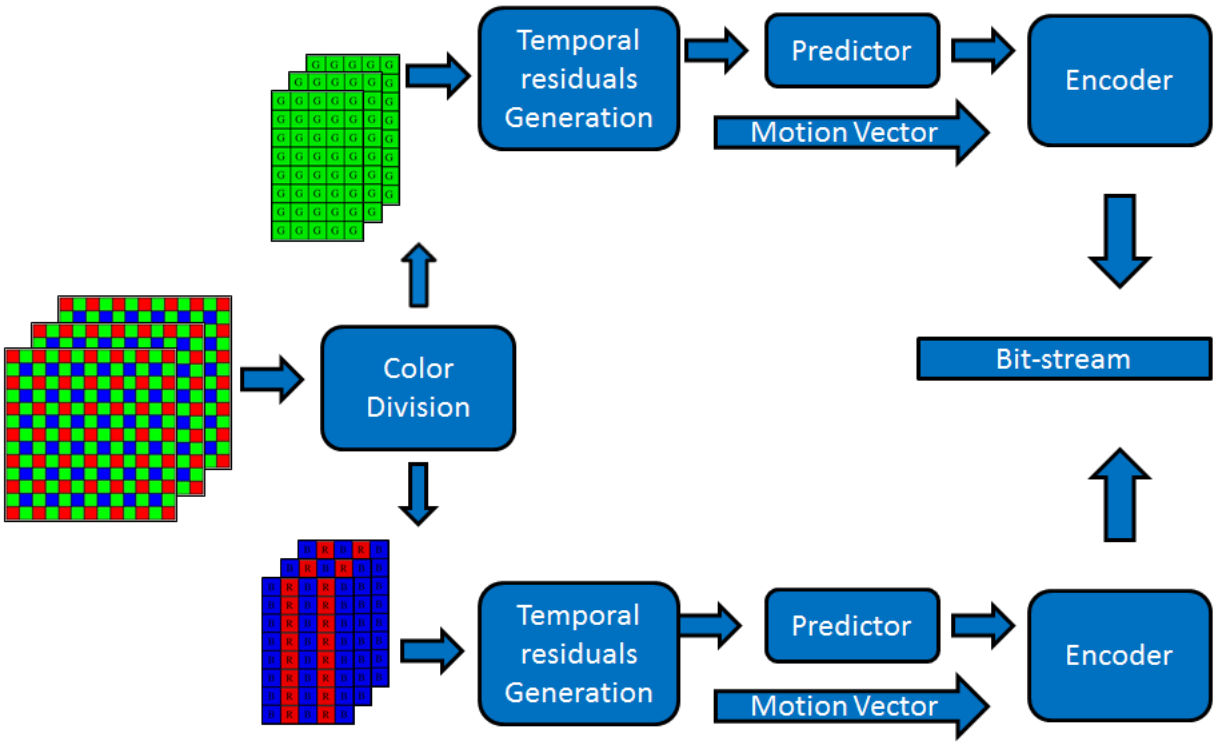


Figure 42- Overview Video Compression Block Diagram

5.2.2. TEMPORAL RESIDUALS GENERATION

The Temporal Residual Generation Block handles motion vectors generation and the Residual Image creation. The methodology follows the steps illustrated in Figure 43.

To sum up, for each subset of two images (from here, we will use frame to make reference to separate images); we are going to find the motion vector which describes the motion between frames. The motion vector illustrates a vector space of two dimensions, the vertical dimension and the horizontal dimension. We have to highlight that as we are not considering fractional motion vector movement our vector will be in an integer vector space. Equation 14 shows the foregoing considerations.

$$\overrightarrow{V} = \begin{pmatrix} V_x \\ V_y \end{pmatrix} \; such\ us\ V_x, V_y \in \boldsymbol{Z} \tag{14}$$

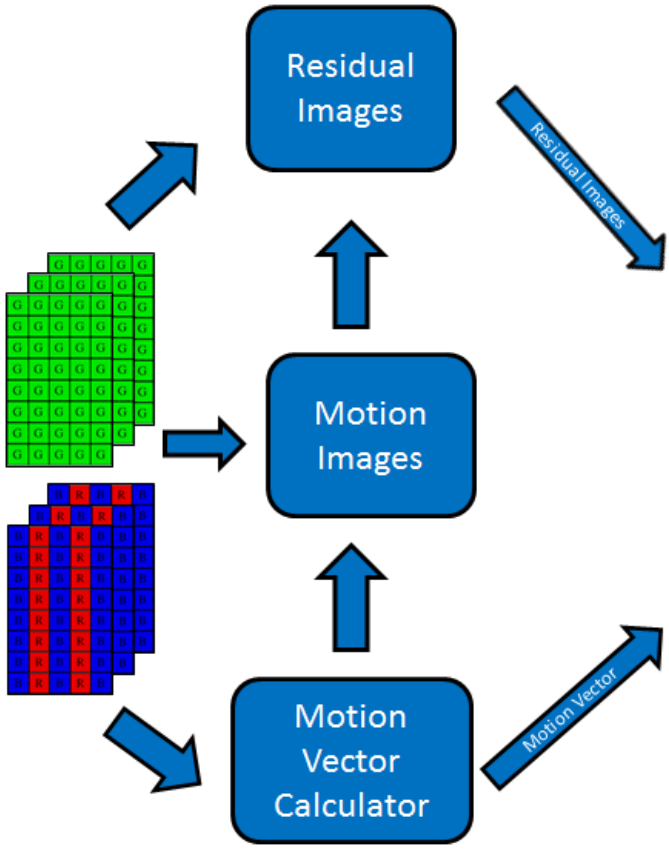


Figure 43- Temporal Residual Generation Block

After that, we will use each motion vector to create each motion image, to do that we will move the first image according to the predicted motion vector. Of course, the newest motion image will have pixels unknown close to edges. If we initialize the unknown pixel to zero, the residual image will have the real pixels of the second images and we are not predicting at all the region of unknown pixels. Another strategy could be, instead of initializing the unknown pixel to zero we can use the average of the first image as a reference value, in order to predict the DC component of the image.

After generating the Motion Images, we just need to create the residual images by subtracting the motion image to the real image (the second one of the subset).

The fact of generating a motion image to create a residual image is based on reduce the temporal redundancy between frames. However, we will still have spatial redundancy, which can be removed. So, the goal is to use the CCSDS – 123 to predict only spatial redundancy in a single band and generate new residuals.

At this point, the following steps of the algorithm are similar to the image compression algorithms developed before. The only change to highlight is into the encoder, due to we have to send the motion vectors. Figure 44 shows a big picture of the following steps.

Now we are going to follow by explaining each block separately.

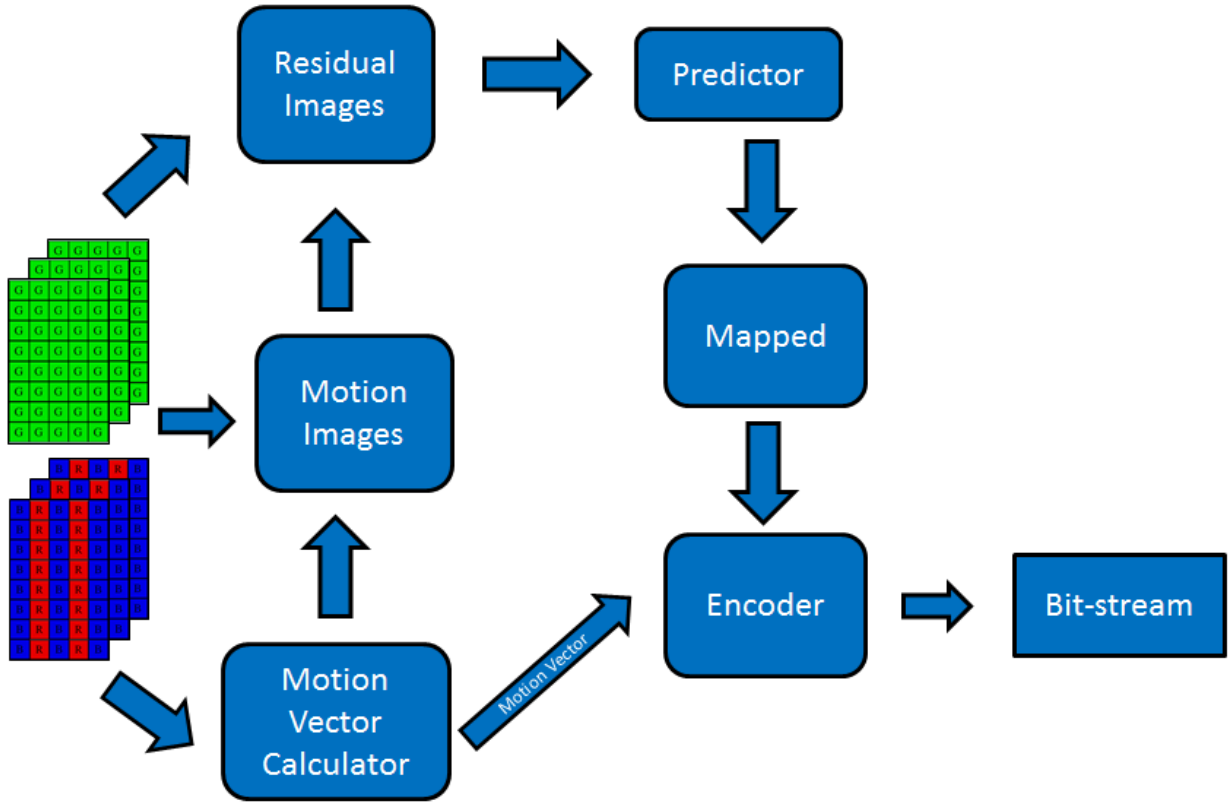


Figure 44- Video Compression Block Diagram

5.2.2.1. Motion Vector Calculator

The motion vector calculator takes care of creating the motion vector. As we said, the motion vector represents an integer bi-dimensional vector space, where each component illustrates the motion over the vertical or horizontal dimensions. To achieve the target of finding the motion vector we should compare the "*actual frame*" with the "*frame before*" and through the differences get the motion vector.

Figure 45 shows a block diagram where it is explained in general terms the methodology to calculate a value which will be used to select the future motion vector on a single region (segment) of the image. After evaluating all the obtained values at every single region, we will be able to decide which will be the most accurate motion vector value.

As it is shown in Figure 45, we take a subset of two frames from the image sequence. In case of being the first iteration, the image will go directly into the CCSDS-123 predictor as the frame before does not exit. Next case will be modeled as subsets of "Actual Frame" and "Frame before". An example is illustrated at Table 1.

Table 1- Subsets of Frames

Subsets	***"Actual Frame"***	***"Frame before"***
1	*Frame 1*	---
2	*Frame 2*	*Frame 1*
3	*Frame 3*	*Frame 2*
4	*Frame 4*	*Frame 3*
…	*…*	*…*
N	*Frame N*	*Frame N-1*

So, the first subset which follows the algorithm will be the subset 2.

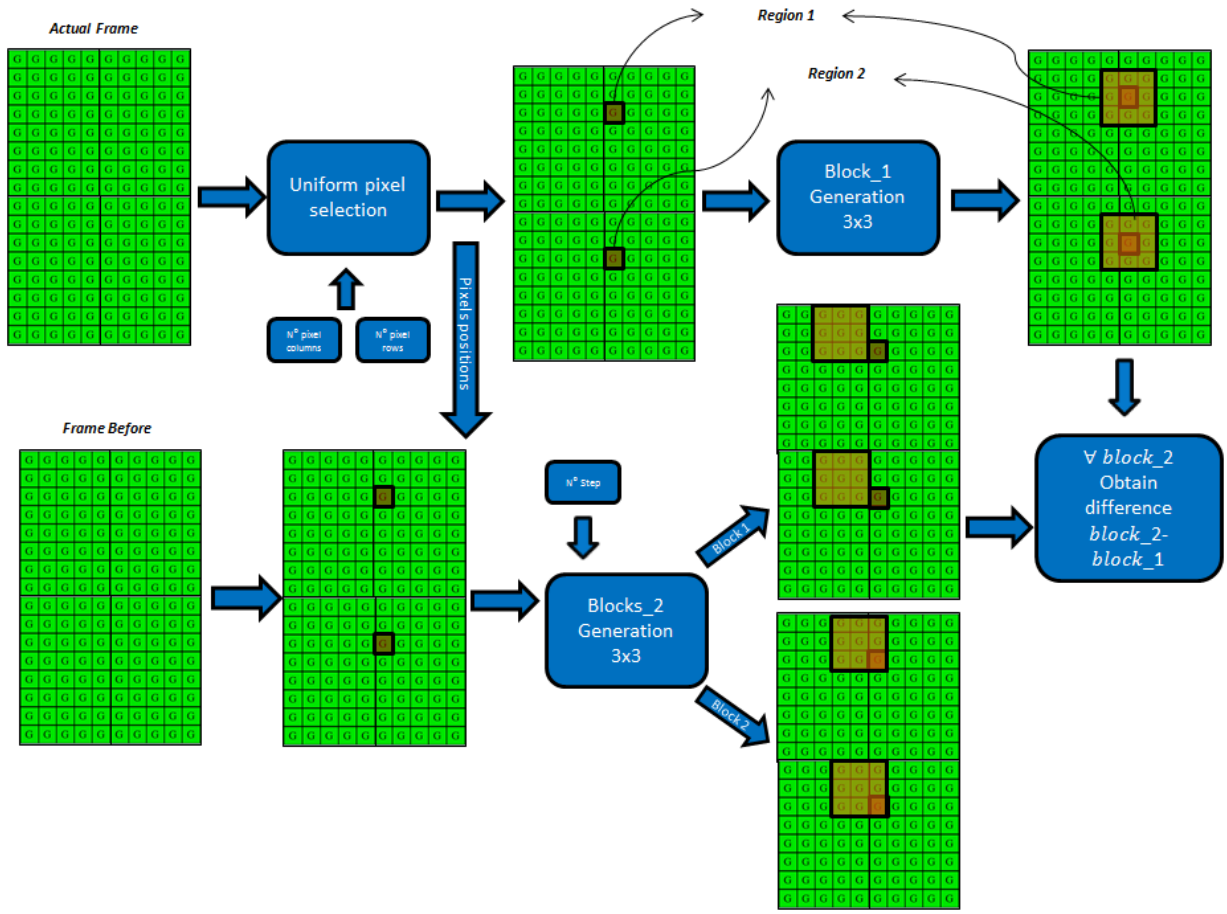


Figure 45- Motion Vector Calculator Diagram Block

The way of identifying an accurate motion vector consist on finding a region within the "Frame before" as similar as possible from a region within the "Actual Frame". An effective manner of finding these regions could be via block comparison. To sum up, we should create small blocks in each frame and define a measurement of likeness. In particular, we have selected the Euclidean distance as we can consider the difference between pixel's blocks as components of a vector. Euclidean distance is defined in Equation 15 and graphic explanation can be found at Figure 46.

$$D(\vec{p},\vec{q}) = \sqrt{(q_1 - p_1)^2 + (q_1 - p_1)^2 + \cdots + (q_n - p_n)^2} \quad (15)$$

Choice of where we should place the blocks can be made following different methodology and theoretical concepts, as selecting a location where signal is more powerful inside of the image or where the variance within the image is higher. In our case, as we use a relative high amount of blocks, we just decided to insert the blocks like in a uniform distribution. Figure 47 shows an example of a uniform distribution block insertion.

Each select pixel will be the center of a 3X3 block, and the representative of an image's region to find the most appropriate motion vector. Figure 48 shows our image with the 3x3 blocks selected.

In order to obtain the most accurate motion vector, we should also create block within "Frame before" to compare the likeness between regions. However, as we need to identify the movement we should create a set of block in every segment of the image. These blocks will be in the surroundings of the central pixel selected in the "Actual Frame". The more block's quantity we will create to compare with blocks created within the "Actual Frame", the better choices we will have to find the most accurate motion vector.

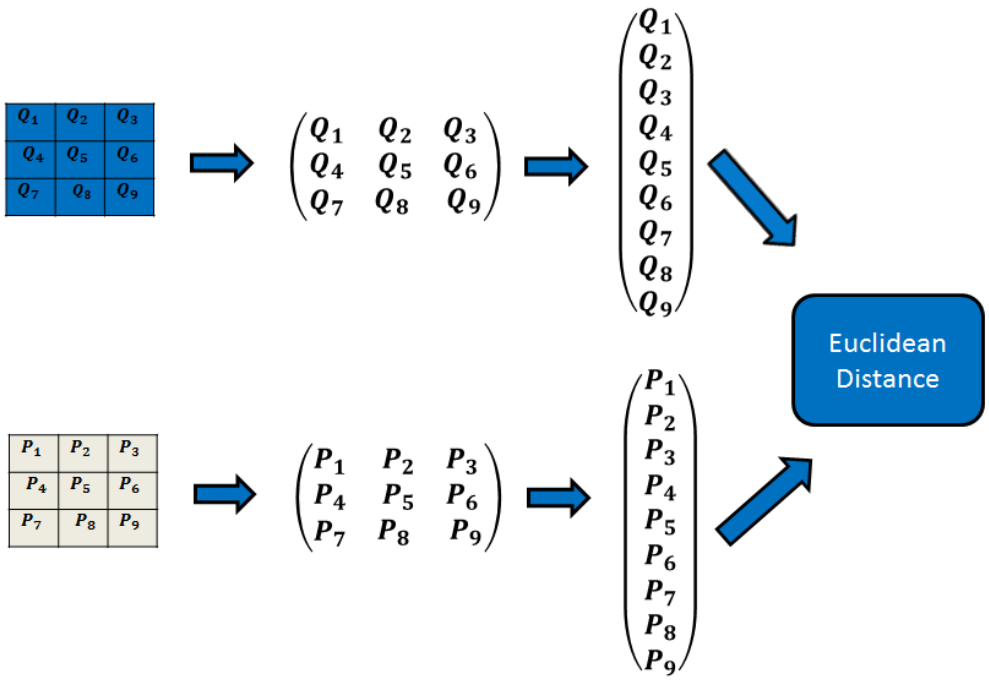


Figure 46- Euclidian Distances

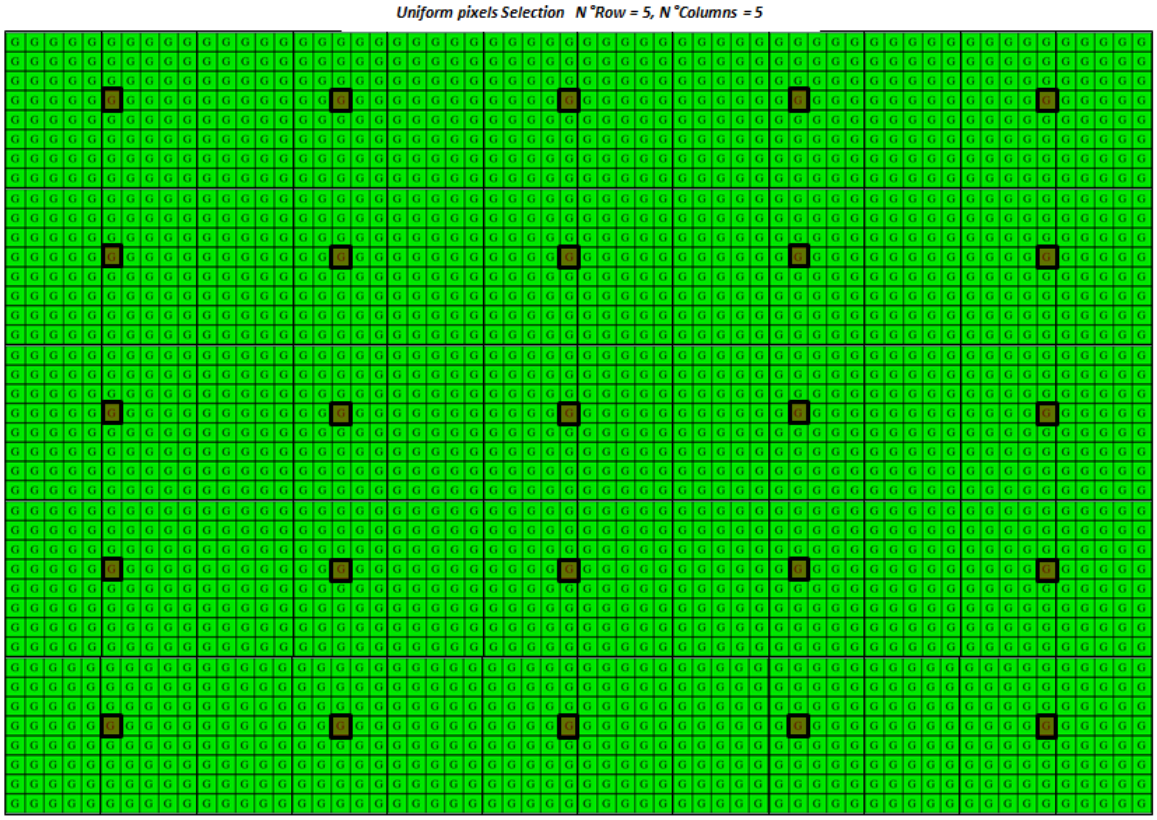


Figure 47- Uniform distribution of Block Insertion

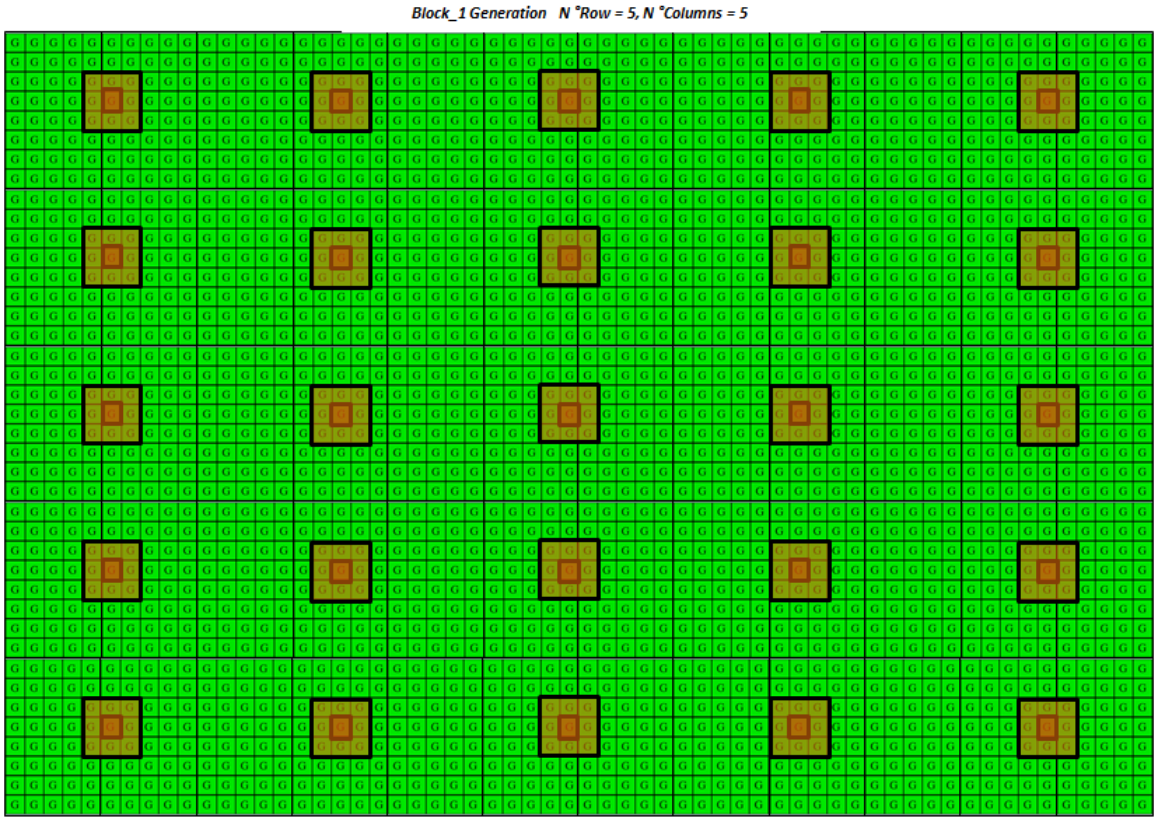


Figure 48- 3x3 Block Selected

However, the more block's quantity we will create to compare, higher complexity in terms of numerical computation we will need. Inside the algorithm we have a parameter which describes how many distance steps shall be checked. Figure 49 shows an example for one step of 1 and Figure 50 shows for one step of 2.

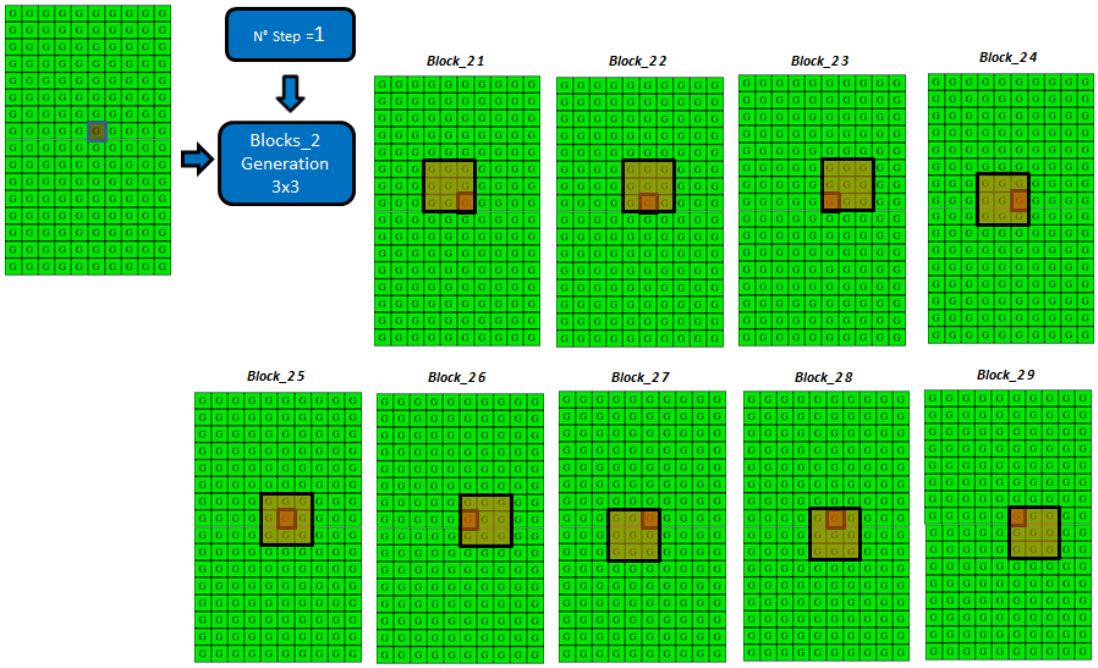


Figure 49- Distances with Step =1

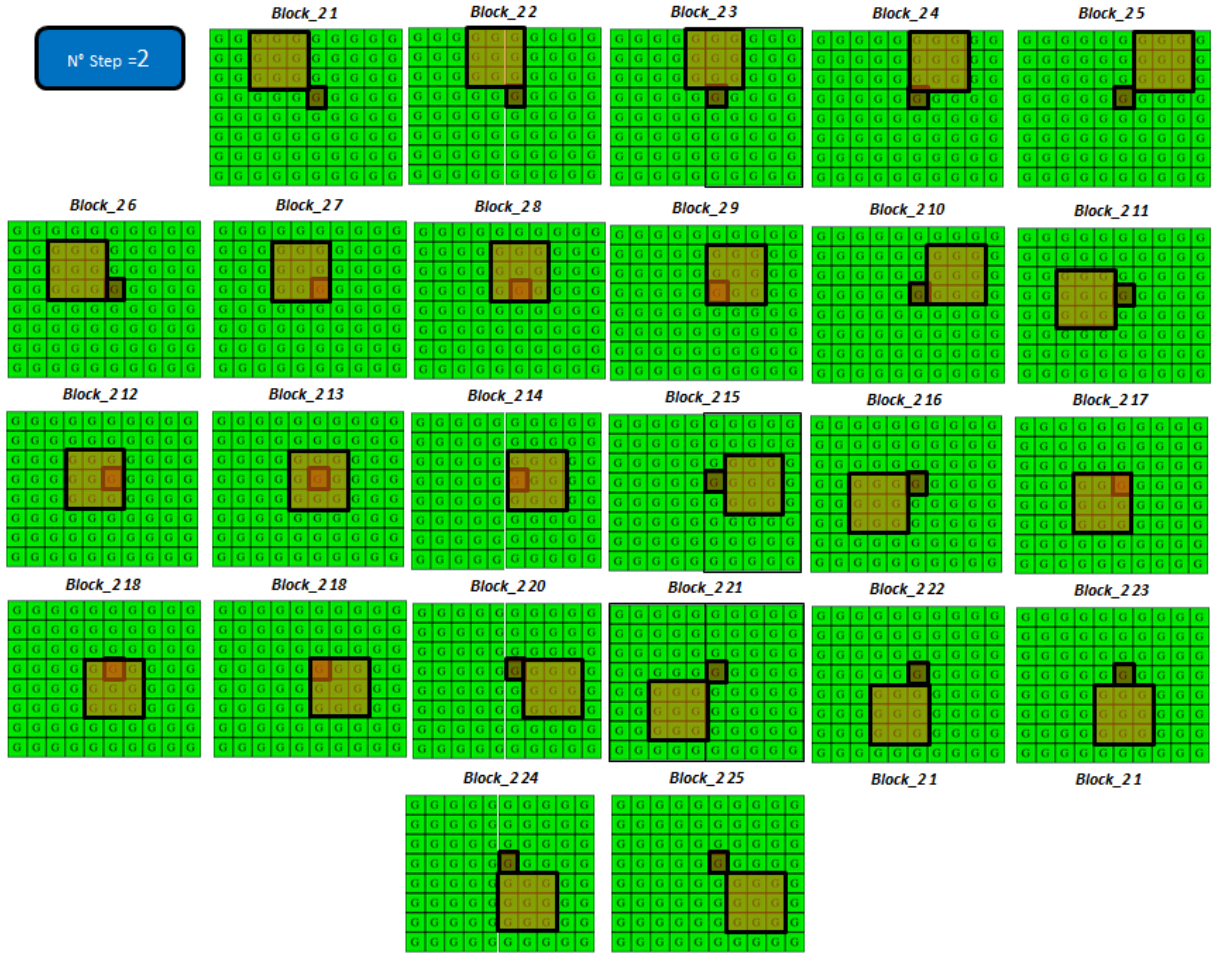


Figure 50- Distances with Step =2

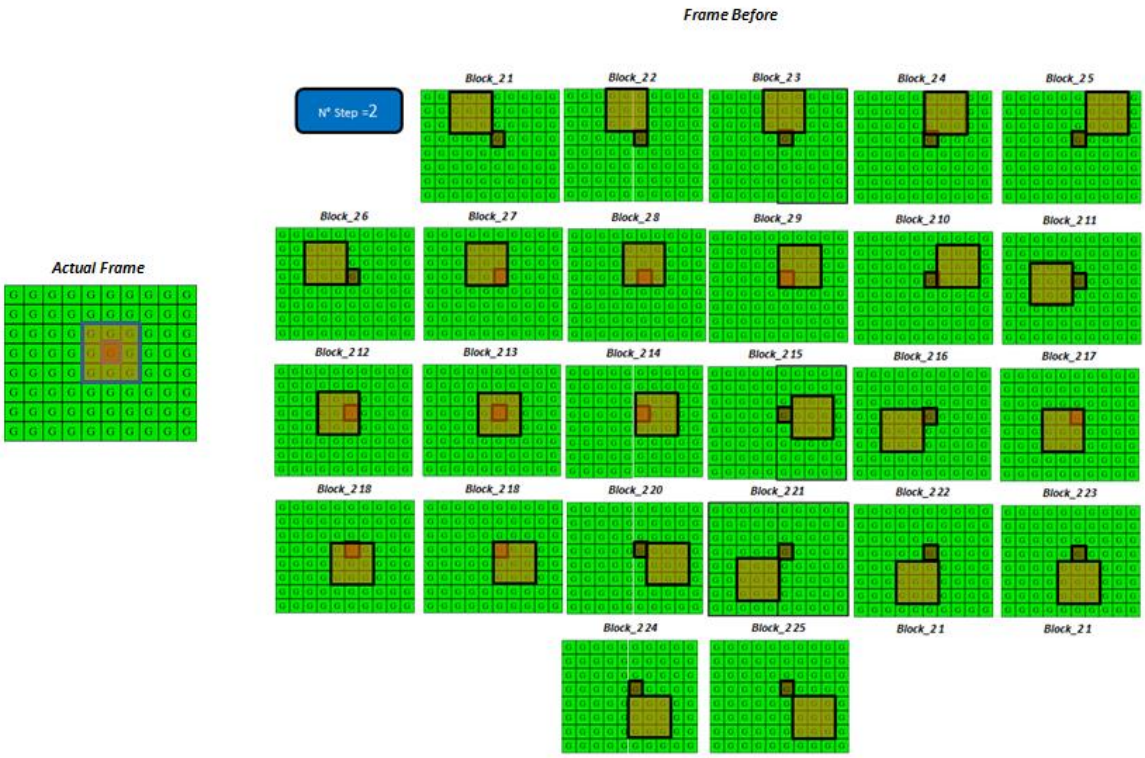
The number of blocks follows the function illustrated in Equation 16, which depends on the number of steps. Let's denote the number of step as N.

$$Num_{blocks} = (2N + 1)^2 \tag{16}$$

The following step consists on generating a vector with all the Euclidian distance between the block of the "Actual Frame" and each block of the "Frame Before". With this vector we will be able to find which blocks will be more likeness and discover which will be the most accurate motion vector at this region of the image.

Between Figure 46 and Figure 51 is explained how calculate the Euclidean distances Vector. The min position of the vector corresponds with the most accurate position and will represent the most accurate motion vector. Figure 52 shows a complete example of how calculating the vector for one region of the images.

Once we have the vector of the first region we iterate the process in each region of the images creation a new vector of motion vectors. As movement within the image should be quite similar, we expect that most of the vectors will be the same. So, next step will be selecting the motion vector which appears with more frequency. Figure 53 Show an example.



$$\vec{D} = (D_1 \quad D_2 \quad D_3 \quad \dots \quad \dots \quad D_{25})$$

Figure 51- Euclidean Distances Vector

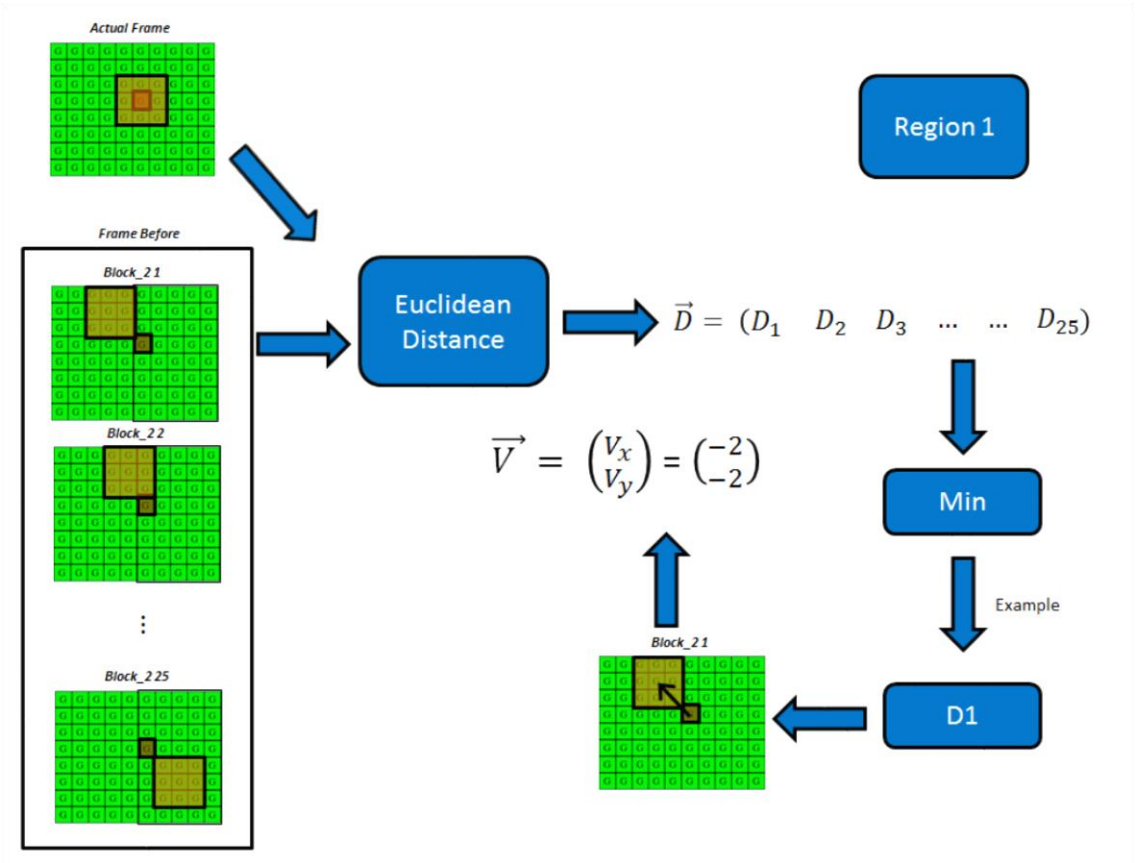


Figure 52- Euclidean Distance Vector for One Region

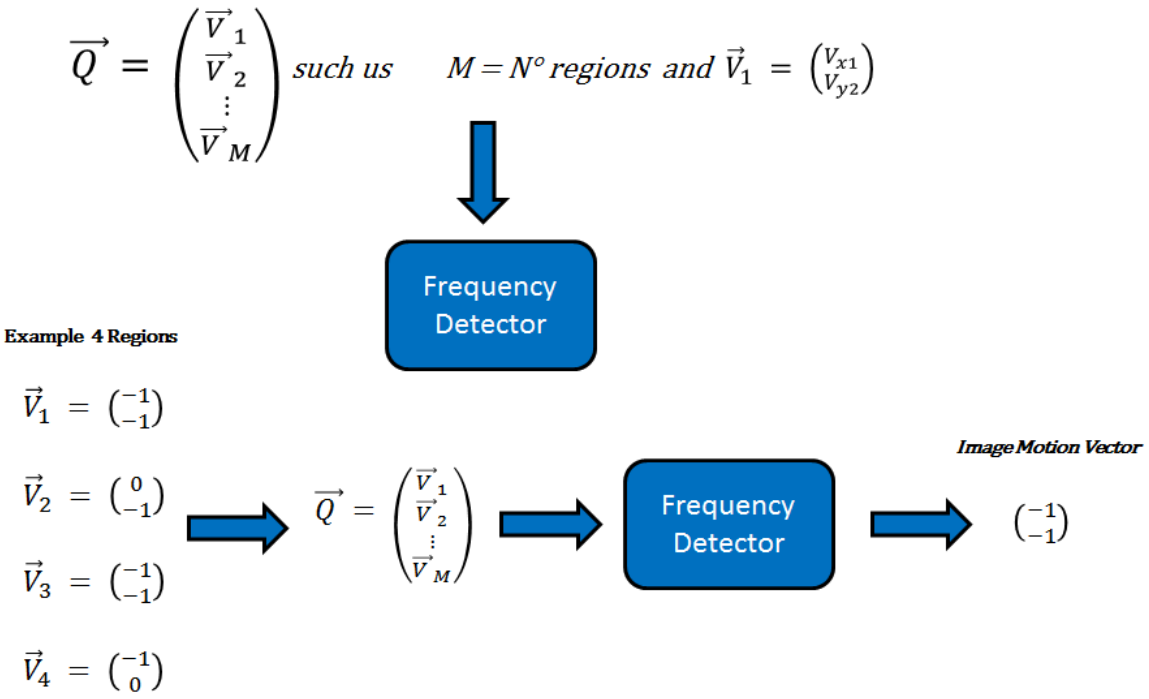


Figure 53- Final Euclidean Distance Vector (All Image)

5.2.2.2. Motion Images

The following steps have to be completed for each subset of images. The goal is to create a prediction image of the actual frame. Figure 54 shows the block diagram. The prediction image takes into account the motion vector which has been calculated in the foregoing block.

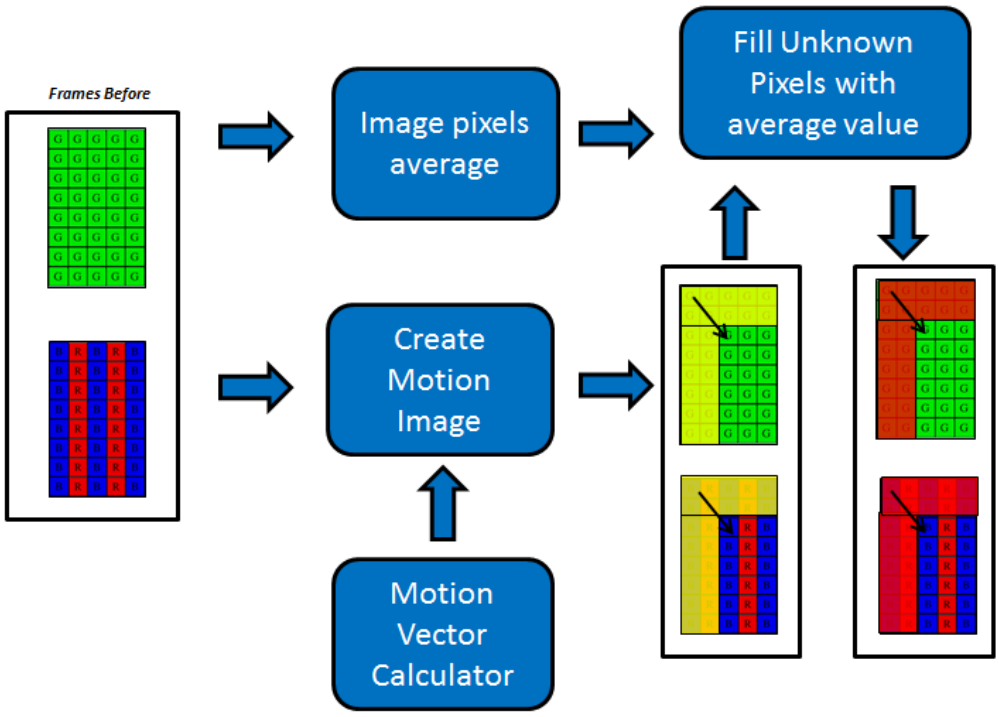


Figure 54- Predicted Image Creation

Concerning this method, we highlight that there are 4 possible movements which should be carefully handled when we will implement the C code. Figure 55 shows the four alternatives.

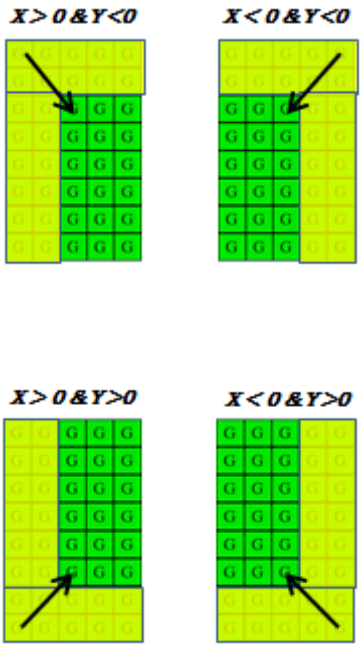


Figure 55- Available Movement

5.2.2.3. Residual Images

To obtain the residual images we only have to make the difference between the synthetic motion image and the "Actual Frame". The resulting image is the residual images which we will send to the CCSDS 123 predictor block.

Next block follows the methodology of the image compression algorithms so we do not focus in a recurrent explanation.

5.3. DECOMPRESSOR

Once we receive the bit stream at ground segment, we have to restore the data acquired by the satellite's camera. To do that, we are going to follow the series of steps illustrated in Figure 56.

Decoder and Un-predictor blocks follow the same steps than in the image decompression algorithm. The newest block that must be explained is the temporal images generator. The goal of this block consist on restoring the Green or Non-Green pixels from the temporal residuals. As in the compression part, the first image was sent directly to the CCSDS-123 predictor, the first residuals after the un-predictor will be the Green and Non-Green first images. These images, along with the motion vectors could generate the motion images. After that, we only have to make the sum between the motion image and the un-predictor residuals. Once, we have Green and Non-Green Images, we only have to sort the pixels and create the Bayer Pattern image in the Color Fusion Block.

As image decompression algorithms, we have the choice of creating a demosaicing step, but this part will be explained further on.

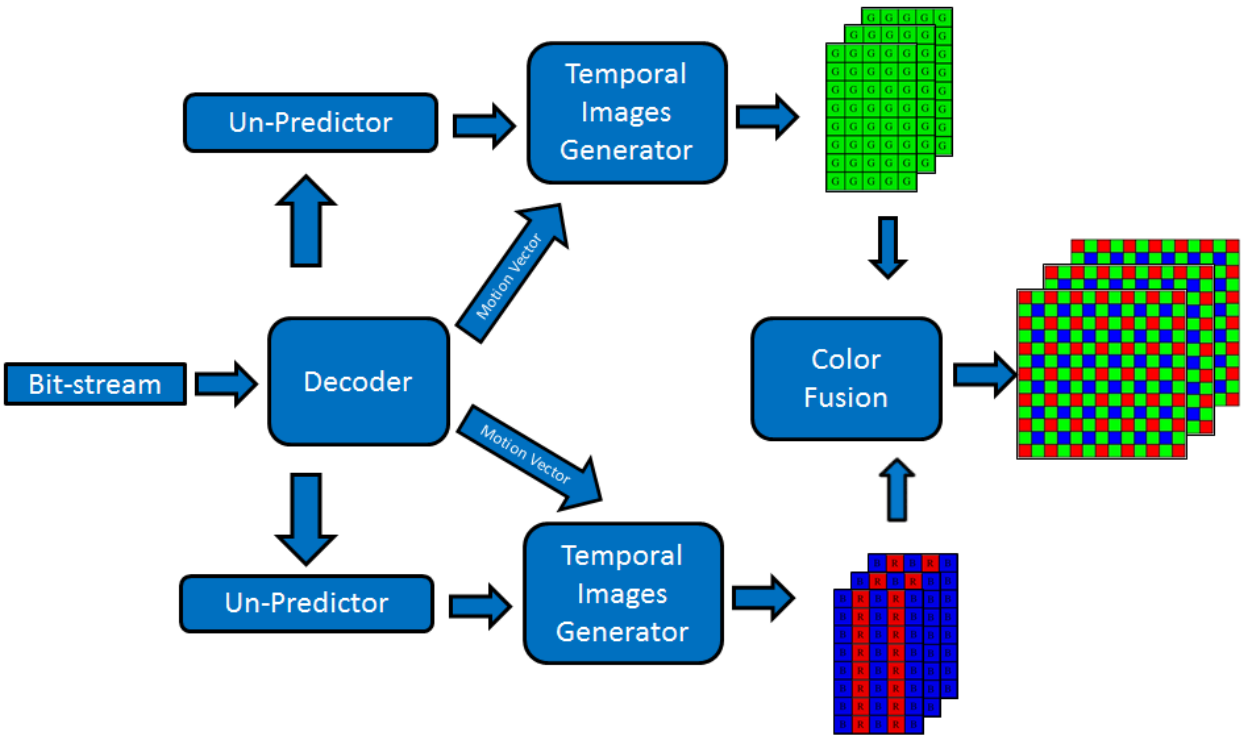


Figure 56 Overview Decompressor Diagram Block

6. VIDEO CODER ALGORITHM 2 LOW COMPLEXITY

Summary

Video compression algorithm allows us to exploit the temporal dimension to estimate pixels between frames. The algorithm explained before, is based on the most basic techniques to estimate motion within of set of sequences. After evaluating the performance of the algorithm, we have noticed that complexity could be lowered without losing much in compression rate. To sum up, the goal of this algorithm consists on reducing the complexity of the algorithm developed before.

6.1. INTRODUCTION

As we have said, the main goal of developing this algorithm consists on reducing the complexity without a high loss in compression rate. Complexity can be seen as number of operations or as time computing, anyway the objective is to reduce this parameter. To do that, we have looked at the performance in terms of time computing and we have found that CCSDS-123 is the biggest contributor to the time-consumption. So, the incoming algorithm tries to avoid the use of CCSDS-123 predictor as much as possible. Moreover, we have tried new techniques to improve motion vector collecting.

As we have made in previous algorithm we will start explaining the compressor.

6.2. COMPRESSOR

The block diagram of the algorithm is presented in Figure 57.

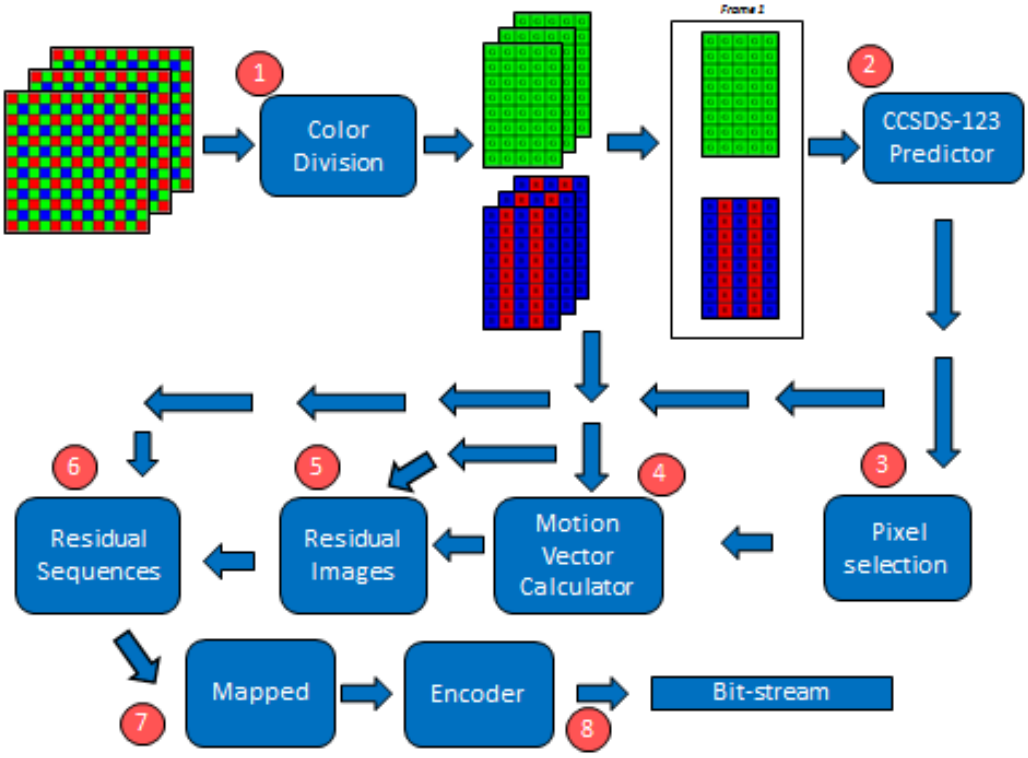


Figure 57- Video Compression 2 Block Diagram

First remarkable change consists on not used the CCSDS-123 predictor for all the residuals, instead of that we only have introduce the first frame due to it cannot be predicted by a previous frame (step 2). Residuals obtained from the step 2 will be used to select the reference pixels (step 3); which will be characterized for being in a zone with high variance. Those pixels have the same function as pixels of Figure 47. Once; we have selected the pixel; we will obtain the Motion Vector and the residual images (step 4-5). Those residuals will be directly mapped and encoded without any spatial predictor (step 6-7-8).

To sum up; the main differences between this algorithm and the first video compression algorithm are the pixel selection, which it is used to create the blocks at motion vector detection process and which uses a new technique based on signal detection. The second difference is that we only use the CCSDS-123 Predictor for the first frame to obtain the residuals; the rest of the residuals are directly generated by temporal prediction, that is, through the difference between the new frame and the estimate frame (Frame before with a predicted movement generated by the motion vector).

6.2.1. PIXEL SELECTION

The new technique, which is used at pixel selection block, is based on signal detection. As we have seen in Figure 57, input data belongs to the CCSDS-123 Predictor block. Predictor's residual data shows us signal changes places, which are the most appropriate to follow the movement of the images.

Let's show this phenomenon by using the first partial derivate in horizontal dimension as a predictor. In Figure 58 we are able to see signal changes upon the horizontal dimension.

Figure 58- Residuals data Prediction Order 1

Taking into account the previous phenomenon, we will decide where we should place the central pixel to generate a search of motion vectors for each region. Figure 59 shows the block diagram for obtaining the search.

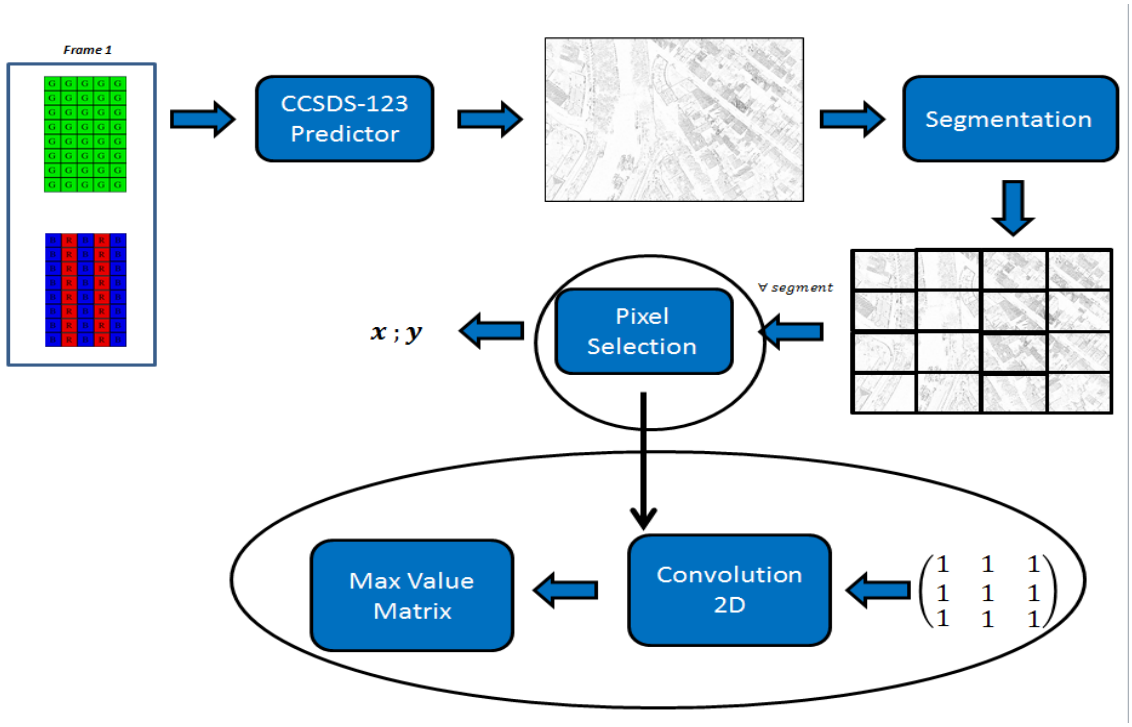


Figure 59 Pixel Selection Search

6.3. DECOMPRESSOR

Concerning the decompressor algorithm, firstly we will decode and de-map the bit-stream information to obtain the residuals. The first frame will be restored by the CCSDS-123 un-predictor and it will be used to restore next frames along with motion vectors and de-mapped residuals.
Once all the frames will be rebuilt, the color De-Division block will handle them to restore the CFA image.

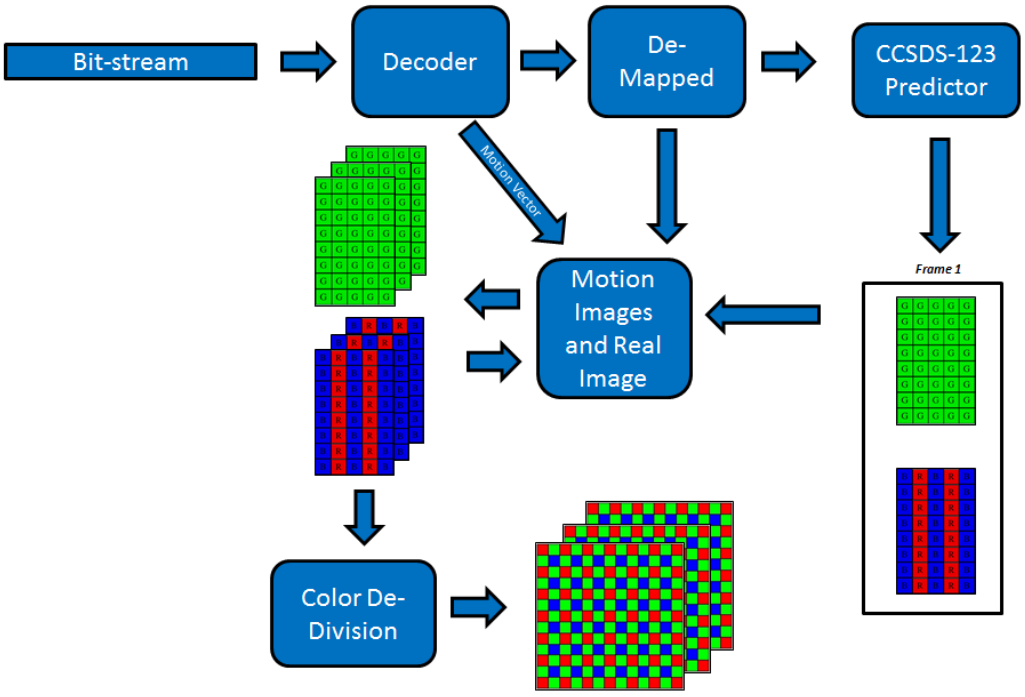


Figure 60 Video Decompressor Diagram Block

Hence, the decompression algorithm will only use the CCSDS-123 un-predictor upon the first frame, achieving low-complexity not only in the compressor step, but also in the decompressor. Once again the residual images generator block differs from compressor block because we have to sum the residuals to the motion image to get real frames. Figure 61 illustrates it.

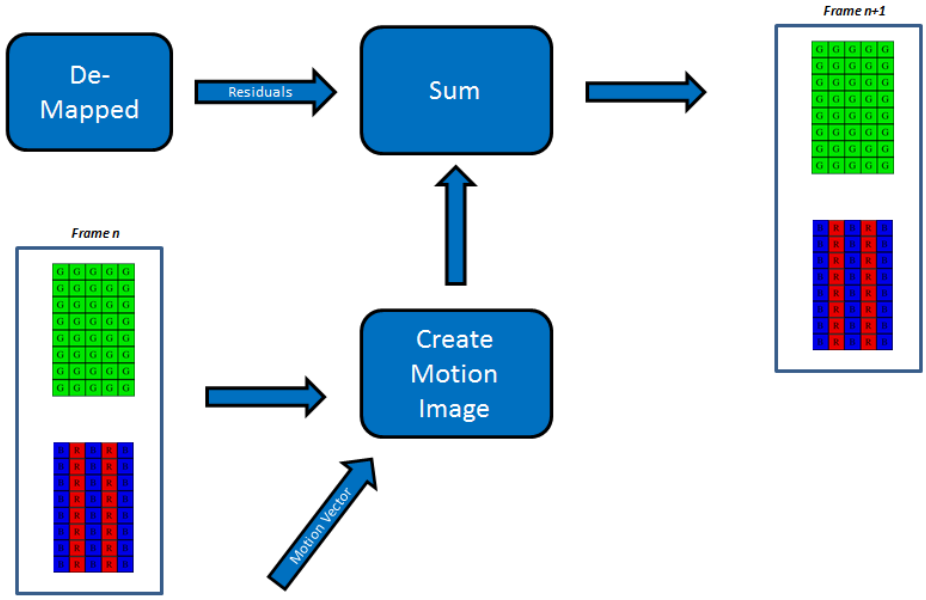


Figure 61 Original Pixel Reconstruction

7.SEGMENTED FINAL ALGORITHM

Summary

Video compression algorithm gives us high performance in terms on compression ratio and even in low complexity. However, after testing all the algorithms and taking into account that rotation movement could appear, we have selected the image compression algorithms as a final method. Moreover, inside the method and as a strategy to mitigate the transmission loss; we have implemented a segmentation method.

7.1. INTRODUCTION

The previously defined compression algorithms share a common characteristic; each decompressed pixel depends on the precedent pixel. What will happen in case of any packet loss…?. Figure 62 shows us the answer.

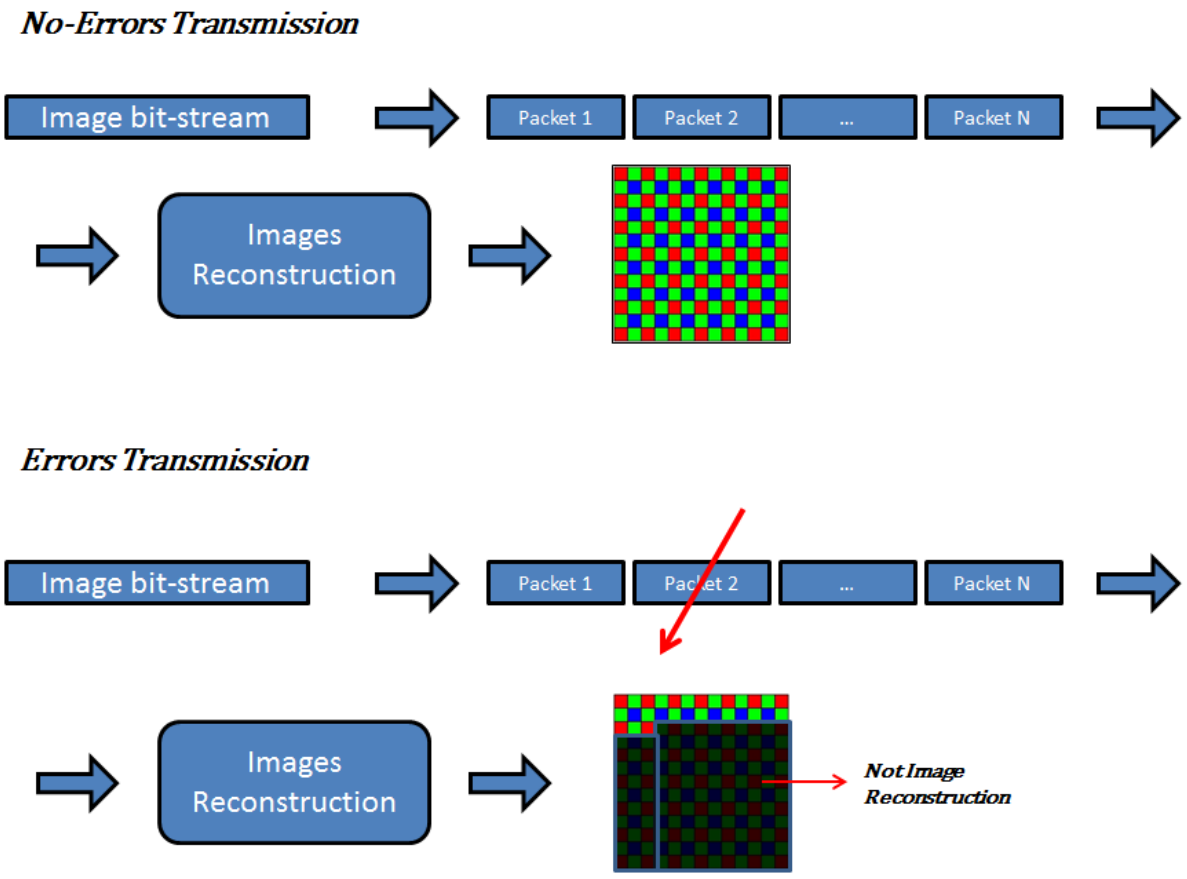


Figure 62- Error Transmission Phenomenon

Any packet lost will involve to not being able to rebuild the rest of the image. If we send the images in N packet, image acceptable rate will be:

$$Arate = 1 - \frac{i}{N} \quad such\ us \quad i = First\ lost\ packet \ \ and \quad N = N°\ packets \tag{17}$$

If the image residuals are encoded in 20 packet and we lose the first packet; the twenty packets will be for nothing:

$$Arate = 1 - \frac{20}{20} = 0 \tag{18}$$

Segmentation technique is presented below. Figure 63 illustrates the methodology.

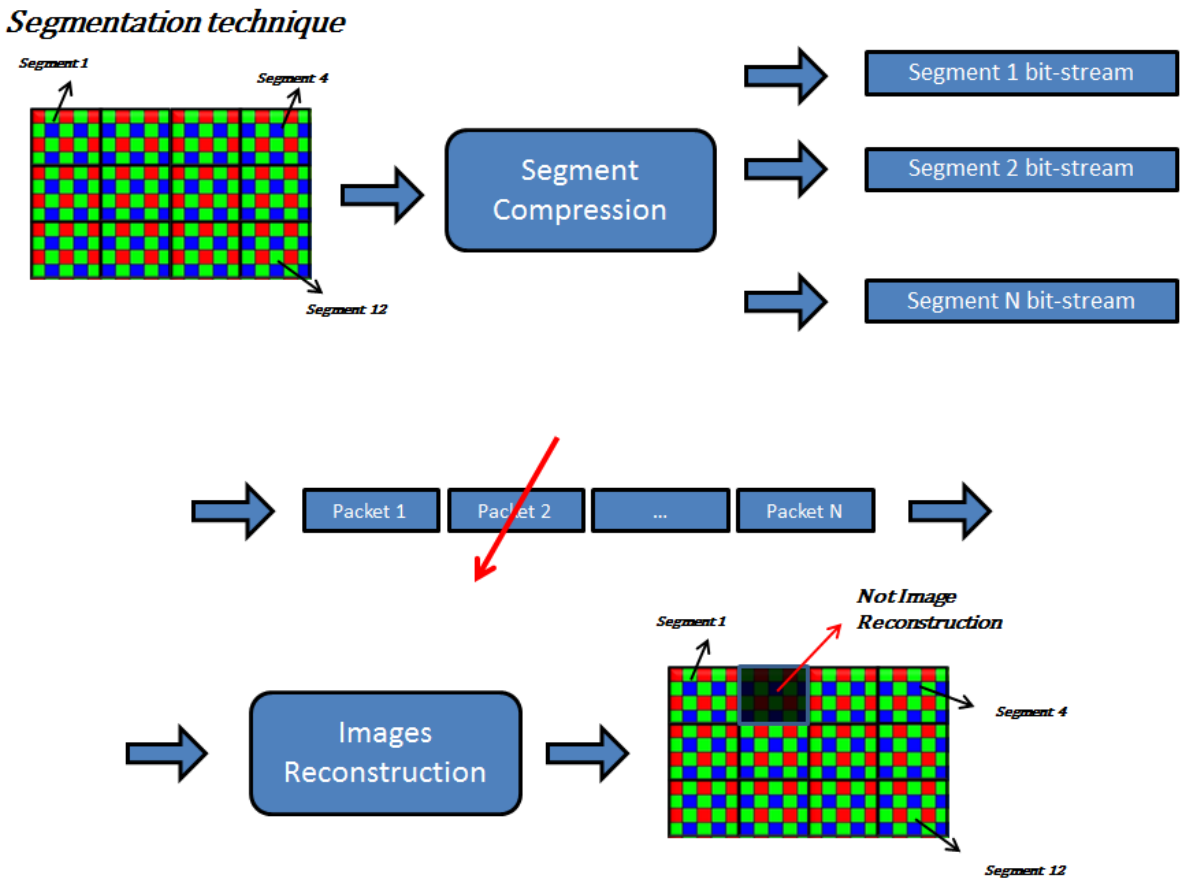


Figure 63- Segmentation Technique

The method consists on splitting out an image in small segments in order to apply separately the compression algorithm. Each segment will be sent within a unique packet and in case of any loss; only on image piece corresponding to one segment will be lost.

The newest acceptable packet rate will be:

$$Arate = 1 - \frac{n_l}{N} \quad such\ us \quad n_l = N°\ of\ lost\ packets \ \ and \quad N = N°\ packets \tag{19}$$

7.2. COMPRESSOR

This algorithm consists on a reposition of the precedent Image compression algorithms. As we are not sure how the images are acquired, we are going to introduce as much choices as possible in terms of algorithm selection. First step handles the image segmentation. Each image of 2048 x 2048 pixels will be divided in 64 segments of 256 x 256 pixels. After that, each segment will be the input data of the algorithm and output data will be a new sent packet.

In both Predictor and Encoder blocks, we still have the same functionalities as in the previous algorithm. We only have one difference which consists on how to create the header of each packet. As we have several possible choices concerning the pixel preprocessing, we should send it within the header since it will be a required parameter along decompression step.

Figure 64 illustrates this segmentation process

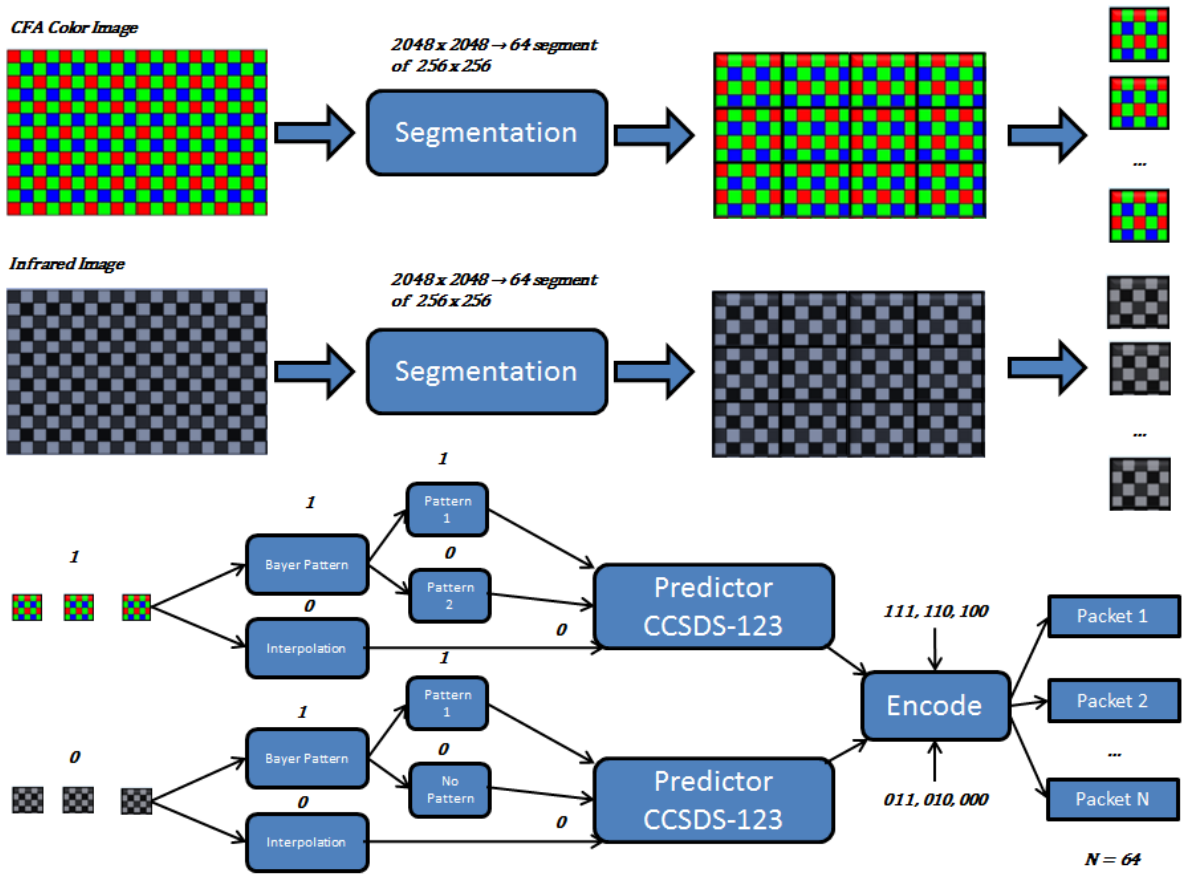


Figure 64- Overview of the Segmented Compression Algorithm

7.3. DECOMPRESSOR

Decompressor handles to rebuild the real image from the packets. The process consists on the following steps. Firstly, we decode each packet to obtain required information like size of the image, method of pre-processing or the residual to restore the images. Residuals are the Un-predict block input data and pixels will be the output data. Pixels can be real or estimated and might need to be restored, therefore we will introduce the data within the De-preprocessing block and we will obtain a single real segment of the image.

We iterate the process for each delivery packet, thus when completed over each one of the 64 packets we will be able to reconstruct the entire image. In case of any corrupted packet, we will not be able to decompress the segment and we will fulfill the segment through padding.

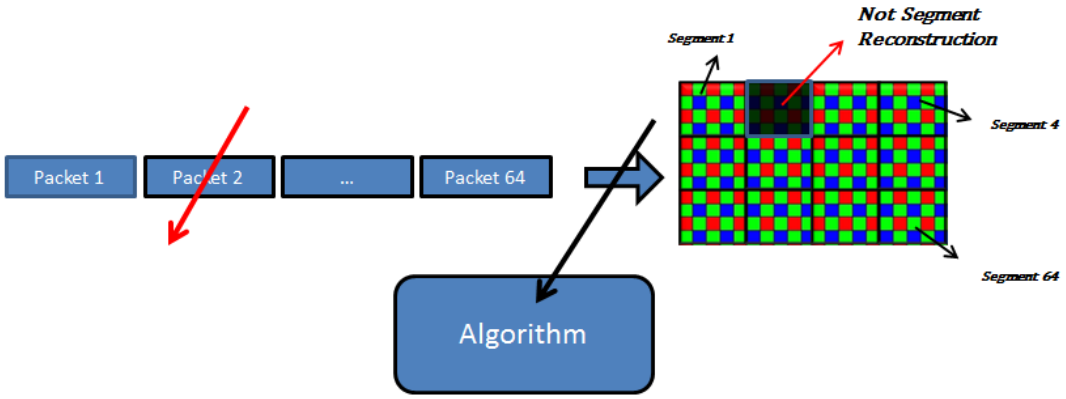


Figure 65- Error Reconstruction mechanism

Underline that, as the compression header has encoded new information, decompression step should be able to invert the process. Header information is needed to decode the residuals, Un-predict the pixels and to De-preprocess information.
Figure 66 illustrates the block diagram of the decompression algorithm.

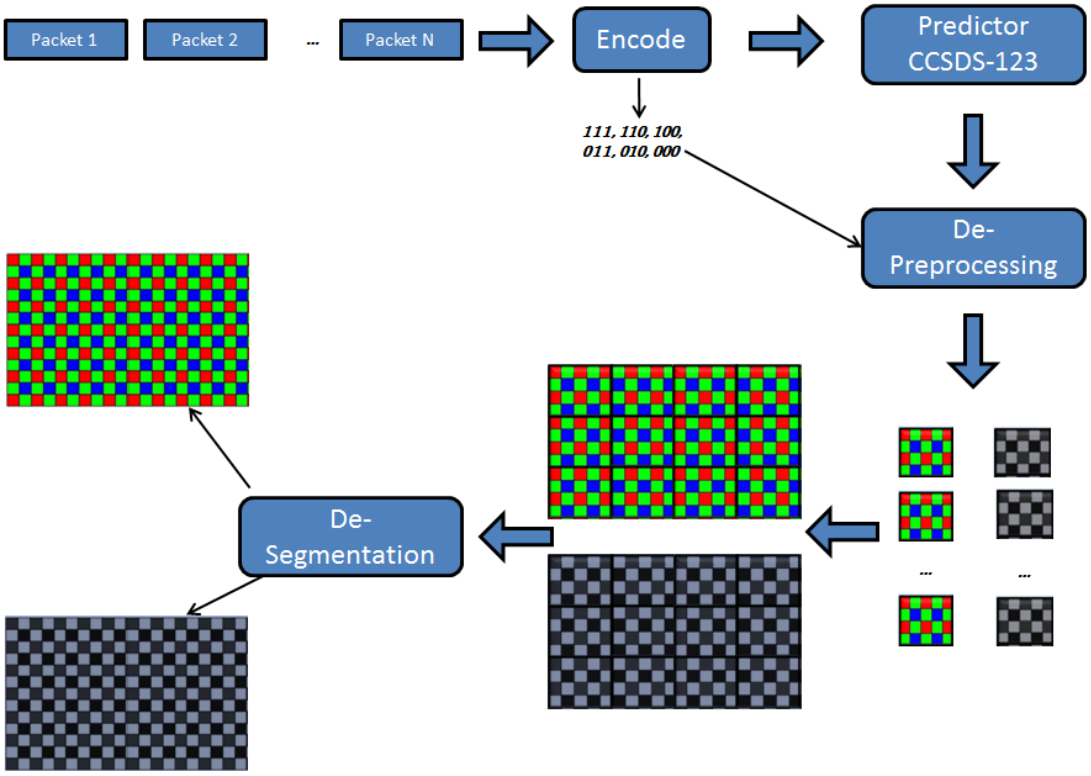


Figure 66- Overview of the Segmented Decompression Algorithm

8. EFFICIENT SEGMENTED FINAL ALGORITHM

Summary

After testing this last algorithm on the Ninano board, we saw that technical specifications were achieved, concerning the complexity time and compression rate. However, once we installed the operating system (Xtratum see part 6) inside the board, the time specification was not fulfilled. The Hypervisor introduce a big delay concerning the memory access. The goal of this algorithm is to find an algorithm fast enough to accomplish the time specification without losing a huge compression rate performance.

8.1. INTRODUCTION

After studying the performances of the CCSDS-123 algorithm, we noticed that the predictor need a lot memory access because we need to update the weight, the local differences, and the local sum. Moreover we must make a multiplication between the weight vector and the local differences vector per each pixel. It is true that we can make in real time the local differences and the local sum to avoid unnecessary memory access, but instead we decided to create a new predictor algorithm based on low cost computation. Concerning the entropy encoder, we are going to use the block adaptive encoder because it gives us better performances not only in compression rate, but also in complexity.

8.2. COMPRESSOR

The algorithm has 3-4 possibilities depending on which kind of images we are going to compress. All the differences are inside the predictor block. The first possibility consists on considering the images as a unique band images. The second one consists on splitting the image in two different spectral bands, first one will be a composition between blue and green pixels and the second one between red and green. The last case consists in three band image decomposition after that each band will be treated independently. Figure 67, Figure 68 and Figure 69 show all the possible cases.

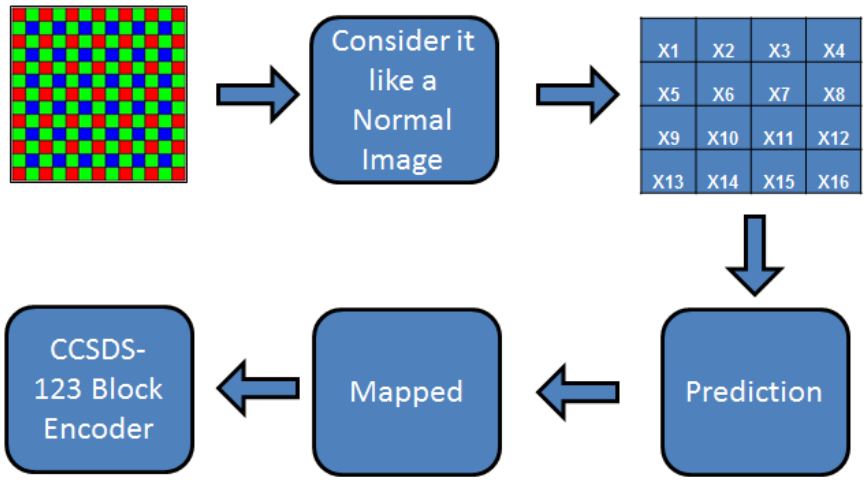


Figure 67- First Case Efficient Algorithm

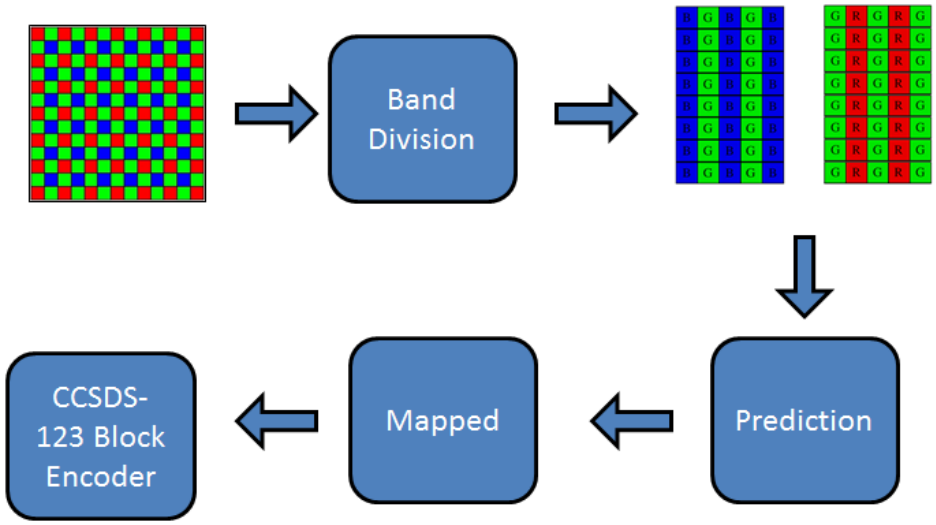


Figure 68- Second Case Efficient Algorithm

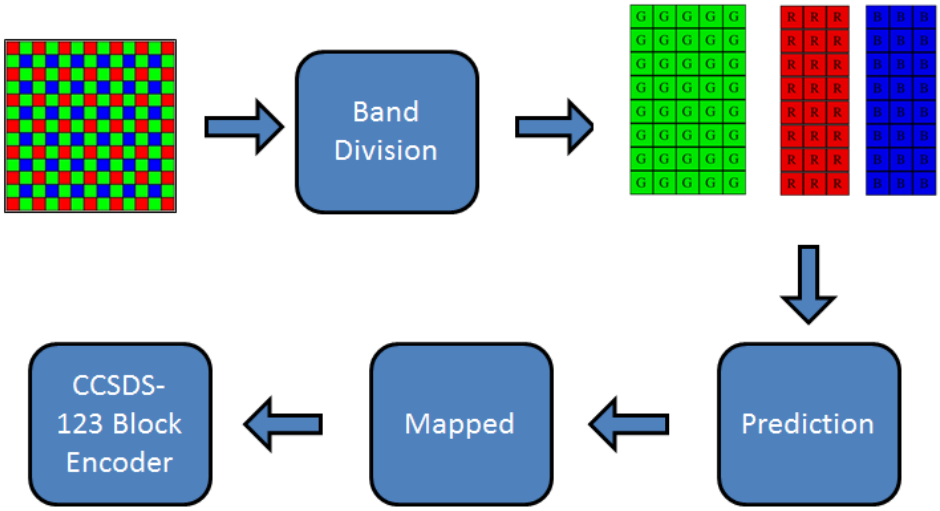
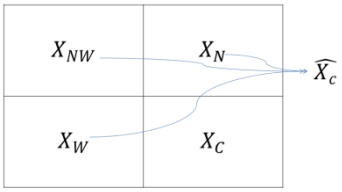


Figure 69- Third Case Efficient Algorithm

8.3.PREDICTOR

This section manages the explanation of each predictor. To explain each case we are going to start by defining the next variables.

- $X_N = North\ pixel$
- $X_W = West\ pixel$
- $X_{NW} = North - West\ pixel$
- $X_C = Actual\ pixel$
- $\widehat{X_c} = Estimated\ pixel$
- $R = Residual$



We define the residual as:

$$R = \widehat{X_c} - X_C \tag{20}$$

The goal is to obtain a residual as low as possible, to do that we must predict $\widehat{X_c}$ as an integer number, which will be as similar as possible of X_C.

8.3.1.FIRST CASE PREDICTOR

We treat the image as a unique band, this method can be use when the image has dark colors (blue, red and green pixels are quite similar) or when we are trying to compress an infrared image. To obtain the residuals we can develop the next equations. These equations are graphic represented in Figure 70.

- *First Sample*

$$X_N = 0, \quad X_W = 0, \quad X_{NW} = 0, \quad \widehat{X_c} = 0, \quad R = X_C \tag{21}$$

- *First Row*

$$X_N = 0, \quad X_W \neq 0, \quad X_{NW} = 0, \quad \widehat{X_c} = X_W, \quad R = X_W - X_C \tag{22}$$

- *First Column*

$$X_N \neq 0, \quad X_W = 0, \quad X_{NW} = 0, \quad \widehat{X_c} = X_N, \quad R = X_N - X_C \tag{23}$$

- *Central Sample*

$$X_N \neq 0, \quad X_W \neq 0, \quad X_{NW} \neq 0, \quad \widehat{X_c} = \frac{X_N}{4} + \frac{X_W}{2} + \frac{X_{Nw}}{4}, \quad R = \left(\frac{X_N}{4} + \frac{X_W}{2} + \frac{X_{Nw}}{4}\right) - X_C \tag{24}$$

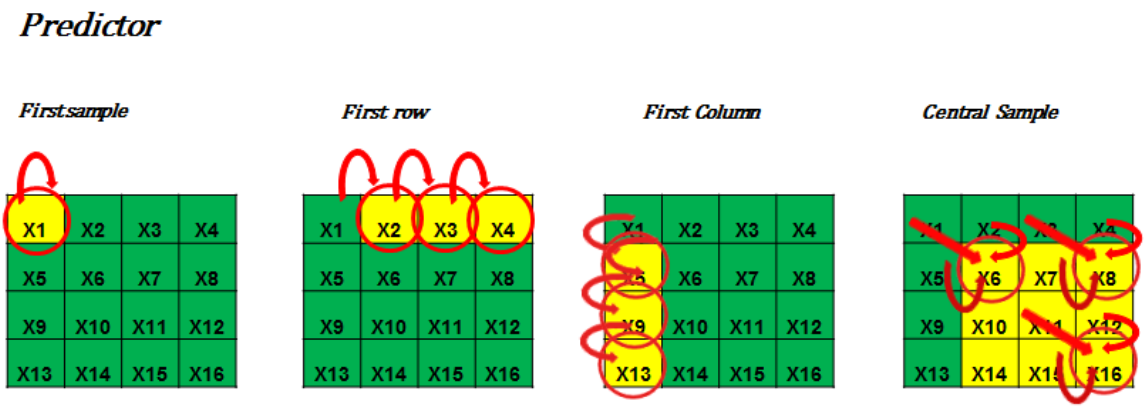


Figure 70- Predictor 1 Algorithm

As it is shown, each residual can be created with a maximum of two sums and one rest because of all the division can be obtained by data shifts (shift the data is free). Moreover, as X_N and X_W are X_{NW} and X_C of the next sample, we can store the values in the cache to avoid a new memory access. To sum up, each residual (central sample) will be obtained though:

$$2\ Read\ Memory\ Access, +3\ Sums + 1\ Write\ memory\ Access \tag{25}$$

8.3.1.SECOND CASE PREDICTOR

Second predictor can be seen as the first one but using two independents band. The number of operations could seem exactly the same as the first predictor, however as now we split the data in two bands we will have two first columns, therefore we will reduce the number of memory access and operations. Anyway this gain can be neglected. Figure 71 shows as the new pattern for predict and Figure 72 the same pattern but taking into account the real indexation.

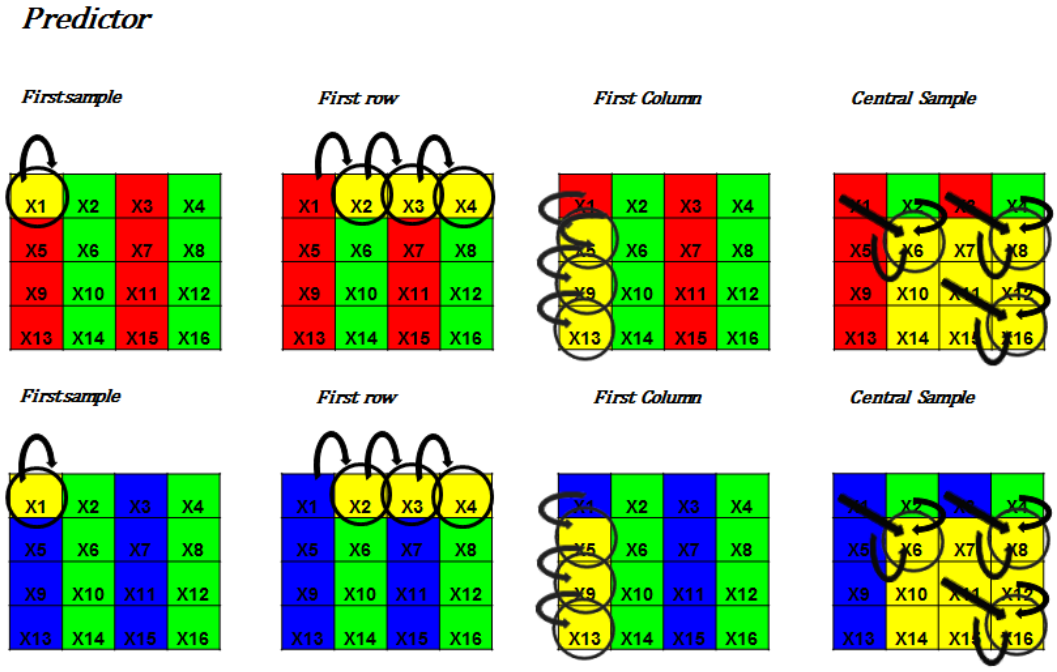


Figure 71- Predictor 2 Algorithm

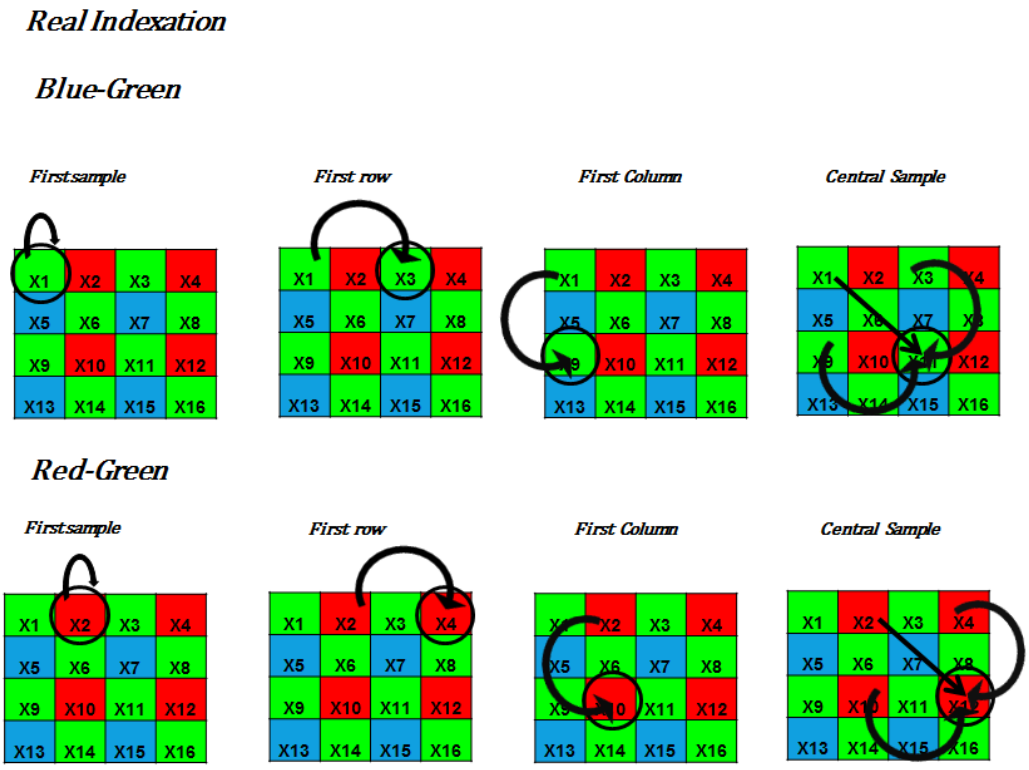


Figure 72- Predictor 2 Real indexation Algorithm

8.3.1.THIRD CASE PREDICTOR

At third predictor, we split into three separate bands. Each band corresponds to the green, red and blue color. The size of each band will be $1024x2048$ for green band and $1024x1024$ for red and blue bands. Figure 73 shows the pattern for the prediction.

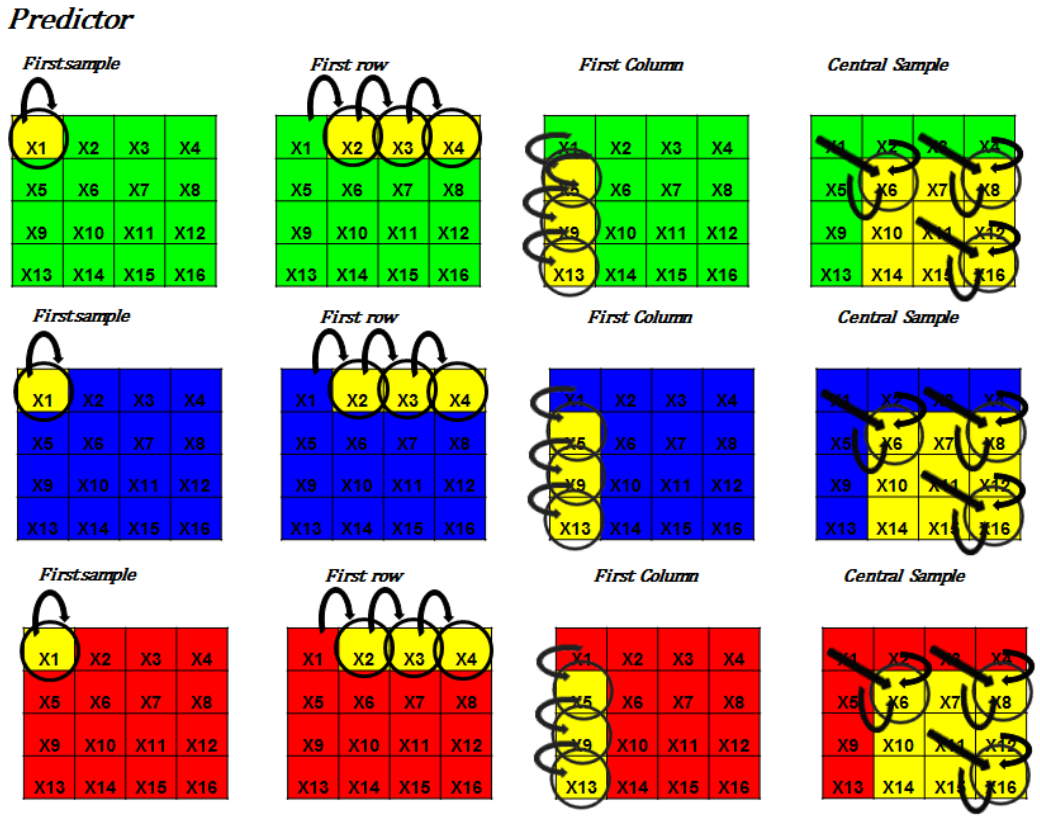


Figure 73- Predictor 3 Algorithm

Concerning the real indexation, Figure 74 shows how to implement that for blue and red color bands.

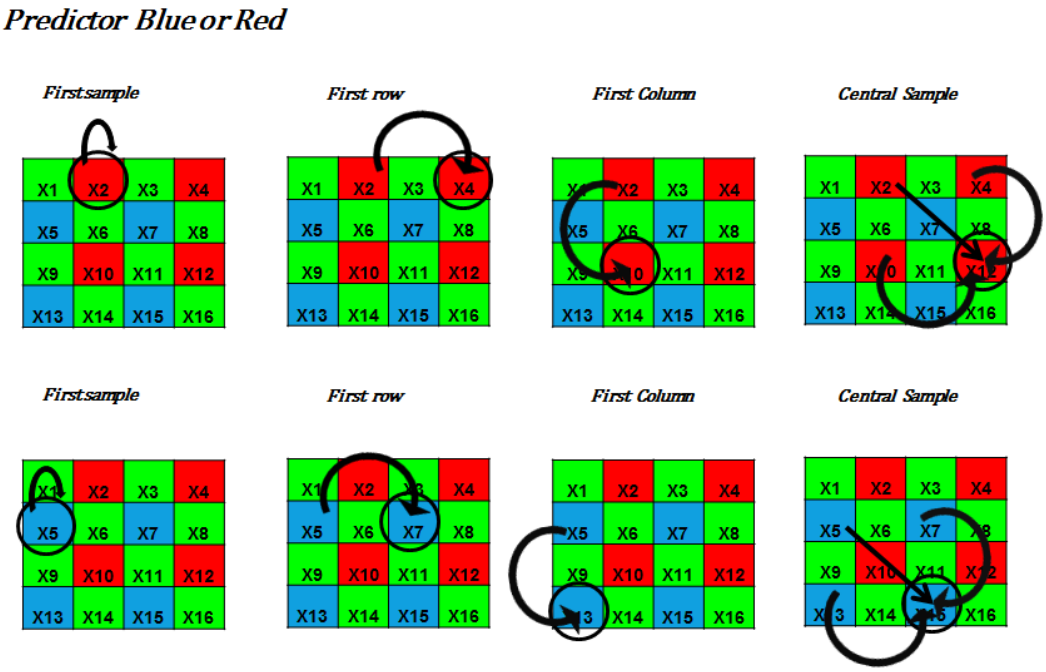


Figure 74- Predictor 3 Real indexation Algorithm Blue-Red Bands

Concerning the green band, to generate the estimate central sample we must follow the pattern illustrated at figure 75:

With $X_C = X_{x,y}$

$$X_N = X_{x+1,y-1}, \qquad X_W = X_{x-2,y}, \qquad X_{NW} = X_{x-1,y-1} \tag{26}$$

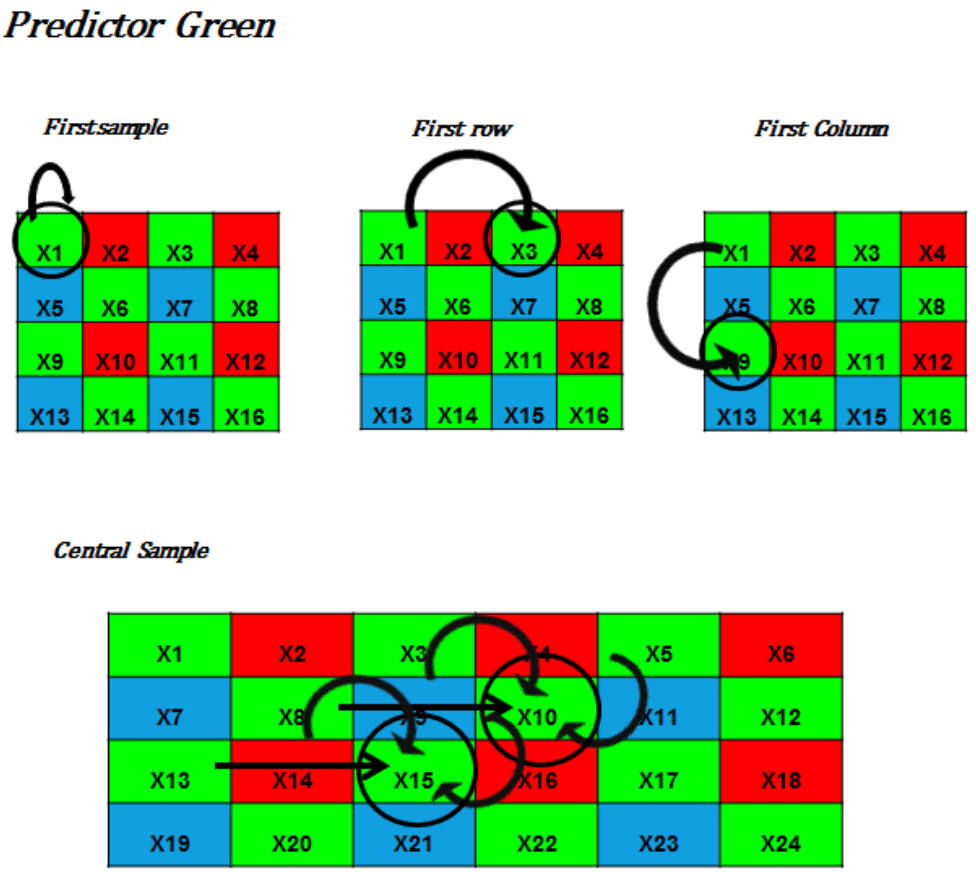


Figure 75- Predictor 3 Real indexation Algorithm Green Band

We must take care in case of being the last column because of $X_N = 0$. It does not exist for that sample.

That predictor allows us the possibility of using the spectral redundancy. The method is based on use the green residuals to re-estimate the blue and red residuals. In fact, when we generate the green residuals, we will have zones where the prediction is quite difficult because of there is energy at the image or there is noise. That information will be reflected inside the green residuals. To model that we only need to carry out the differences between green residuals and the blue and red residuals. Those new residuals will be codified by the entropy encoder. Figure 76 illustrated how to manage this new method.

Concerning the inverse prediction, we should underline that for reconstructing the green band, we must take care if we are in an odd row or in an even row. If we define the first row like $j = 0$.

With $X_C = X_{x,y}$

$$if\ j\%2 == 1 \ \rightarrow \ X_N = X_{x+1,y-1}, \qquad X_W = X_{x-1,y}, \qquad X_{NW} = X_{x,y-1} \tag{27}$$

$$if\ j\%2 == 0 \ \rightarrow \ X_N = X_{x,y-1}, \qquad X_W = X_{x-2,y}, \qquad X_{NW} = X_{x-1,y-1} \tag{28}$$

Figure 77 shows graphically the concept explained before.

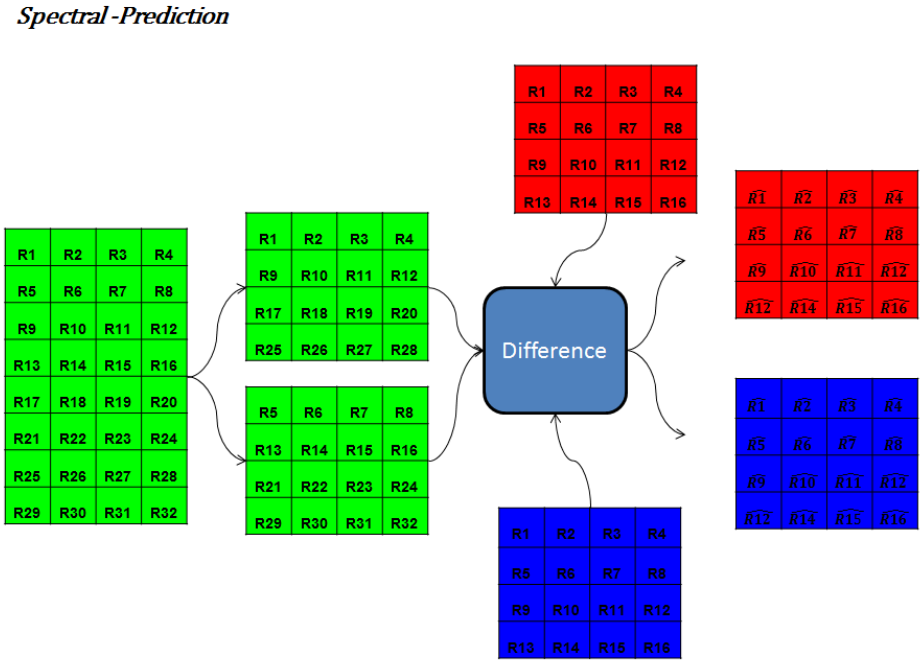


Figure 76- Exploit Spectral Redundancy

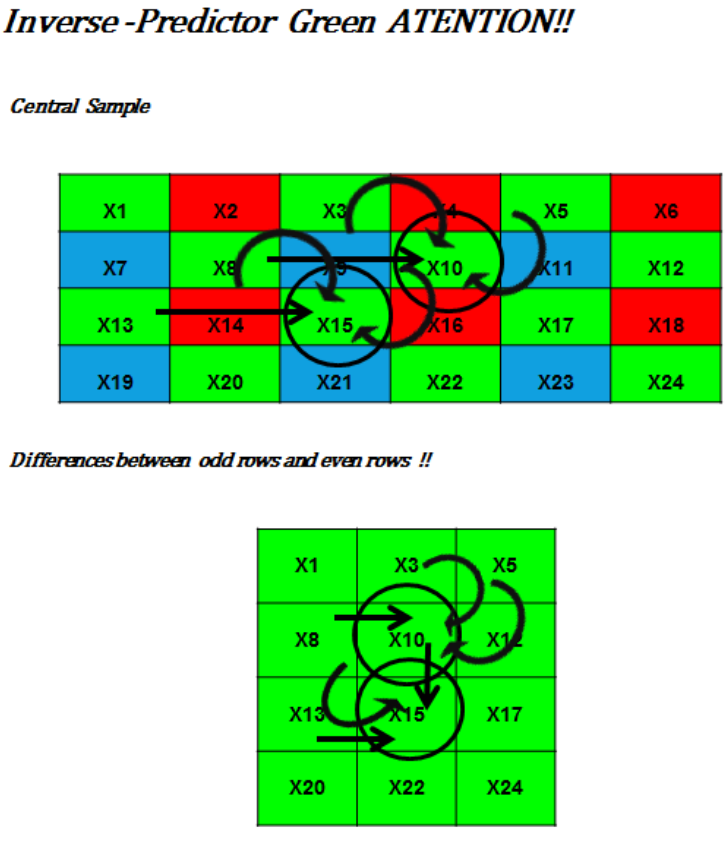


Figure 77- Inverse Predictor 3 Green Band

8.4.MAPPING

With the goal of reducing the number of operation, we are going to introduce a simpler mapping operation. The positive samples will be transformed in even numbers. Negative samples will be seemed as odd numbers. We will use the same the next operations to achieve the corresponding numbers.

- Even numbers : $x \ll 1$
- Odd number $\left((X\ xor\ 0xFFFF) \ll 1\right) + 1$

As it has been illustrated; even number does not need any operation as we only need a shifter. Concerning odd numbers we only use a sum and a binary operation, therefore the number of operation in both cases are optimized.

9. DEMOSAICING ALGORITHM

Summary

Demosaicing [7] [8] [9] is the process through which three fully populated color planes are created from the CFA data. Several algorithms exist for this purpose, ranging from simple linear interpolators to high-end nonlinear interpolators that exploit as much spatial and spectral information as possible. The goal of this section was to implement a demosaicing algorithm to be able to visualize the reconstructed image. Therefore, we are going to implement two of the most basic methods, both based in a bilinear interpolation.

9.1. INTRODUCTION

Estimated colors have less fidelity to color stimuli from the observed scene than those provided by a three-CCD camera. Improving the quality of color images acquired by mono-CCD cameras is still a highly relevant topic, investigated by researchers. However, as it is not the goal of the project, we do not focus on finding the highest performance method as possible, but only we use the simplest one in order to check results.

According to the camera type we can define two models of data acquisition as we illustrate at Figure 78. The second model is based on CFA Filter acquisition; therefore demosaicing block is compulsory to estimate the RGB color image.

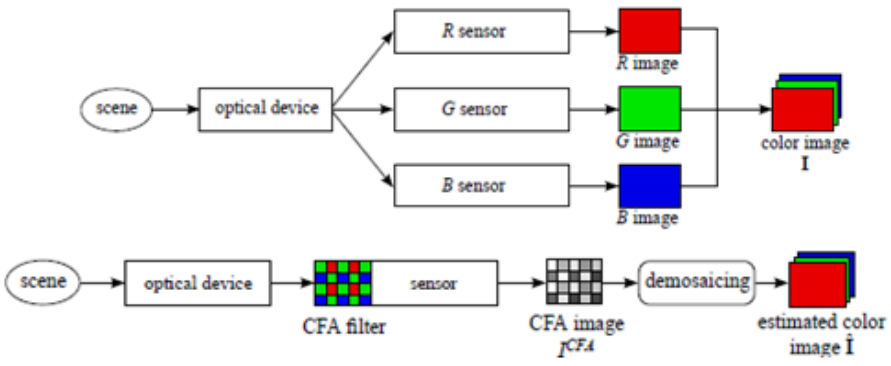


Figure 78- Models Data Acquisition

9.2.DEMOSAICING METHOD 1

The simplest demosaicing method, which is called bilinear interpolation, consists on processing each component plane separately and finds the missing levels by applying linear interpolation on the available ones, in both main directions of the image plane.

The following diagram block, which is illustrated at Figure 79, shows step by step how to develop the bilinear interpolation. Each band will create with information from the same band.

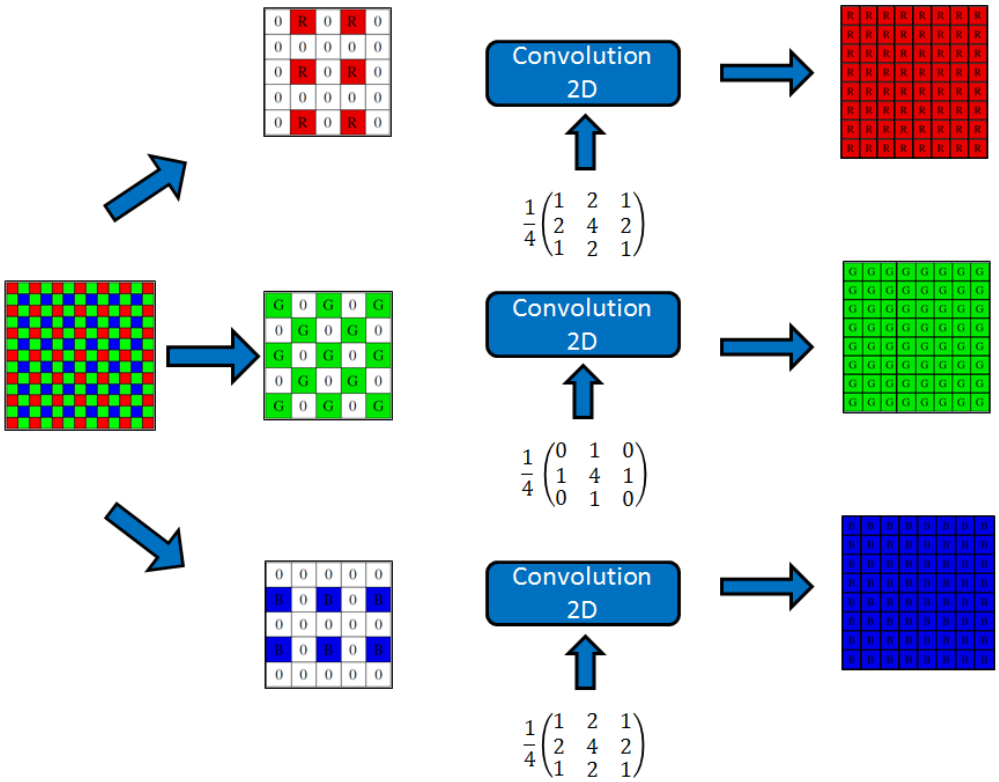


Figure 79- Demosaicing Method 1

After estimating each separate color image, we will write the result in a binary file using unsigned short integer to represent each pixel. The file will be written in BSQ format; specifically band sort will be R, G and B respectively. In order to check the RGB images, we will create a header to characterize image properties. Each header will be defined by image size, image name, number of bands, band names….

9.3.DEMOSAICING METHOD 2

As it is known, green color component has correlation with red and blue color components. The idea of the second method is used the green pixels to improve the estimation of the red and blue components.

First step is to obtain the green picture. To do that, we apply the same interpolation as the first method. Figure 80 shows us that method.

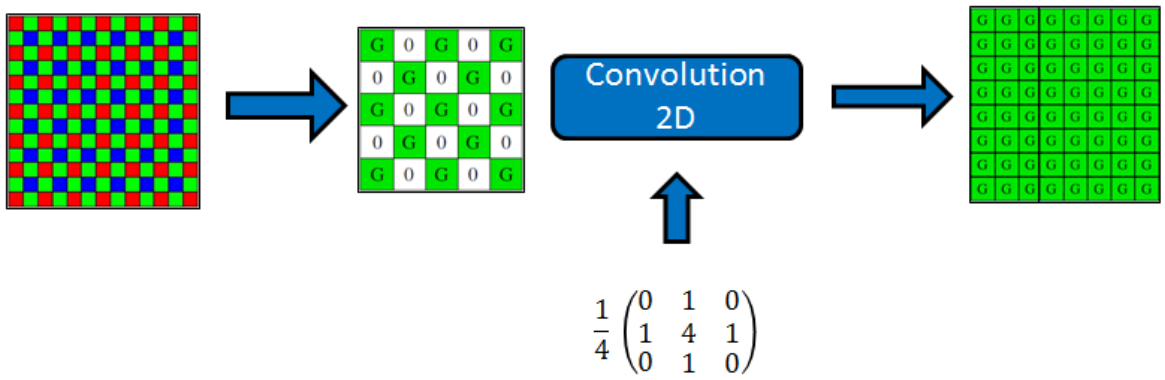


Figure 80- Demosaicing Method 2 Green Band

To estimate the blue and red bands, we use the green information. To get the new bands first we take the equivalent pixels of the red and blue bands, but of the green band. We applied the bilinear interpolation over the differences. After that we sum the obtained values with the green image. Figure 81 shows the process.

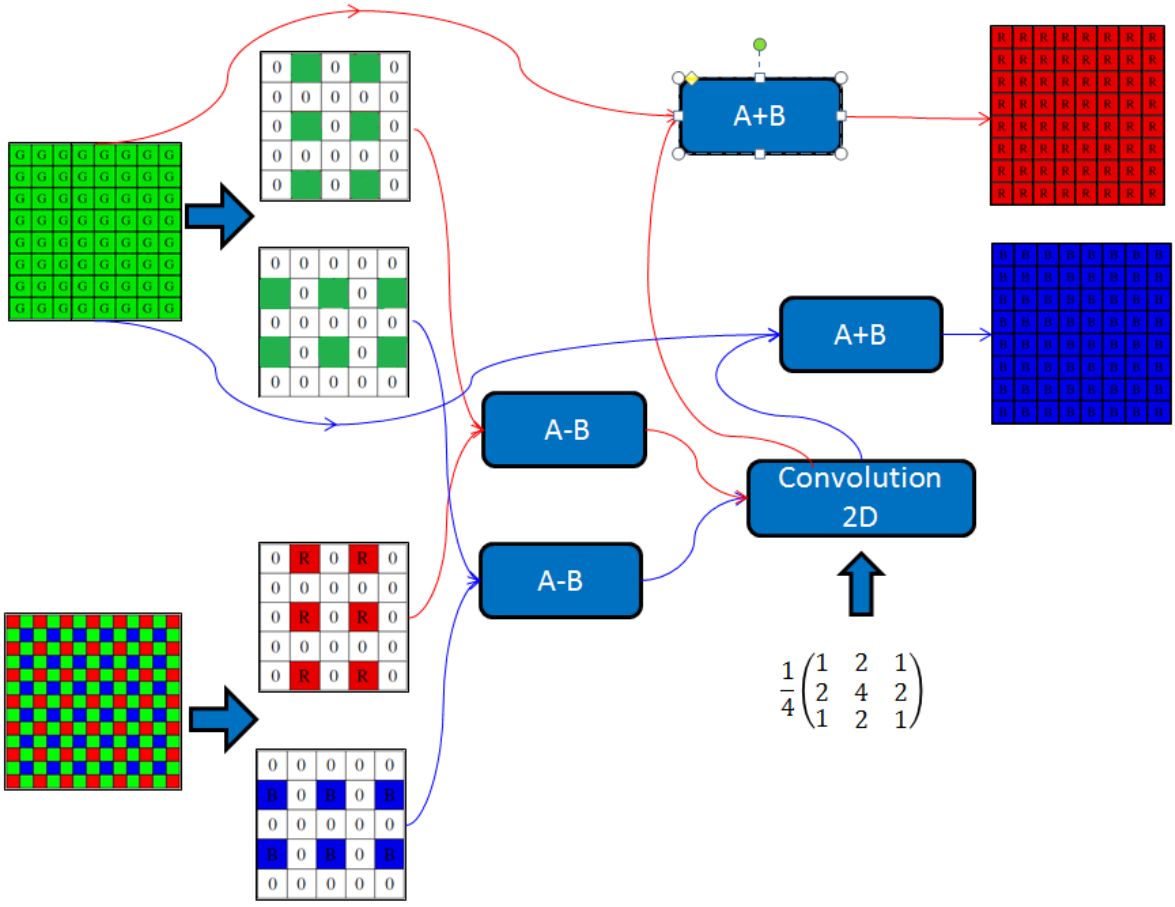


Figure 81- Demosaicing Method 2 Blue-Red Band

Part 5: Results

1.INTRODUCTION

Summary

Compression algorithms have been developed to apply over image sequences, which will be acquired, as satellite's mission. These algorithms must have some properties concerning compression ratio and complexity. At this point, we must evaluate both performances to decide the algorithm which will be implemented upon Ninano board.

1.1.IMAGES SEQUENCES

We are faced with carrying out a performance analysis upon satellite images, but without any image test. As a result, we should study the performance of the algorithms over a huge amount of images to cover all the possible results.

Selected images are illustrated in Annex 1.

Moreover, we should underline the difficulty of obtaining images with similar characteristics as those that we will acquire on board. That is, images acquired by a CFA sensor or an infrared sensor, with 12 bits per pixel and in raw format. Image treatment has been explained at Part 3.

Before starting with the performance analysis of each algorithm, I would like to introduce which tool has been used to visualize the images after decoding

1.1.1.ENVI

The Environment for Visualizing Images (ENVI) is the leading software package for analyzing all types of remotely sensed data. ENVI is a full featured, menu-driven general purpose image processing system, providing state-of-the art tools for panchromatic, multispectral, hyperspectral, advanced Synthetic Aperture Radar (SAR) imagery, as well as virtually any other type of raster image data.

ENVI can be used to perform all of the commonly used image analysis techniques, including multispectral classification, various types of spatial filtering, image registration, principal components transformations, band ratios, and images statistics. ENVI also has a unique suite of advanced spectral analysis tools designed specifically for working with hyperspectral data (although many are also appropriate for multispectral analysis) and a complete set of tools for working with radar data (both single band and fully polarimetric SAR_. Furthermore, ENVI provides full access to the programming language in which it was written, the Interactive Data Language (IDL), a powerful, yet easy to use fourth generation language whose programs can easily be incorporated into ENVI.

Concerning our project; ENVI [24] allow us to load and represent raw format images. As Raw format only supply image pixels, we have to introduce some additional information within the software to be able to represent the data. Figure 82 is an example of how introduce the metadata information. We have other possible choices to load directly the data, which consists on creating a header file. A header file will describe the size image, number of bands, data type…. Once we have already loaded the raw data, ENVI can represent the data as it is illustrated at Figure 83. Three picture display windows appear. These are linked in a special way and act as a single view of the image. The biggest one, it is the main window, which it is a small box from

the scroll window. Moreover inside the main window, we have a zoom box, which represents the zoom window

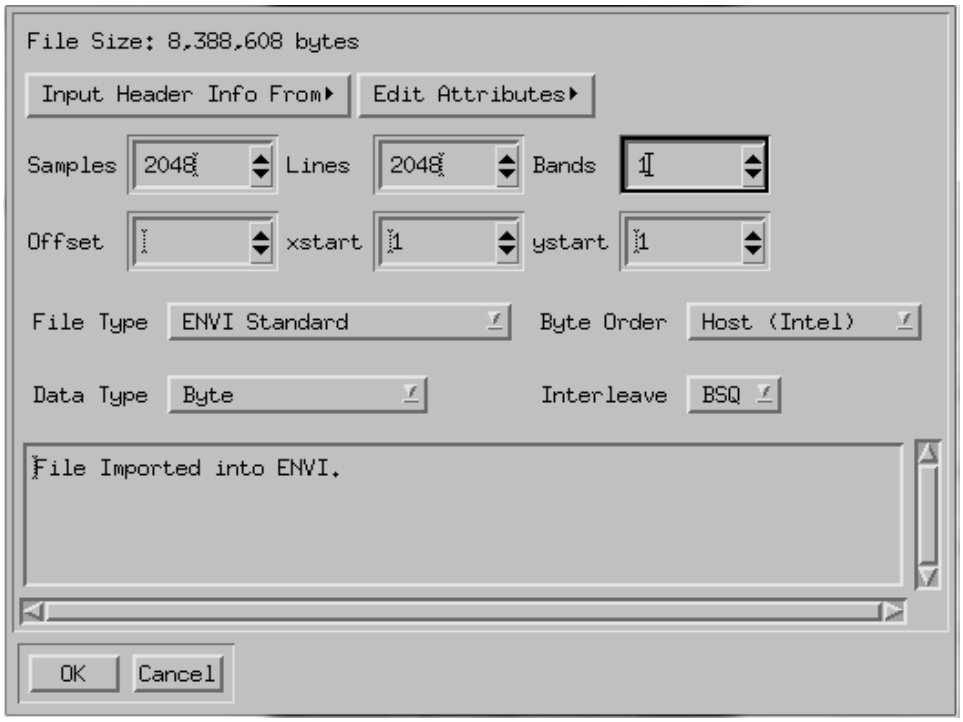


Figure 82- ENVI Header Information Dialog

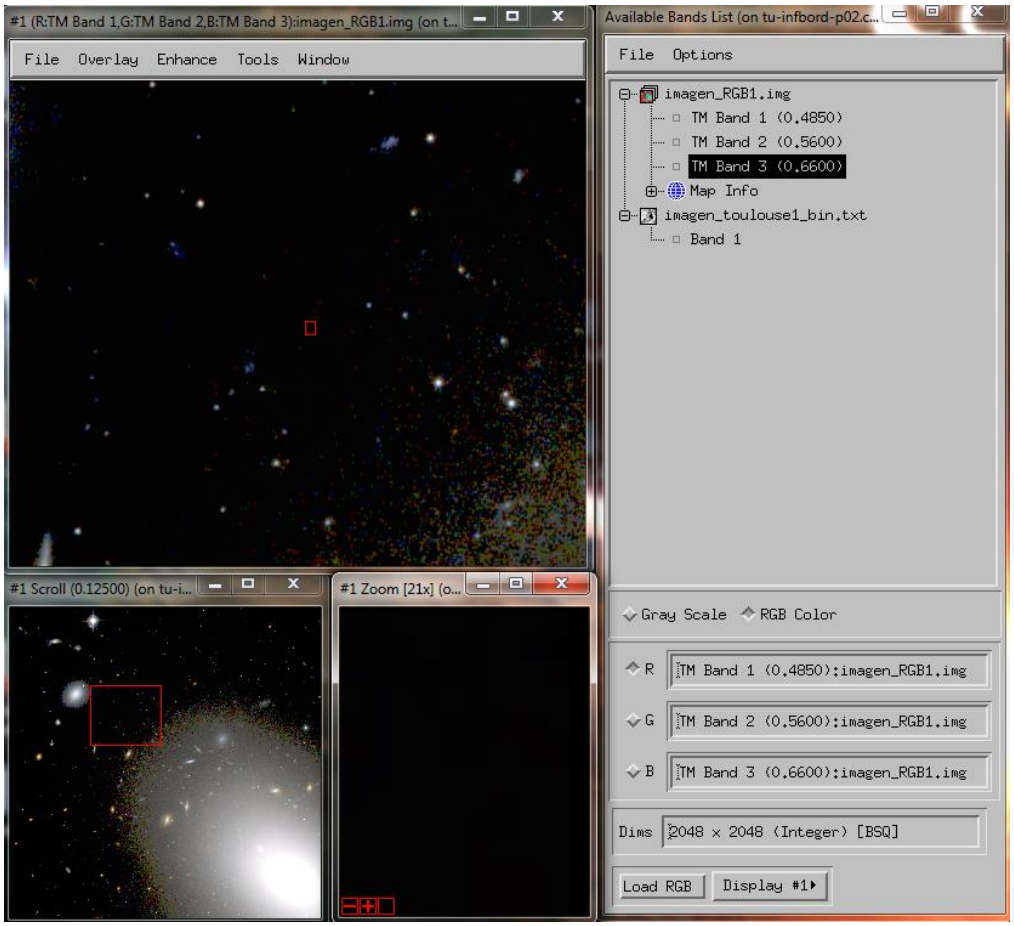


Figure 83- ENVI Display Group

2.COMPRESSION RATE AND COMPLEXITY RESULTS

Summary

For each algorithm and each image will be calculated the compression ratio. Of course, depending on the image, the number of bits per pixel or the noise level, the performances will vary. Moreover, most of the algorithms are based in the CCSDS-123 standard, so depending on the image characteristics, Predictor and Encoder properties should be changed to obtain the best performance as possible. Image repository can be found at Annex 1.

At this first stage, all the algorithms have been tested with an Inter® Core ™ I5-2400 CPU @ 3.10 GHz with 4 GB of Ram Memory.

2.1.IMAGE COMPRESSION ALGORITHMS

Concerning the performances of each algorithm, we have measured the compression rate and the computation time of each algorithm. As each algorithm are based on the CCSDS-123 Standard, which has many inputs parameters, we are not showing all the possible values, but only the best performance in each case.

Table 2 shows us the rate value for each algorithm, in the other hand Table 3 shows the computation time also for each algorithm.

2.1.1. ALGORITHM BAYER MATRIX WITH CCSDS 123

The results of this algorithm correspond with the first six columns of table 2 and table 3 and the algorithm which was explained at section 2 Part 4. The first subset of columns corresponds with a sample adaptive encoder. The second subset uses a block adaptive encoder. Moreover, in all the cases we are using the full predictor mode with the neighbor-oriented prediction. For each subset we test 3 possibilities, the first one corresponds with using two band, each band will be generated using the first pattern. The second choice will use also two bands, but now we will use the second pattern. The last case consists on considering the image as a unique band (use for infrared images or with images with poor color disparity).

After checking the results, we noticed that the block-encoder not only has better performances concerning the compression rate, but also in computing time. As results, from here we will consider the block-encoder to check the following algorithms. As a general comment, we must underline that the sample adaptive encoder has better performance than the block adaptive encoder for really noisy environments.

Concerning the pattern modes, we notice that it exists a correlation between the color disparity and the mode. For the images of Toulouse, where the color intensity is high, patterns 1 and 2 have better performance than not using any pattern and considering the image as a unique image. However, with the Space images, as we are compressing images with low color component, not use a pattern usually has better performance.

To sum up, we are going to use the Block adaptive encoder in all the cases and depending on if we have images of high color component patterns 1 or 2 will be used, otherwise we will decide not to use a pattern.

2.1.2. INTERPOLATED ALGORITHM WITH CCSDS 123

The columns 7-9 of the tables 2 and 3 show the compression rate and the time complexity for this algorithm, whose explanation can be found at section 3 Part 4. All the results at these tables are for the “pattern 1”, to check the results for “pattern 2” or for “not pattern” check the annex 4.

At it is illustrated, we have three columns, each one corresponds with not allowing any loss, allowing small losses for green samples (a different between 0-2 values) and allowing small losses for any sample. In case of allowing losses, these losses will be lower than the known noise level at any pixel.

Of course the fact of having loss involve that the reconstructed image won’t be exactly the same as the real image. We can define the error as the difference between the real image and the reconstruct image.

$$Error_{image} = Real_{image} - Reconstructed_{image} \tag{29}$$

In case of allowing small losses for green samples for a generic image we could generate an image as Figure 84. Blue pixels correspond with a non-error pixel; green pixels correspond with a pixel with a difference of 1 value and red pixels with pixels with differences of 2.

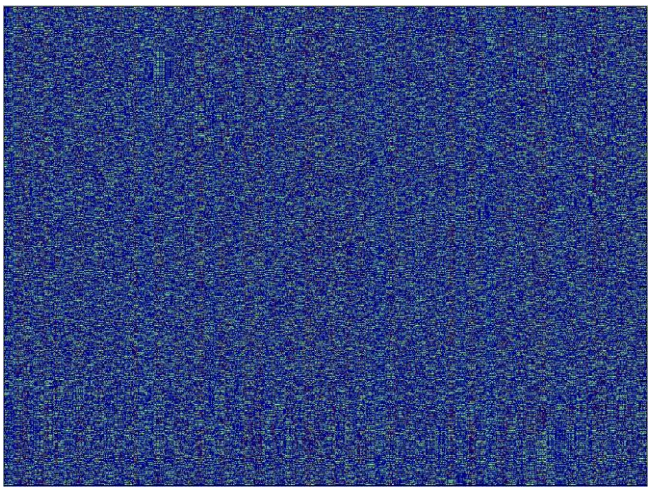


Figure 84- Error Image G-Sample

Figure 85 illustrates the histogram of the error image, as we can see, almost 60 % of the pixel are correct.

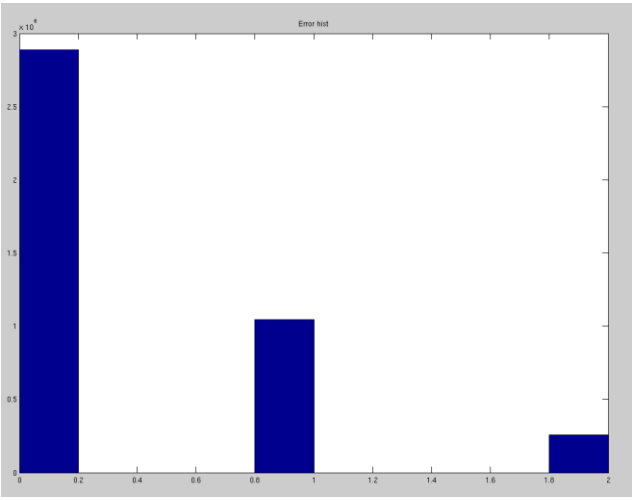


Figure 85 – Error Histogram G-samples

In case of allowing errors in the entire image the error Image will seem like the figure 86.

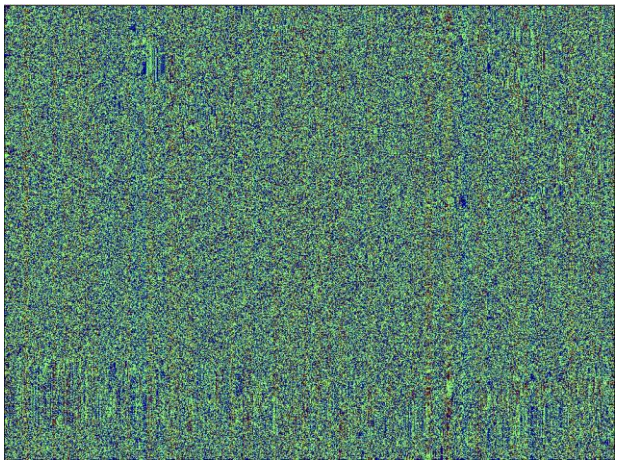


Figure 86- Error Image Entire Image

The new histogram is illustrated at figure 87. We notice an increment of pixels with errors.

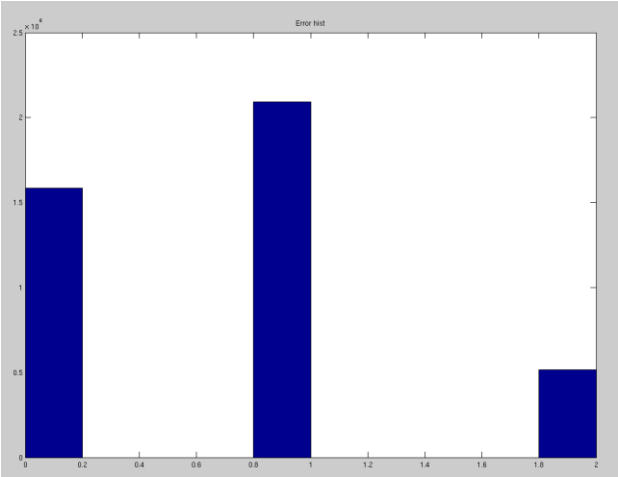


Figure 87 – Error Histogram Entire Image

Table 3 shows a small increment concerning the time complexity that is due to the interpolation step. Finally, as having any possible error was not allowed, we have excluded this possible algorithm.

2.1.3.CCSDS-122 ALGORITHM

As the ccsds-122 standard has both possibilities, loss and lossless, the goal was to check if this standard can improve the performance of the ccsds-123 algorithm for lossless compression systems. To check it, we were supplied with a C implementation, where we must introduce the compression rate. In case of that rate could be achieved by a lossless method, the algorithm will be lossless. However, in case of that rate couldn't be achieved, the algorithm will achieve the rate switching to a lossy compression encoding.

To check the performance, we had to introduce the compression rate, which was achieved by the CCSDS-123 algorithm. After that, we checked if the decompressed data has any lost. In case of having a lossless sequence, we can consider that the algorithm had at least same performance as CCSDS-123. Unluckily, after check the set of images, performances were always worse than the results obtained by using the CCSDS-123 Standard. As results we don't include any value from this algorithm inside the results tables.

2.1.4.SEGMENTED-FINAL ALGORITHM

Compression rate and time computing performances are illustrated at tables 2 and 3. The algorithm is explained at section 7 Part 4

Considerations:

- Depending on the size of the block, the algorithm will be able to converge, predict the signal and reduce the compression rate.
- Regardless of whether the algorithm has converged, the noise will be unpredictable, as results the compression rate will be limited by the noise of the image.
- Each block could have different level of compression rate because each one has different level of signal and noise.

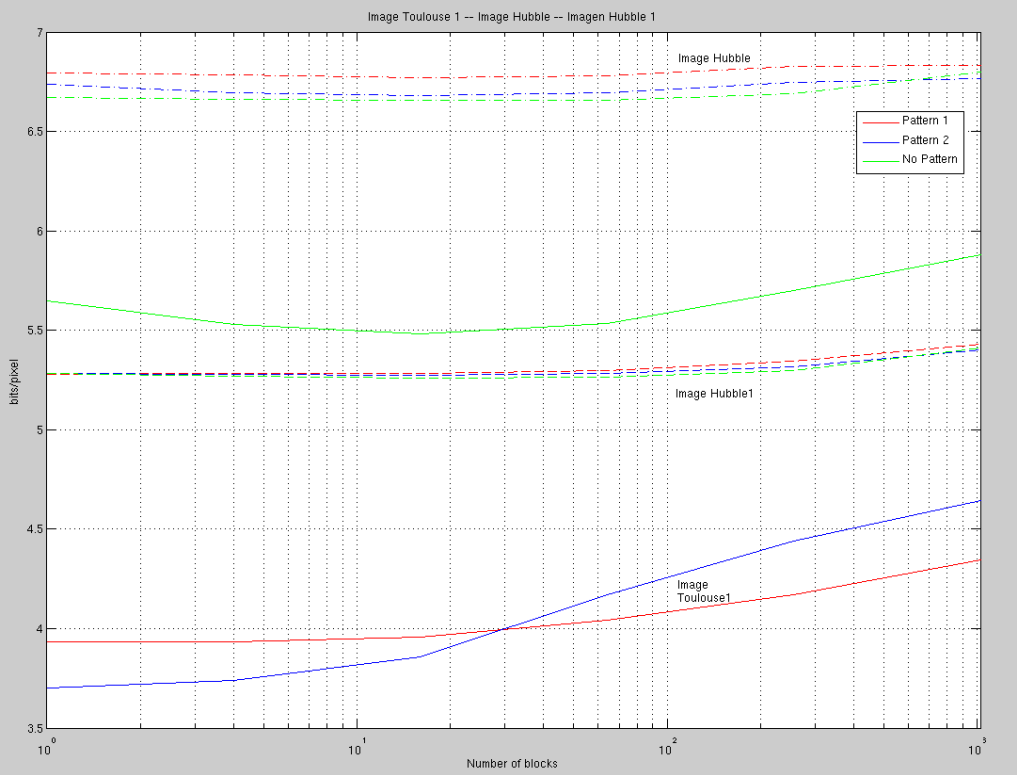


Figure 88- Bits/Pixel Segmentation Effect

Figure 88 illustrates several phenomena:

- Concerning the Image Toulouse 1, as we have not increased the noise, we notice the relationship between the increase of the bits/pixel and the number of blocks. Remember that increasing the number of blocks involves reducing the size of each block and as a result to have the possibility that the algorithm cannot converge.
- Image Hubble and Image Hubble 1 compression rate almost are constant, irrespective of reducing the size of the block. This is because the images have a high noise level; therefore, compression ratio is limited by the noise.
- In all the cases, there is a size block where the performances will be stabilized.

To sum up, good compression ratio performance shall not be achieved in case of:

- Noisy acquired images.
- Images with high signal
- Small size block as a result of segmentation.

A good way to estimate the theoretical limit of compression can be illustrated by the Entropy, especially by the conditional Entropy. At Annex 2, more information can be found.

2.1.5.EFFICIENT-SEGMENTED-FINAL-ALGORITHM

Last algorithm results are shown at the last columns of the tables 2 and 3. The explanation of the algorithm ca be found at section 8 Part 4. If we carry out a comparison between this algorithm and the "segmented final algorithm", we can notice that in most cases, the current algorithm has worse performances. In fact, it is completely normal, as the predictor does not use weight values to weight up the coefficients of the prediction. In the other hand, as images have noise, which we can consider as an unpredictable random variable not possible to predict, and have the effect of the segmentation, not to use weight coefficients does not deteriorate deeply the compression rate performances.

Moreover, as it is shown at table 3, time complexity performances are much better than any other algorithm shown before. The improvement of the performance comes from the predictor. The new predictor has been designed as a low-cost predictor, where the number of memory access and operations are minimized.

Concerning the effect of the segmentation, figure 89 illustrates the small effect of the segmentation. In fact, as we do not have adaptive weight coefficients (the algorithm does not need to converge), the performance will only affect at the first raw and first column per segment and won't affect all the pixels.

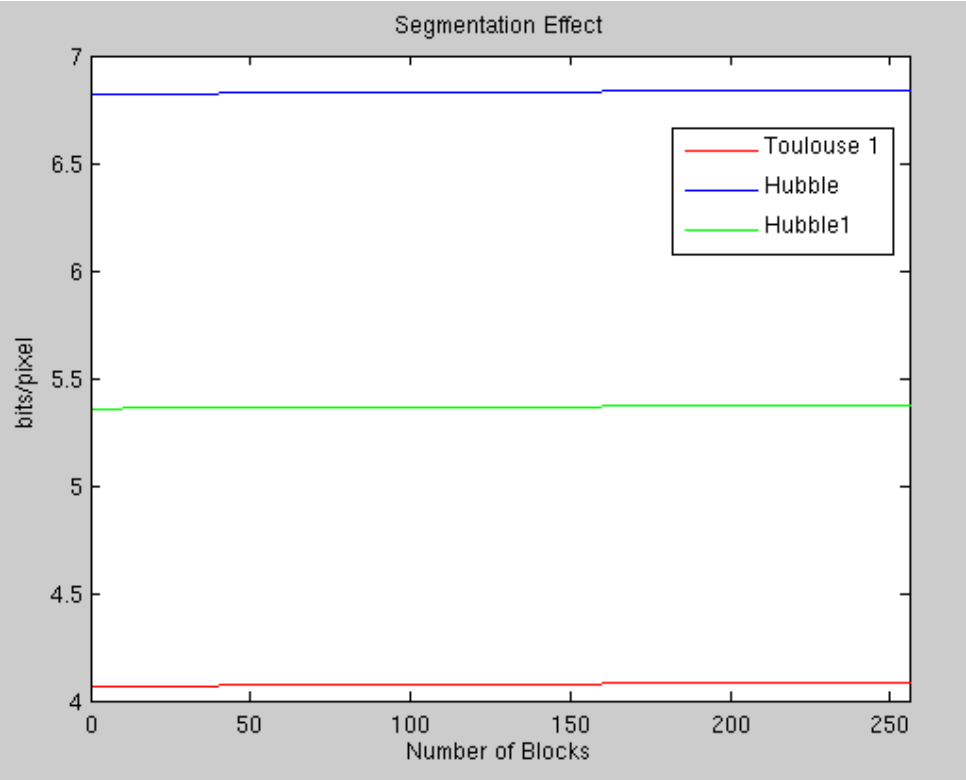


Figure 89- Bits/Pixel Segmentation Effect

All the results shown at tables 2 and 3 can be found at Annex 4.

Table 2- Bits/Pixels Image Compression Algorithms

Bits/Pixels	Images	Algorithms															
		CCSDS-123- Sample Encoder			CCSDS-123- Block Encoder			Interpolation CCSDS-123- Block Encoder			Segmented CCSDS-123- Block Encoder			Efficient Algorithm- Block Encoder			
		Pattern 1	Pattern 2	No Pattern	Pattern 1	Pattern 2	No Pattern	Green NLost	Green Lost	All Lost	Pattern 1	Pattern 2	No Pattern	Mode 1	Mode 1 Spec	Mode 2	Mode 3
8 bits	Toulouse 1	4.078476	3.792938	5.620712	3.935501	3.699142	5.650192	4.124756	3.661041	3.569077	4.040268	4.171082	5.535645	4.072174	4.014709	5.402557	5.779388
	Toulouse 2	3.371323	3.132416	5.084976	3.318268	3.126007	5.112686	3.546875	3.046753	2.963211	3.475647	3.704636	5.201172	3.431503	3.389282	4.805649	5.376251
	Toulouse 3	3.953598	3.766998	5.725861	3.857315	3.694107	5.756714	4.087891	3.607285	3.490295	3.975494	4.133331	5.683090	4.106171	4.025955	5.317749	5.781464
12 bits/pixel +noise	Hubble	6.897705	6.816055	6.736176	6.793793	6.821472	6.671463	6.974747	6.448181	6.105423	6.781860	6.694031	6.657608	6.825058	6.956726	6.952332	6.769653
	Hubble1	5.303955	5.307007	5.297745	5.279877	5.281555	5.281677	5.440353	4.932571	4.554947	5.299469	5.283936	5.264893	5.434875	5.558762	5.463791	5.363541
	space	6.736496	6.577942	6.327164	6.553238	6.464478	6.095596	6.756424	6.256622	5.990631	6.588135	6.499557	6.103516	6.678925	6.854645	6.750198	6.241928
	milky	6.899841	6.950989	6.794510	6.761948	6.821472	6.704041	6.920212	6.420929	6.092224	6.770111	6.831573	6.676956	6.86505	6.935089	6.994675	6.809525

Table 3- Computation Time Image Compression Algorithm

Time (s)	Images	Algorithms															
		CCSDS-123- Sample Encoder			CCSDS-123- Block Encoder			Interpolation CCSDS-123- Block Encoder			Segmented CCSDS-123- Block Encoder			Eficient Algorithm- Block Encoder			
		Pattern 1	Pattern 2	No Pattern	Pattern 1	Pattern 2	No Pattern	Green NLost	Green Lost	All Lost	Pattern 1	Pattern 2	No Pattern	Mode 1	Mode 1 Spec	Mode 2	Mode 3
8 bits	Toulouse 1	1.513000	1.498000	1.364000	1.420000	1.420000	1.207000	1.451000	1.435000	1.452000	1.478000	1.480000	1.338000	0.506000	0.529000	0.519000	0.513000
	Toulouse 2	1.482000	1.482000	1.355000	1.420000	1.420000	1.195000	1.420000	1.435000	1.447000	1.479000	1.474000	1.340000	0.500000	0.516000	0.506000	0.507000
	Toulouse 3	1.513000	1.499000	1.368000	1.419000	1.419000	1.213000	1.449000	1.436000	1.454000	1.488000	1.488000	1.343000	0.501000	0.523000	0.513000	0.515000
12 bits/pixel +noise	Hubble	1.577000	1.568000	1.427000	1.494000	1.494000	1.336000	1.529000	1.497000	1.516000	1.562000	1.558000	1.389000	0.571000	0.590000	0.580000	0.575000
	Hubble1	1.542000	1.536000	1.389000	1.467000	1.467000	1.312000	1.498000	1.485000	1.492000	1.524000	1.516000	1.369000	0.552000	0.561000	0.551000	0.540000
	space	1.545000	1.539000	1.394000	1.473000	1.469000	1.32100	1.498000	1.491000	1.502000	1.545000	1.524000	1.398000	0.553000	0.571000	0.571000	0.560000
	milky	1.569000	1.561000	1.424000	1.447000	1.481000	1.336000	1.497000	1.502000	1.514000	1.548000	1.556000	1.390000	0.571000	0.594000	0.584000	0.569000

2.2.VIDEO COMPRESSION ALGORITHMS

Video compression algorithms were developed in order to improve the compression performances by exploiting the time dimension. However, as rotation movements were not considered, we decided to not consider these algorithms as a final option to be implemented.

2.2.1.VIDEO CODER ALGORITHM 1

Next section shows the results concerning the first video compression algorithm; to simulate the motion between sequences we have used convolutions and translation (in dimension, vertical and horizontal) inside the image. The results show better performance at compression ratio. However, memory resources, and possible rotation movements exclude the algorithm to be implemented as the final algorithm.

Checking table 4, first column represents the compression rate for a sequences composed of five images. Compression rate is better than any images compression algorithm; however, time computing, which can be found at columns two and three, is higher than any image compression algorithm. To develop the algorithm, we must have at least to two store images to start to compress, the number of memory access are much bigger than the image compression algorithms because of we need two frames to carry out the compression of a unique image.

2.2.2.VIDEO CODER ALGORITHM 2

The second algorithm has been developed, as a solution of the high complexity of the first video compression algorithm. In fact, instead of applying the ccsds-123 predictor for each time-residual image, we avoid the ccsds-123 predictor and apply directly the entropy encoder on the time-residuals samples.

Table 4 illustrates the results of compression rate and time complexity. As it is shown, the time complexity has been highly reduced, however as a result of this new low-cost technique, the compression rate has worsened.

As it has been shown, most complexity is often analogous with best compression rate performances; therefore, as it is a trade-off, we must check and decide which algorithm can fit as a better solution.

Table 4- Video Compression Algorithms Performances

	Images	***Algorithms***					
		Video Compression Algorithm 1			***Video Compression Algorithm 2***		
		Bits/Pixel	***Time (s)***	***Time/Image (s)***	***Bits/Pixel***	***Time (s)***	***Time/Image (s)***
8 bits	***Sequence Toulouse 1***	*2.332050*	*8.268000*	*1,6536*	*2.943024*	*5.119000*	*1,0238*
	Sequence Toulouse 2	*2.356140*	*8.268000*	*1,6536*	*2.571429*	*5.005000*	*1,001*
	Sequence Toulouse 3	*2.752649*	*8.330000*	*1,666*	*2.937631*	*5.116000*	*1,0232*

2.3.DEMOSAICING METHOD

Next section was developed as a visualizing goal, to check that the Bayer images [7] [8] [9] developed was correct. In fact, these algorithms there are not going to be implemented on-board. Therefore, the next section is focused on showing the effects of carrying out a demosaicing method upon the Bayer CFA. As it is known, Bayer CFA only has one band of information, which it is composed by red, blue and green pixels. Based on a RGB images, we will select the appropriated pixels to create the Bayer CFA. After that, we will use the demosaicing method to check the results between the real image and the estimate image.

2.3.1.METHOD 1

As it was explained at part 4, the first demosaicing method consist on a bilinear interpolation, where from the green, blue and red sample of the CFA, we are able to reconstruct the RGB image. First method Performances are studied by the followings steps. Figure 90 and 91 shows and RGB and CFA images respectively.

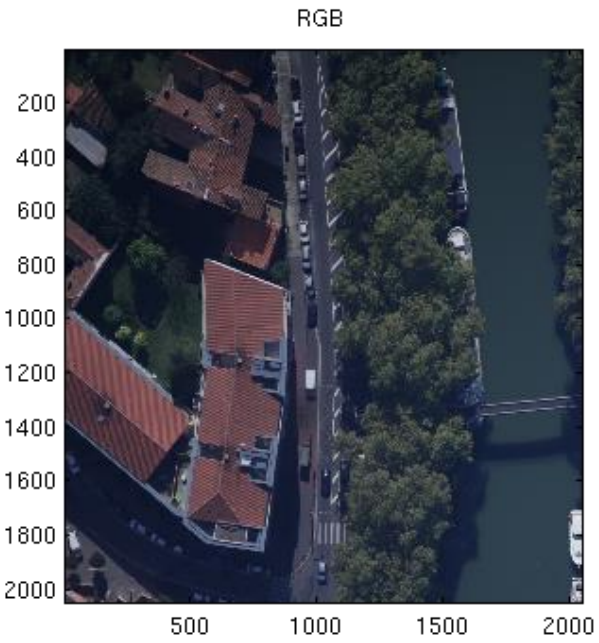


Figure 90- RGB Image

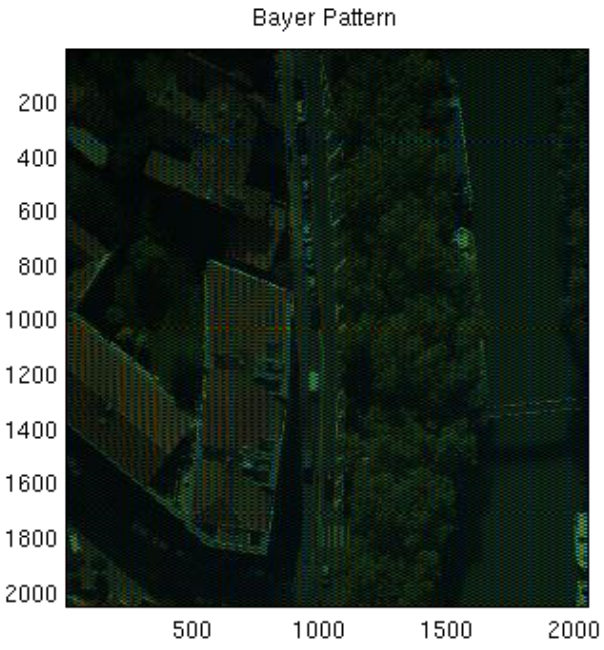


Figure 91- Bayer Image

Figure 92 shows the reconstruct RGB images after using the demosaicing technique. As it is illustrated, the images look equals, however there are not similar. Figure 93 shows one of the most common artifacts after carrying out a demosaicing technique, “the false color”. In fact, the bilinear interpolation provides good results in image areas with homogenous colors; however it creates zones of false colors in images zones with spatial high frequencies.

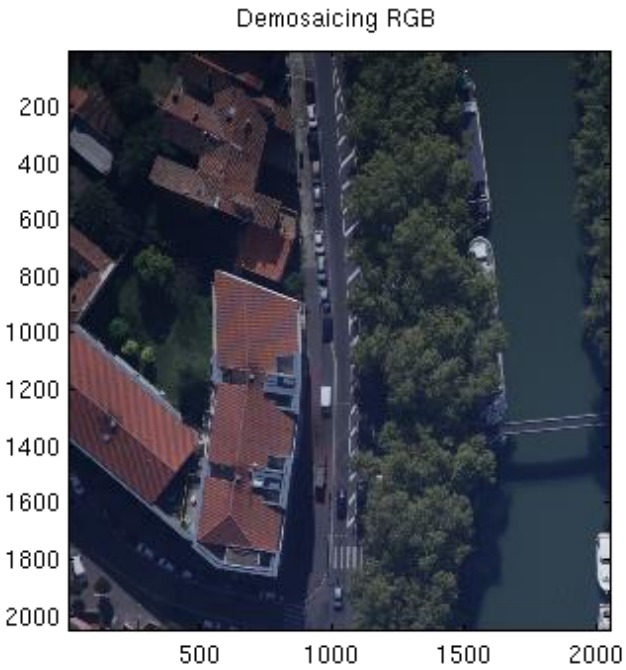


Figure 92- Demosaicing RGB Image Method 1

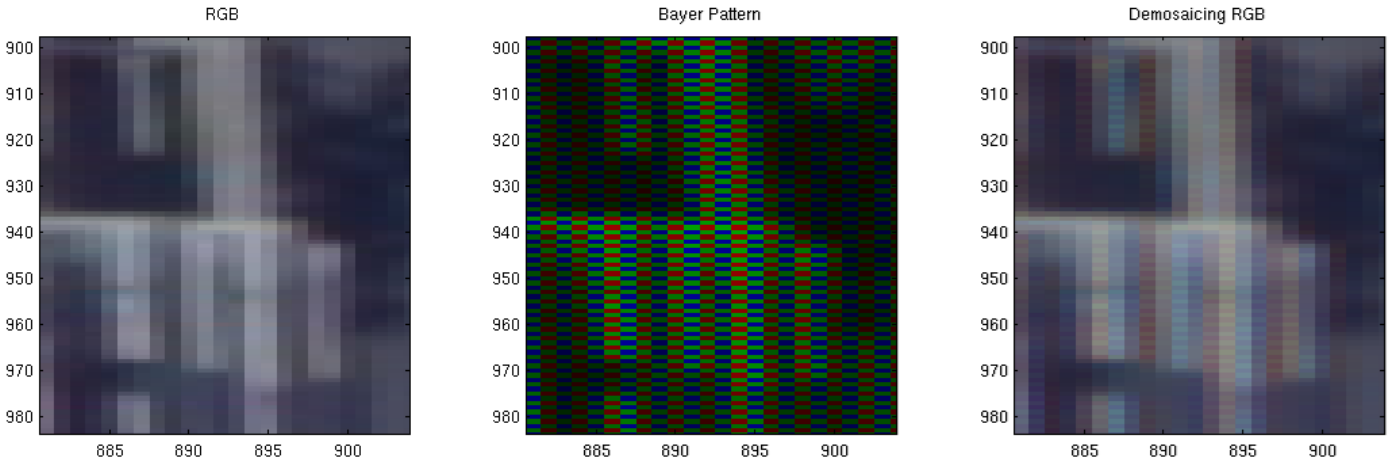


Figure 93- False Color Artifact Method 1

To check the real performances between methods, we could compare each recontructed image with the real RGB image. By substracting each color component images, we will obtain the error images for each color. Once error images have been generated, the mean square error measurement can be used as an indicative error value. Figure 94 shows the error Images for the first method of demosaicing and table 5 the mse values.

Table 5- MSE Method 1

Image	***Red***	***Green***	***Blue***
MSE	***7.1502***	***2.8855***	***9.4096***

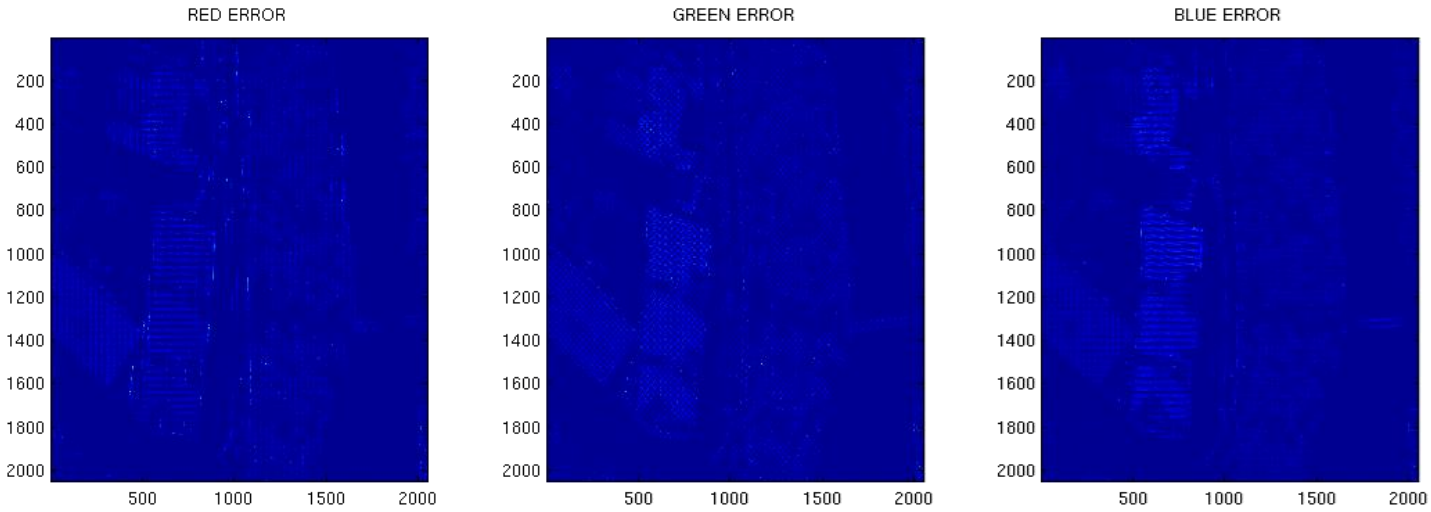


Figure 94- Error Images Method 1

2.3.2.METHOD 2

As it was explained at part 4, the second demosacing was based on a bilinear interpolation, where green, blue and red pixels were used to create the RGB image. This method differs from the first method due to green pixels are also used to reconstruct the blue and red color images. Figure 95 shows the new reconstructed image. One of the most significant improvements of this method is not only concerning the improvement of the MSE value for each error image, but also a better performance in avoiding the false color.

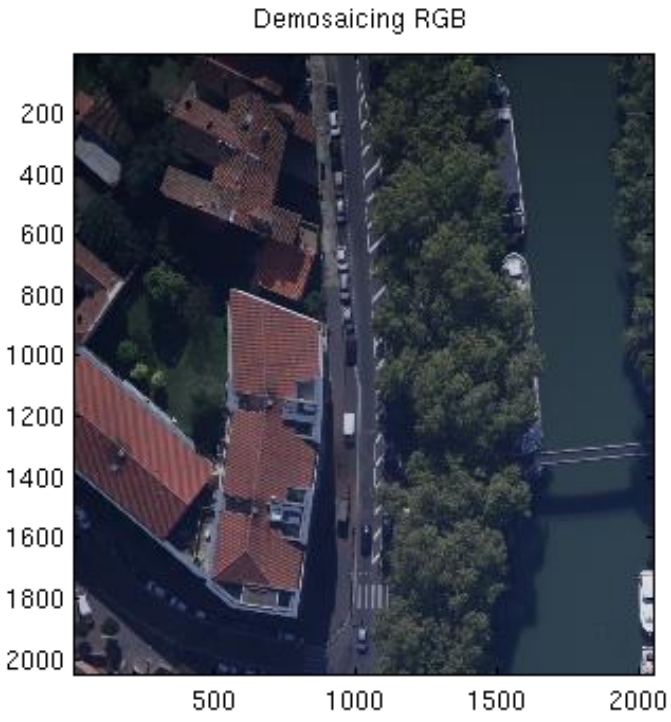


Figure 95- Demosaicing RGB Image Method 2

Figure 96 shows again the false color artifact. As it is illustrated a better performances has been achieved. Moreover error images are illustrated at image 97 and the newest measurement of MSE are shown at table 6.

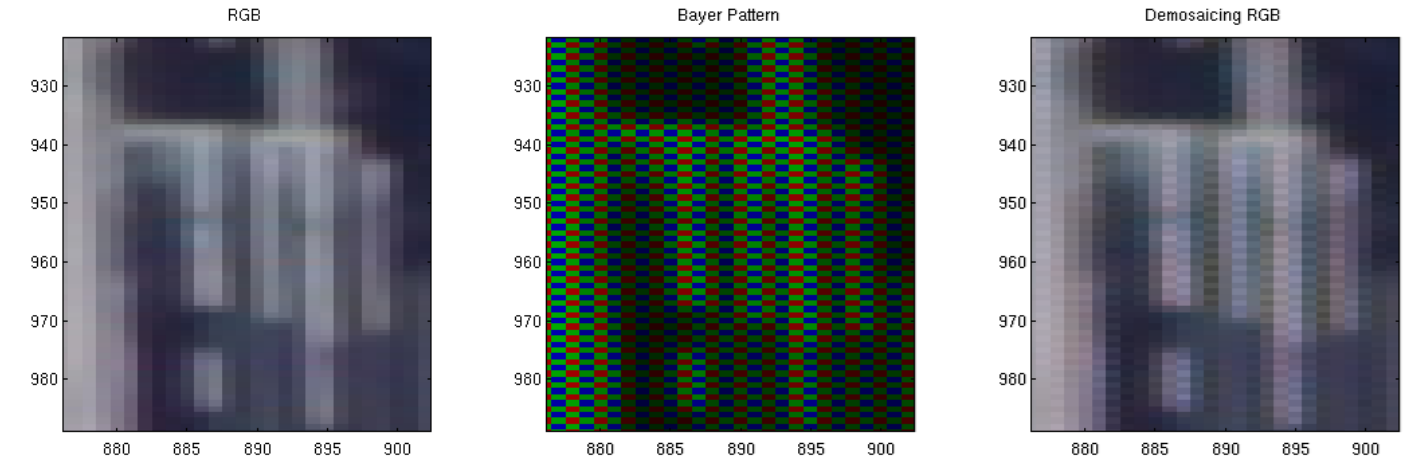


Figure 96- False Color Artifact Method 2

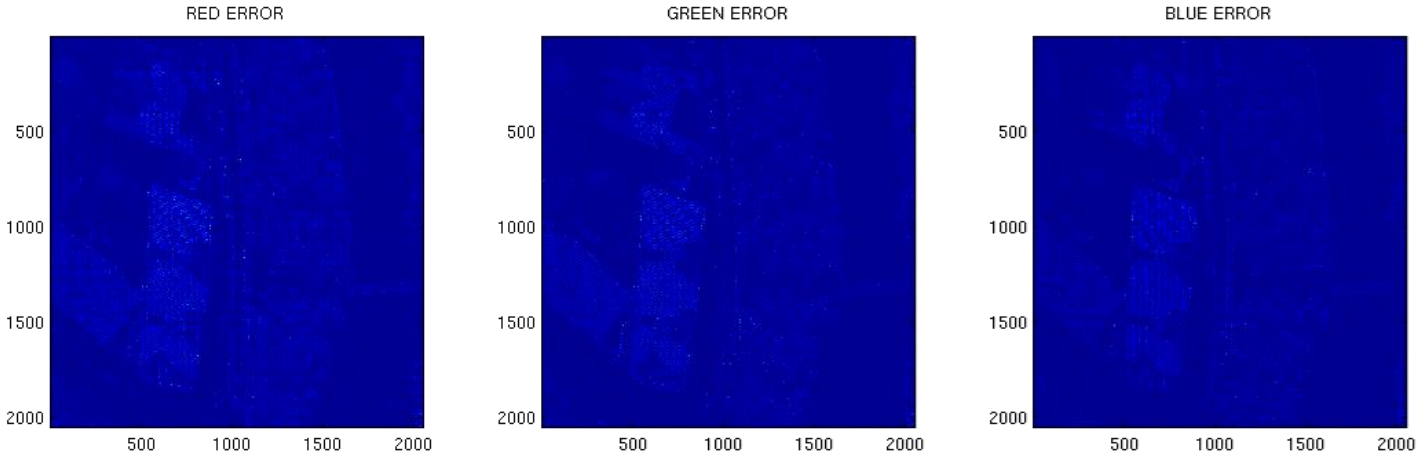


Figure 97- Error Images Method 2

Table 6- MSE Method 2

Image	*Red*	*Green*	*Blue*
MSE	*3.4331*	*2.8855*	*3.2417*

To finish this section, though the figure 98, we show the edge effect of the demosaicing technique, especially upon an image which has been segmented and has been affected by fading in the transmission process.

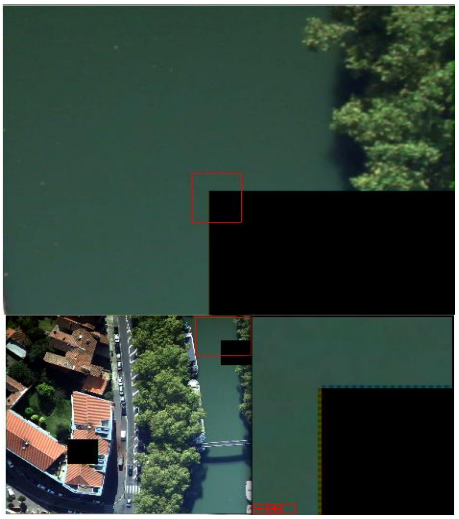


Figure 98- Segmentation Effect

Part 6: Ninano-XtratuM Implementation

1.INTRODUCTION

Summary

The goal of the project, not only consists on carrying out the research and the development of lossless compression algorithms along with their implementation in C language, but also handles the implementation into the satellite central processor board. This satellite board, whose name is Ninano, is based in System-on-Chip (SoC) technology; especially it uses a Xilinx ZYNQ® All Programmable SoC which includes a programmable logic module (FPGA) and a Dual-core ARM® Cortex™-A9 processor with NEON™ DSP/FPU Engines. Moreover; at this project it will be used XTratuM, which is a bare metal hypervisor used for real time embedded systems and which is available for architectures x86, LEON 2 and processor ARM Cortex-R4F. Memory resources will be taken into account to develop the algorithm as well as error notifications. Compression rate and computational complexity will be analyzed to conclude if the performances are good enough. To check the algorithm on-board; firstly we will use the Xilinx SDK and after that we will implement it on XtratuM.

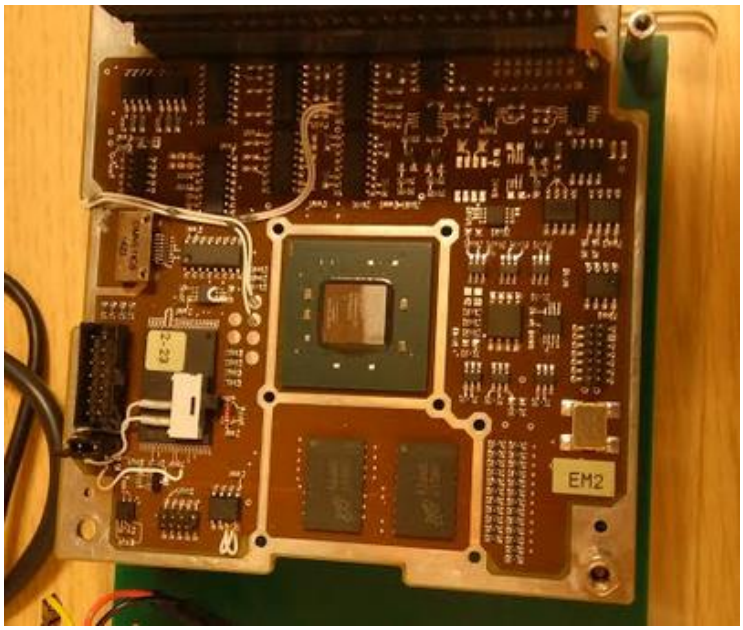


Figure 99- Ninano Board

1.1.GOAL

Good compression rate and low complexity are mission requirements; as a result a compression rate of almost 2 and a computing time lower than 15 seconds are demanded for each image.

1.2.BOARD DESCRIPTION

This section focuses on summing up the SoC technology, especially ZYNQ® technology, as well as describing the basic technical board characteristics which will be used to develop our project [18] [19] [20] [21] .

The System-on-Chip technology involve that through a unique silicon chip, to be able to implement the functionality of an entire system, rather than several independent physical chips which should be connected

to create an equivalent system. In fact, SoC solution is lower cost, enables faster and more secure data transfers between the various system elements, has higher overall system speed, lower power consumption, smaller physical size, and better reliability. System-on-Chip technology seems a really good technology, which is able to give us outstanding performance. However, there is a drawback concerning the low flexibility of the system.

Technical solutions have focused on dealing with this lack of flexibility; as a result, All Programmable SoC devices were created. ZYNQ® is an example of these technologies; in fact, it provides an outstanding platform by combining a dual-core ARM® Cortex-A9 processor with a traditional Field Programmable Gate Array (FPGA) logic fabric. In ZYNQ®, the ARM® Cortex-A9 is an application grade processor, capable of running full operating systems such as Linux, while the programmable logic is based on Xilinx 7-series FPGA architecture. The architecture is completed by industry standard AXI interfaces, which provide high bandwidth, low latency connections between the two parts of the device.

Figure 100 illustrates a basic sketch of ZYNQ® architecture.

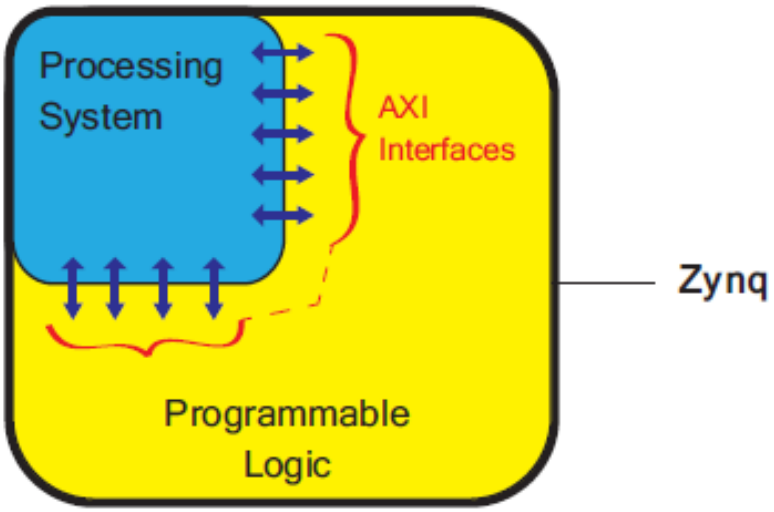


Figure 100-- A simplified model of the ZYNQ® architecture

All ZYNQ® devices have the same basic architecture, and all of them contain, as the basis of the processing system, a dual-core ARM® Cortex-A9 processor. This is a 'hard' processor. Moreover, the ZYNQ® processing system encompasses not just the ARM processor, but a set of associated processing resources forming an Application Processing Unit (APU), and further peripheral interfaces, cache memory, memory interfaces, interconnect, and clock generation circuitry as it is shown at Figure 101.

ZYNQ® architecture allows us to carry out multitude of implementation, however the goal of this section does not consist on explaining all the characteristics of ZYNQ® (for more interest check [18], [19], [20], [21]). Concerning our project, memory management is the most important matter. ZYNQ®-7000 AP devices feature a number of different types of memory and memory interfacing facilities.

Zynq-7000 AP SoCs have support for an address space of 4 GB. The memory map is provided in table 7.

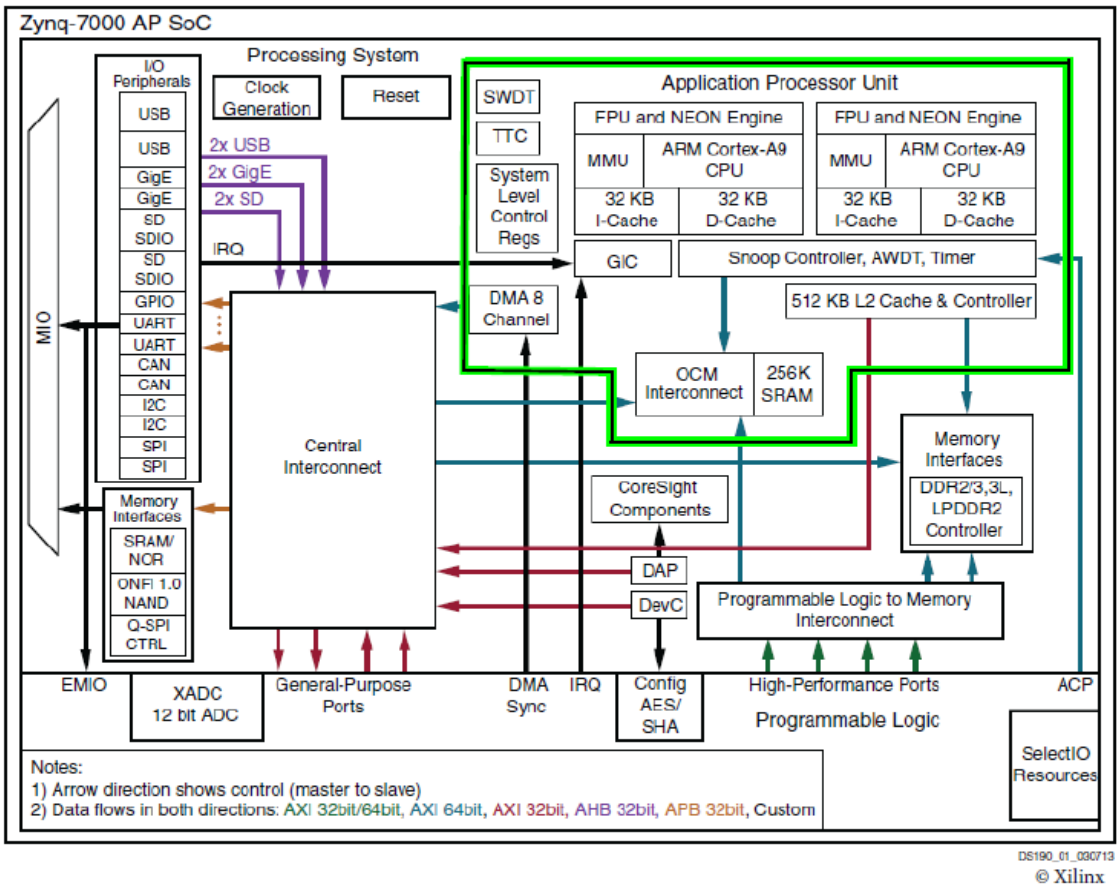


Figure 101- The Processing ZYNQ® System

Table 7- ZYNQ®-7000 SoC memory map

Start Address	Size	Description
0x0000_0000	1024 MB	DDR DRAM and OCM
0x4000_0000	1024 MB	PL AXI slave port 0
0x8000_0000	1024 MB	PL AXI slave port 1
0xE000_0000	256 MB	IOP devices
0xF000_0000	128 MB	Reserved
0xF800_0000	32 MB	Programmable registers access via AMBA APB
0xFA00_0000	32 MB	Reserved
0xFC00_0000	64 MB — 256 KB	Quad-SPI linear address base (except top 256 KB which is OCM), 64 MB reserved, only 32 MB is currently supported
0xFFFC_0000	256 KB	OCM when mapped to high address space

Concerning our implementation, everything will be saved at the first memory raw, table 7. Image and global implementation data will be stored at DRAM and memory access will be carried out by the DDR3 (The multi-protocol Double Data Rate). Code and internal variables will be stored on OCM, in fact internal variables are store at the cache to increase the speed of the algorithm and reduce the memory access.

1.3.XTRATUM HYPERVISOR

The concept of partitioned software architectures was developed to address security and safety issues. The central design criteria involve isolating modules of the system into partitions. Temporal and Spatial isolation are the key aspects in a partitioned system. Based on this approach, the Integrated Modular Avionics (IMA) is a solution allowed by the Aeronautic Industry to manage the increment of the functionalities of the software maintaining the level of efficiency.

XtratuM is a bare-metal hypervisor designed to achieve temporal and spatial partitioning for safety critical applications.

The design of a hypervisor for a real-time system must follow the following criteria:

- strong temporal isolation: fixed duty planner
- strong spatial isolation: all partitions are executed in a user processor mode
- Basic resource Virtualization: clock and timer, interrupt, memory, CPU and specialty models
- policy of real-time scheduling for partitions
- hypercalls (hypervisor system calls) deterministic
- static system definition via configuration file (XML).

Figure 102 shows the complete architecture the Xtratum; for more information check [25].

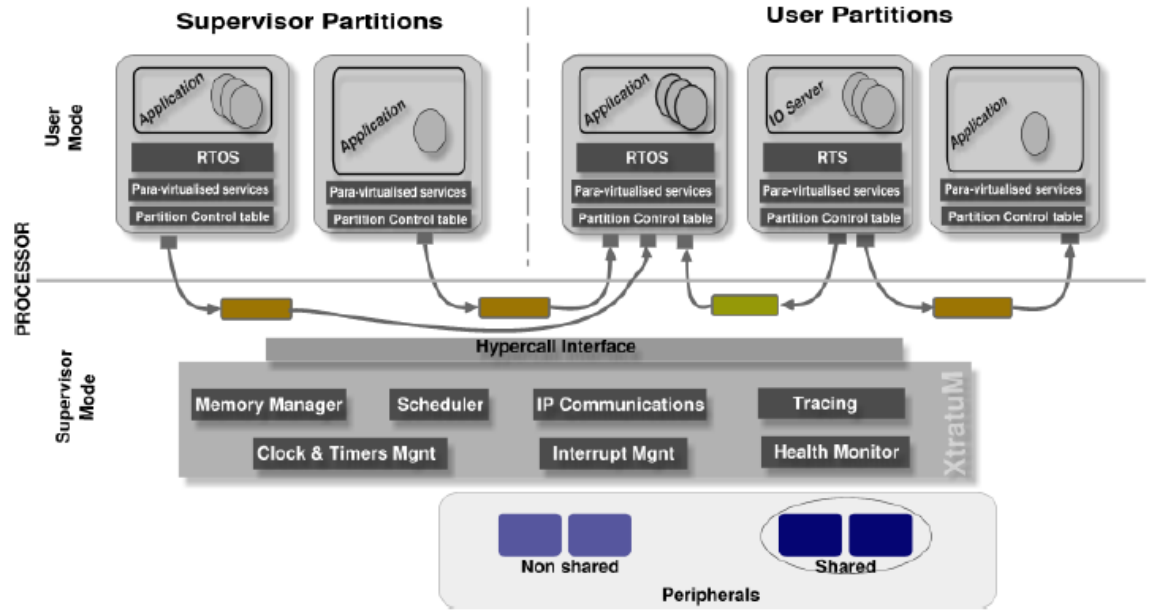


Figure 102- XtratuM architecture

1.4.XILINX SDK (SOFTWARE DEVELOPMENT KIT)

The Xilinx Software Development Kit (XSDK) is the Integrated Design Environment for creating embedded applications on any of Xilinx Zynq®. The SDK is the first application IDE to deliver true homogenous and heterogeneous multi-processor design and debug. Among its tools we should underline:

1.4.1.XILINX® SYSTEM DEBUGGER

Xilinx® System Debugger enables you to see what is happening to a program while it executes. You can set breakpoints or watchpoints to stop the processor, step through program execution, view the program variables and stack, and view the contents of the memory in the system.

Xilinx System Debugger supports debugging through SDK and Command-line interface (CLI).

SDK System Debugger uses the Xilinx hw_server as the underlying debug engine. SDK translates each user interface action into a sequence of TCF commands. It then processes the output from system Debugger to display the current state of the program being debugged. It communicates to the processor on the hardware using Xilinx hw_server. The debug workflow is described in the Figure 103.

The workflow is made up of the following components:

• Executable ELF File: To debug your application, you must use an Executable and Linkable Format (ELF) file compiled for debugging. The debug ELF file contains additional debug information for the debugger to make direct associations between the source code and the binaries generated from that original source.

• Debug Configuration: In order to launch the debug session, you must create a debug configuration in SDK. This configuration captures options required to start a debug session, including the executable name, processor target to debug, and other information.

• SDK Debug Perspective: Using the Debug perspective, you can manage the debugging or running of a program in the Workbench. You can control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables.
You can repeat the cycle of modifying the code, building the executable, and debugging the program in SDK.

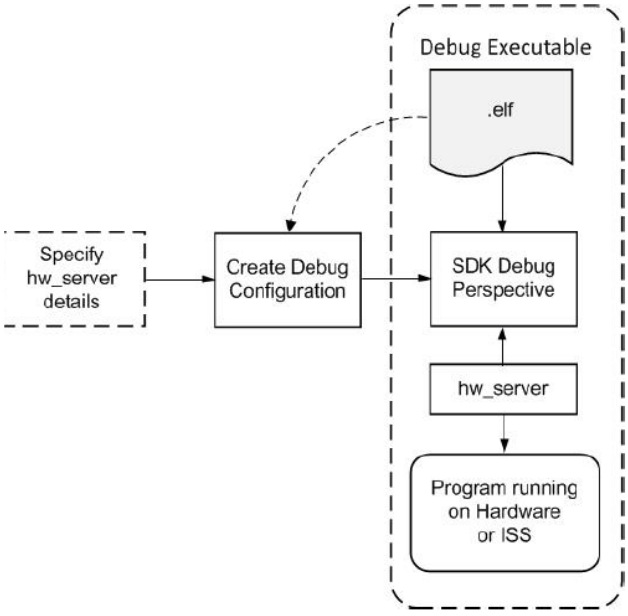


Figure 103- Debug Workflow

1.4.2.XILINX SYSTEM DEBUGGER COMMAND-LINE INTERFACE (XSDB)

Xilinx System Debugger Command-line Interface (XSDB) provides a user-friendly, interactive, and scriptable command line interface to Xilinx hw_server and other incarnations of TCF servers used by Xilinx. You can take full advantage of the features supported by the TCF servers as XSDB interacts with the TCF servers. A major source of inspiration for the commands in XSDB comes from Xilinx Microprocessor Debugger (XMD).

As with XMD, the XSDB scripting language is based on the Tools Command Language (Tcl).

Concerning the project; read and write in memory instruction were the most important since there was the only way to ensure that the algorithm was correct. Moreover, we use the write the instruction to store the binary images at the memory. XSDB commands can be found at Annex 5.

2. DEVELOPMENT–TEST–RESULTS

Each step has been tested on the board by two methods: First one consists on using directly the SDK to compile and debug the code. Second one consists on using an environment to cross-compile the C code for XtratuM, after that we will use a command line to debug and check the results. All the feedback of the Ninano is supply by the UART interface; therefore we must configure also the appropriate port to read the data.

2.1. SDK

Once we have developed the algorithm in C. We must test the algorithm on the board. Therefore, the first step consists on checking if the source code can compile. To compile the code we can use directly the SDK. Inside the SDK we can select several choices for compiling such as optimization modes, add specific libraries…. In fact, concerning out algorithm, in order to carry out the logarithm in base 2, we must add the math library.

Once everything works properly, the second step consists on loading a binary image on the memory. To manage that step, we have to use the XSDB console. XSDB supply us with functionalities which allow us to write on memory the binary data of a binary file. Figure 104 illustrates an example of how manage the step commented before.

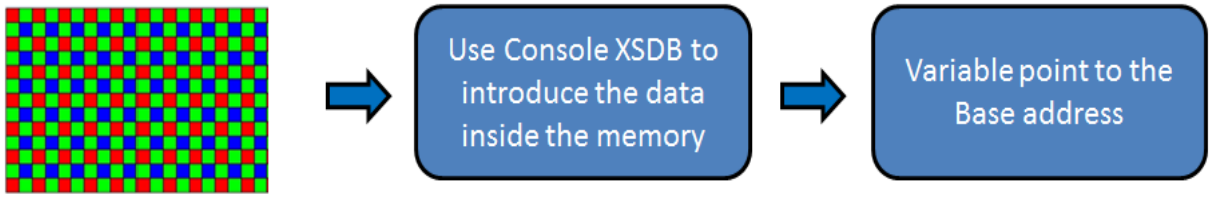
Next step consists on loading our code and run the program. Those functionalities are available with the SDK. To check if the packet has been generated in a properly way, we must read the right memory registers and compare that files with the files generated by simulation. To read the memory resources, we can use one more time the XSDB commands. To check the files we can create a simple code on Linux to check if there are any differences. At Annex 6, there is an example explaining the steps to run the code.

Results have been obtained for the followings algorithms: “EFFICIENT SEGMENTED FINAL ALGORITHM” and “SEGMENTED FINAL ALGORITHM”. Of course, results only concerns time computing performance since the compression rate is exactly the same as simulation.

In all cases the time are lower than 15 seconds, however it seems that the “efficient algorithm” is much quicker due to the reduction of memory access and operations.

Table 8- Time Computing Results SDK Environment

Segmented CCSDS-123- Sample Encoder			*Segmented CCSDS-123- Block Encoder*			*Efficient Algorithm- Block Encoder*			
Pattern 1	***Pattern 2***	***No Pattern***	***Pattern 1***	***Pattern 2***	***No Pattern***	***Mode 1***	***Mode 1 S***	***Mode 3***	***Mode 3***
12s	*12s*	*11s*	*11s*	*11s*	*10s*	2s	2s	2s	2s




```
//load in memory the file

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xsdb% connect
tcfchan#1
xsdb% targets
  1  APU
     2  ARM Cortex-A9 MPCore #0 (Running)
     3  ARM Cortex-A9 MPCore #1 (Running)
  4* xc7z030
xsdb% targets 1
xsdb% source C:/Outils_Nanosatellites/Package_hard/2016-02-05_Ninano_V1.332/
Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/Zynq_Ninano_hw_platform_0/ps7_init.tcl
xsdb% ps7_init
xsdb% ps7_post_config
xsdb% dow C:/Outils_Nanosatellites/Package_hard/2016-02-05_Ninano_V1.332/
Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/testPrimer/Debug/testPrimer.elf

Downloading Program -- C:/Outils_Nanosatellites/Package_hard/2016-02-05_Ninano_V1.332/
Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/testPrimer/Debug/testPrimer.elf
    section, .text: 0x00100000 - 0x0011ad5f
    section, .init: 0x0011ad60 - 0x0011ad77
    section, .fini: 0x0011ad78 - 0x0011ad8f
    section, .rodata: 0x0011ad90 - 0x0011b8e3
    section, .data: 0x0011b8e8 - 0x0011c1f3
    section, .eh_frame: 0x0011c1f4 - 0x0011c267
    section, .mmu_tbl: 0x00120000 - 0x00123fff
    section, .ARM.exidx: 0x00124000 - 0x00124007
    section, .init_array: 0x00124008 - 0x0012400f
    section, .fini_array: 0x00124010 - 0x00124013
    section, .bss: 0x00124014 - 0x0012407f
    section, .heap: 0x00124080 - 0x0012607f
    section, .stack: 0x00126080 - 0x0012987f

  0%    0MB   0.0MB/s  ??:?? ETA
100%    0MB   0.4MB/s  00:00
cannot update program counter

Successfully downloaded C:/Outils_Nanosatellites/Package_hard/2016-02-05_Ninano_V1.332/
Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/testPrimer/Debug/testPrimer.elf
xsdb% mwr -size h -bin -file C:/Users/Public/Hubble.txt 0x00F28080 4194304

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// program the FPGA
// run -->Launch the program in hardware
```

Figure 104- Store and Image on Memory

2.2.XTRATUM

First step consists on compiling the C code inside the adequate XtratuM environment. That means we must carry out a cross-compilation. A cross compilation consists on being capable of creating executable code for a platform other than the one on which the compiler is running. That generates several problems as libraries are often no-similar. Moreover, to generate the compilation we must create a “makefile”, where write all the required commands to generate the executable file.

As it has been explained; XtratuM [25] is a hypervisor, whose task consists on sharing in an optimal way the microprocessor resources. Each partition of the global system will have access to specific memory address registers and time slot of the processor. To configure all the properties for each partition a XML file can be used, there you can specify which memory addresses you are able to access, the rights you have upon that memory address, the frequency of the processor, UART configuration to have feedback….

Frequency specification for our entire test is $400\ MHz$. Concerning memory address, depending on which code will be implemented, we will use more or less memory.

2.2.1.SEGMENTED FINAL ALGORITHM

Figure 105 shows the final established memory Address allocation and figure 106 the XML file description.

```
#define ADDRESS_SAMPLES             ((unsigned short *) 0x10000000)

#define ADDRESS_SEGMENT             ((unsigned short *) 0x14000000)

#define ADDRESS_RESIDUALS           ((unsigned short *) 0x14100000)

#define ADDRESS_COMPRESSED_STREAM   ((unsigned char *) 0x14200000)

#define NUMBER_BYTES                ((unsigned int *) 0x17200800)
#define ERROR_ADDRESS               ((unsigned int *) 0x17200000)
```

Figure 105- Memory Address Allocation- Segmented Final Algorithm

```
<Area start="0x10000000" size="8MB" flags="uncacheable read-only" mappedAt="0x10000000"/>
<Area start="0x14000000" size="128KB"/>
<Area start="0x14200000" size="128KB"/>
<Area start="0x14200000" size="6MB"/>
<Area start="0x17200000" size="4KB"/>
```

Figure 106- XML file Description- Segmented Final Algorithm

2.2.2.EFFICIENT SEGMENTED FINAL ALGORITHM

Figure 107 shows the final established memory Address allocation and figure 108 the XML file description.

```
#define ADDRESS_SAMPLES             ((unsigned short *) 0x10000000)

#define ADDRESS_RESIDUALS           ((unsigned short *) 0x14000000)

#define ADDRESS_COMPRESSED_STREAM   ((unsigned char *) 0x14100000)

#define NUMBER_BYTES                ((unsigned int *) 0x17200800)
#define ERROR_ADDRESS               ((unsigned int *) 0x17200000)
```

Figure 107- Memory Address Allocation- Efficient Segmented Final Algorithm

```
<Area start="0x10000000" size="8MB" flags="uncacheable read-only" mappedAt="0x10000000"/>
<Area start="0x14000000" size="128KB"/>
<Area start="0x14100000" size="6MB"/>
<Area start="0x17200000" size="4KB"/>
```

Figure 108- XML file Description- Efficient Segmented Final Algorithm

For both cases, it will be necessary to use a small memory address to allocate the execution program. Moreover, in both cases we can see a memory zone where we only have selected read rights, the reason is that memory space is reserved for the original image that only need to be read.

To execute the code we must use the command line, moreover we uses CoolTerm to obtain all the feedback given by the UART interface. At Annex 6, there is example where the entire steps to execute the program are developed.

Table 9- Time Computing Results XtratuM Environment

Segmented CCSDS-123- Sample Encoder			*Segmented CCSDS-123- Block Encoder*			*Efficient Algorithm- Block Encoder*			
Pattern 1	***Pattern 2***	***No Pattern***	***Pattern 1***	***Pattern 2***	***No Pattern***	***Mode 1***	***Mode 1 S***	***Mode 3***	***Mode 3***
200s	*200s*	*180s*	*180s*	*180s*	*150s*	34s	34s	34s	34s

Table 9 illustrates the results obtained with XtratuM. As it can be seen, results show bad performances concerning the time computing and in comparison with the results with the SDK. The most likely problem could be a bad XtratuM configuration which can limit the time computing performances.

Part 7: Gantt Diagram

Gantt Diagram

Eye-Sat Compression algorithm

Tasks

Name	Begin date	End date
State of Art Review	3/1/16	3/14/16
Images Sofwares Tutorials	3/10/16	3/16/16
CCSDS-123 Standard C programming	3/10/16	4/8/16
Image Generation (Matlab)	3/25/16	4/1/16
CCSDS-122 Standard C programming	4/6/16	4/15/16
Interpolation Algorithm	4/6/16	4/22/16
Test Image Algorithm and Performance Study	4/15/16	4/29/16
Video Compression Algorithm 1	4/29/16	5/27/16
Sequence Image Generation (Matlab)	5/11/16	5/18/16
Video Compression Algorithm 2	5/18/16	5/30/16
Test Video Compression Algorithms	5/11/16	6/3/16
Image Compression Algorithm (Ninano)	6/3/16	6/10/16
Segmentation Functionality	6/10/16	6/17/16
Final Image Compression Algorithm (Ninano)	6/17/16	6/24/16
Test Final Image Compression Algorithm (Ninano)	6/21/16	6/27/16
Optimitation (Ninano)	6/27/16	7/1/16
Rapport	3/16/16	7/29/16
Other Projects	7/1/16	8/26/16
Paper CCSDS Conference	8/1/16	8/26/16
Presentation	8/22/16	8/26/16

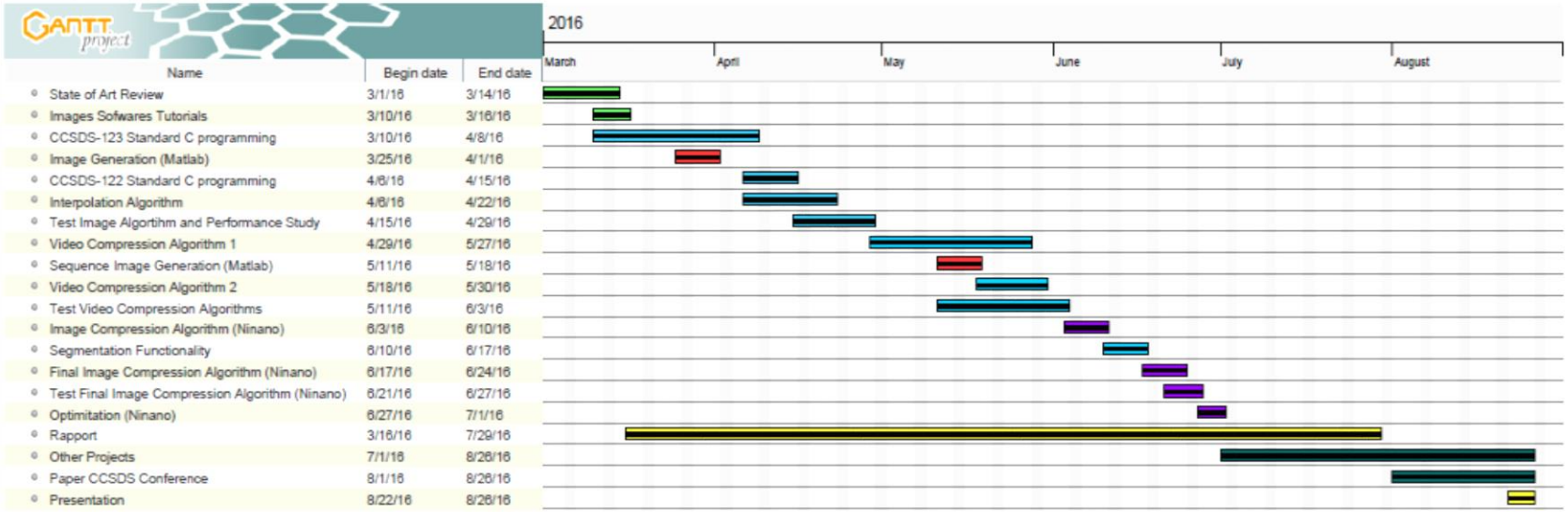
Eye-Sat Compression algorithm

Resources

Name	Default role
Eclipse C/C++	developer
Matlab 2013b	developer
Ninano -- SDK xilinx	developer
ENVI	tester
Microsoft Word	doc writer
Gantt Project	developer
Microsoft Power Point	developer

Eye-Sat Compression algorithm

Gantt Chart



Name	Begin date	End date
State of Art Review	3/1/16	3/14/16
Images Sofwares Tutorials	3/10/16	3/16/16
CCSDS-123 Standard C programming	3/10/16	4/8/16
Image Generation (Matlab)	3/25/16	4/1/16
CCSDS-122 Standard C programming	4/8/16	4/15/16
Interpolation Algorithm	4/8/16	4/22/16
Test Image Algortihm and Performance Study	4/15/16	4/29/16
Video Compression Algorithm 1	4/29/16	5/27/16
Sequence Image Generation (Matlab)	5/11/16	5/18/16
Video Compression Algorithm 2	5/18/16	5/30/16
Test Video Compression Algorithms	5/11/16	6/3/16
Image Compression Algorithm (Ninano)	6/3/16	6/10/16
Segmentation Functionality	6/10/16	6/17/16
Final Image Compression Algorithm (Ninano)	6/17/16	6/24/16
Test Final Image Compression Algorithm (Ninano)	6/21/16	6/27/16
Optimitation (Ninano)	6/27/16	7/1/16
Rapport	3/16/16	7/29/16
Other Projects	7/1/16	8/26/16
Paper CCSDS Conference	8/1/16	8/26/16
Presentation	8/22/16	8/26/16

Eye-Sat Compression algorithm

Resources Chart

Part 8: Conclusion

Along this project and especially along this report, we have presented the work methodology to carry out and manage the development and the implementation of the image compression algorithm for the EYE-SAT mission. This algorithm will be implemented on the satellite board Ninano. Ninano is a System-on-Chip (SoC) technology; especially it uses a Xilinx ZYNQ® All Programmable SoC which includes a programmable logic module (FPGA) and a Dual-core ARM® Cortex™-A9 processor with NEON™ DSP/FPU Engines. As the compression algorithms it is not the unique software available inside the Ninano, it will be used a Hypervisor to manage the time partitioning. XtratuM is the selected Hypervisor.

Mission Images must be compressed as we have a transmission time limit, therefore the compression algorithm will be used to reduce the data to increase the mission capacities by trying to duplicate the number of transmitted images. There are three kinds of images to be compressed. The first one will be a black image which will be used to calibrate the sensors. The second subset of images will be a sequence of infrared images acquired by the sensor with the proper spectral filter. Finally, RGB images will be acquired by the same CFA sensor but without any spectral filter. Especially last subset of images will produce the already explained Bayer Pattern

To develop the algorithms to compress the different subsets of images, we have taken into account the characteristics and requirements of the mission. In fact, the most important requirement consists on developing a lossless compression algorithm on board which will be able to fulfill the time-computing limits and to have good performance concerning the compression rate.

As we have sequences of images we have tested two different methodologies. The first methodology consists on using image compression algorithms, in fact as we have several subsets of images we have focus on developing algorithms based on the CCSDS-123 Hyperspectral image compression algorithms, which is able to exploit spectral information in case of CFA images. The second methodology is based on the fact that we have sequences of images and consist on implementing video compression algorithms.

After implementing, testing and taking into account all the requirements. We have decided to implement an image compression algorithm, whose entropy encoder is the block adaptive entropy encoder of the CCSDS-123 Standard. Concerning the predictor, we have selected an easier predictor with the goal of accomplishing the time-computing requirements. Predictor has been implemented in an optimal way in order to reduce the number of memory access, the number of operations and the use of the memory registers. Moreover, the predictor has several modes in order to be able to have the best performance as possible independently of the subset of images we are compressing. A segmentation technique is included at this algorithm with the goal of trying to decrease as much as possible the effects of the transmission errors.

Compression result show good performance, accomplishing in several examples compression rates higher than 2 and improving in some case the performances of the CCSDS-123 predictor. These cases are usually at CFA images because the predictor has been generated to take into account how that images are generated. Algorithm time execution has been tested by two tools on the satellite board. The first tool is the SDK of Xilinx, here time-computing results are much better than we have expected. The second tool, where the algorithm has been tested is using the Hypervisor XtratuM. As we have expected, the results are worse concerning time-computing because of the time-partitioning and a lower microprocessor frequency.

Although the compression performances and the time-complexity has really good performances, we actually do not know how the noise is going to be, therefore the real compression rate performances could vary depending the real environmental conditions.

References

[1] Khalid Sayood "Introduction to Data Compression"

[2] Wei-Yi Wei "An Introduction to Image Compression"

[3] King-Hong Chung and Yuk-Hee Chan "A Lossless Compression Scheme for Bayer Color Filter"

[4] Sang-Yong Lee, Member, IEEE, and Antonio Ortega, Fellow, IEEE "A Novel Approach for Compression of Images Captured using Bayer Color Filter Arrays"

[5] David Alleysson1, Sabine Süsstrunk1, and Jeanny Hérault2 "Color demosaicing by estimating luminance and opponent chromatic signals in the Fourier domain"

[6] Sung Hee Park and Albert No "Analysis on Color Filter Array Image Compression Methods"

[7] Ning Zhang and Xiaolin Wu "LOSSLESS COMPRESSION OF COLOR MOSAIC IMAGES"

[8] Olivier Losson, Ludovic Macaire, Yanqin Yang "Comparison of color demosaicing methods"

[9] Bruno Carpentieri, Raffaele Pizzolante "Lossless, Multiband, on Board, Compression of Hyperspectral Image "

[10]Eric Welch "A Study of the use of SIMD instructions for two image processing algorithms"

[11]A. M. Di Giorgio, J. S. Liu, G. Giusi, G. Palamara INAF –IAPS, Rome - Italy Mark Cropper, Sami Niemi, Jamie Denniston MSSL, UK " Euclid Visible Imager on-board lossless data compression: performance assessment trade-off activities"

[12] CCSDS 121.0-B-2, Lossless Data Compression, Blue Book, Issue 2, 5/2012

[13] CCSDS 122.0-B-1, Image Data Compression, Blue Book Issue 1, 11/2005

[14] CCSDS 120.0-G-2, Lossless Data Compression, Green Book, Issue 2, 12/2006

[15] Lossless Multispectral & Hyperspectral Image Compression Recommendation for Space Data System Standards, CCSDS 123.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 2012.

[16] Estanislau Augé, Jose Enrique Sánchez, Aaron Kiely, Ian Blanes, and Joan Serra-Sagristàa "Performance impact of parameter tuning on the CCSDS-123 lossless multi- and hyperspectra image compression standard"

[17] Fore June "An Introduction to Video Compression in C/C++"

[18] Louise H. Crockett Ross A. Elliot Martin A. Enderwitz Robert W. Stewart "The Zynq Book, Embedded Processing with the ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC"

[19] Xilinx "Zynq-7000 All Programmable SoC Technical Reference Manual"

[20]http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/SDK_Doc/SDK_concepts/sdk_c_xsd_over.html

[21]http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_3/SDK_Doc/SDK_references/reference_mwr.html

[22] MARTIN ROULANCE "ARCHITECTURE SYSTEM-ON-CHIP POUR CALCULATEUR NINANO"

[23] http://fr.mathworks.com/

[24] Exploring ENVI Research Systems Inc.

[25] Manuel Munoz, Javier Coronel, Miguel Masmano, Alfons Crespo *"XtratuM Hypervisor for ARM CORTEX-A9 Volume 2: User Manual"*

Annex

Annex 1: Images

1.INTRODUCTION

The goal of this annex consists on carrying out a study of the entropy of the images. Therefore, why should we make this study? The answer is simply because of the entropy give us a theoretical limit for the lossless compression rate. This limit cannot be improved for any implemented compression algorithm. Moreover this annex defines basic concepts as joint entropy and conditional entropy. Usually, when we start to study the information theory, we initialize our learning with source without memory that is entropy of order 0.

Concerning the study of the images, we cannot consider each pixel as an independent symbol, in fact each pixel has correlation with the closer pixels, because spatial redundancy. In order to manage this case, we should define new ways of entropy to define a new theoretical limit. Those are the joint entropy and conditional entropy. In fact, concerning our case, the conditional entropy or entropy of order N will be our new limit. In case of an image of X pixels the maximum order N will be $X-1$, as all the pixels of the image will have any kind of redundancy with the evaluate pixel. To sum up that case, we are considered the image as a unique symbol which is generating determined information.

2.STUDY

Entropy can be seen as a theoretic limit of the lossless compression rate. In fact, we define the entropy of order 0 as the information average in bits/pixel.

$$H_0(S) = \sum_{i=1}^{i=L_S} p(s_i).I(s_i) = -\sum_{i=1}^{i=L_S} p(s_i).\log_2(p(s_i)) \quad bits/symbole$$

Taking into account this quantity, we only have a theoretic model of the encoder because we are considering an uncorrelated source, where each pixel is independent from the closer pixels.

As image can be seen as a correlated source, it is very useful to define the joint entropy and the conditional entropy of order N.

Joint entropy

$$H_n^{Cjte}(S) = \sum_{s_1 \in U} \cdots \sum_{s_{n+1} \in U} p(s_1 \cdots s_{n+1}).\log_2\left(\frac{1}{p(s_1 \cdots s_{n+1})}\right)$$

Conditional entropy

$$H_n^{Cdl}(S) = \sum_{s_1 \in U} \cdots \sum_{s_{n+1} \in U} p(s_1 \cdots s_{n+1}).\log_2\left(\frac{1}{p(s_1/s_2 \cdots s_{n+1})}\right)$$

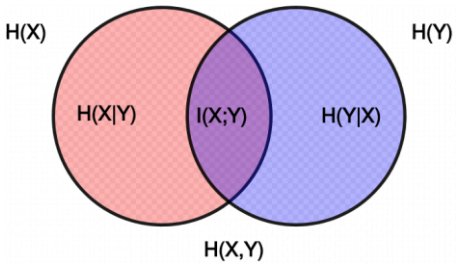
To reduce the complexity, we will calculate the order 1 and the order 2.

Joint entropy order 1

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y).$$

Conditional entropy order 1

$$H(Y|X) = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} p(x,y)\log\frac{p(x)}{p(x,y)}.$$



We have select three different possibilities to calculate the entropy of order 1:

The first case corresponds with having the knowledge of the west pixel. That pixel share information with actual pixel and we can consider as a prior information, therefore that information that has been shared by the two pixels, we must associated to just one pixel for the global computation of the provided Information generated for the image.

X

$X_{1,1}$	$X_{2,1}$	$X_{3,1}$	...	...	...	$X_{2048,1}$
$X_{1,2}$	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
$X_{1,2048}$	$X_{2,2048}$	...	...	...	...	$X_{2048\ 2048}$

Y

0	$X_{1,1}$	$X_{2,1}$	$X_{3,1}$	...	...	$X_{2047,1}$
0	$X_{1,2}$	...	...	...	...	...
0	...	...	...	...	...	...
0	...	...	...	...	...	...
0	...	...	...	...	...	...
0	...	...	...	...	...	...
0	$X_{1,2048}$	$X_{2,2048}$	...	...	...	$X_{2047\ 2048}$

The second case corresponds with having the a priori information of the north pixel. Finally, the third case that we have chosen corresponds with the North-West pixel. We can test with any other pixel of the image, but because of the spatial redundancy, the closer will be the pixel more joint information they will share.

X

$X_{1,1}$	$X_{2,1}$	$X_{3,1}$	...	...	...	$X_{2048,1}$
$X_{1,2}$	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
$X_{1,2048}$	$X_{2,2048}$	...	...	...	...	$X_{2048\ 2048}$

Z

0	0	0	0	0	0	0
$X_{1,1}$	$X_{1,2}$	...	...	...	...	...
$X_{1,2}$	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
$X_{1,2047}$	$X_{2,2047}$	...	...	...	...	$X_{2048\ 2047}$

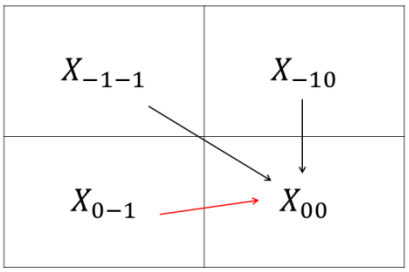
X

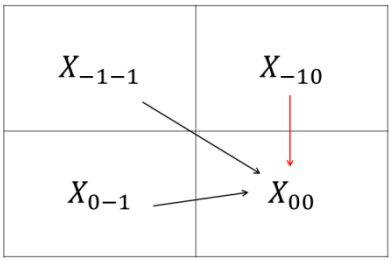
$X_{1,1}$	$X_{2,1}$	$X_{3,1}$	...	...	...	$X_{2048,1}$
$X_{1,2}$	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
$X_{1,2048}$	$X_{2,2048}$	...	...	...	...	$X_{2048\ 2048}$

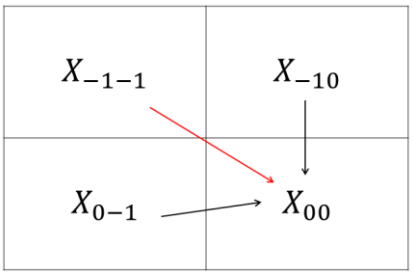
W

0	0	0	0	0	0	0
0	$X_{1,1}$	$X_{2,1}$	$X_{3,1}$	...	...	$X_{2047,1}$
0	...	...	...	...	...	...
0	...	...	...	...	...	...
0	...	...	...	...	...	...
0	...	...	...	...	...	...
0	$X_{1,2047}$	...	...	...	...	$X_{2047\ 2047}$

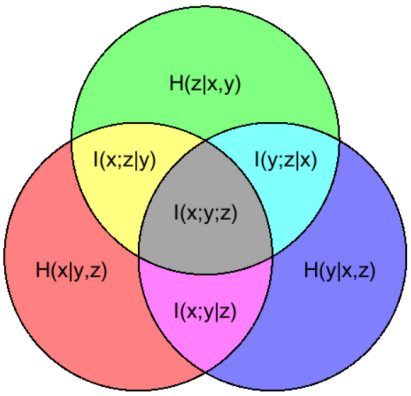
The way to calculate the entropy is equivalent to predict with these pixels at the predictor step:



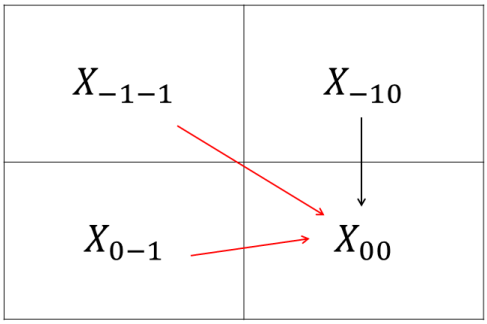


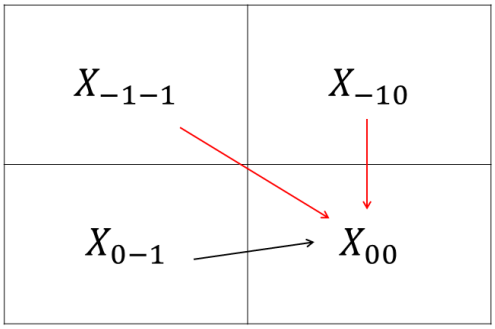


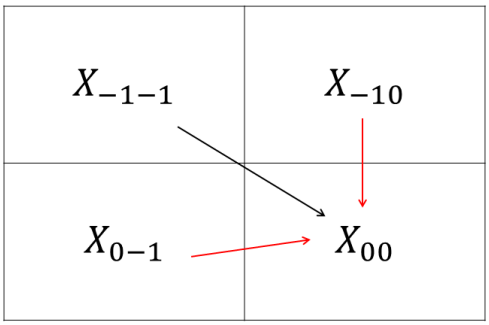
For order 2, instead of having into account only one pixel, we have two pixels with could share information with the actual pixel and moreover between those two pixel, information can be shared. Next figure can illustrate that phenomenon.



An equivalent phenomenon can happen if we predict with two close pixels the actual pixel, as there is redundant information between them, we can have better ratios of compression.



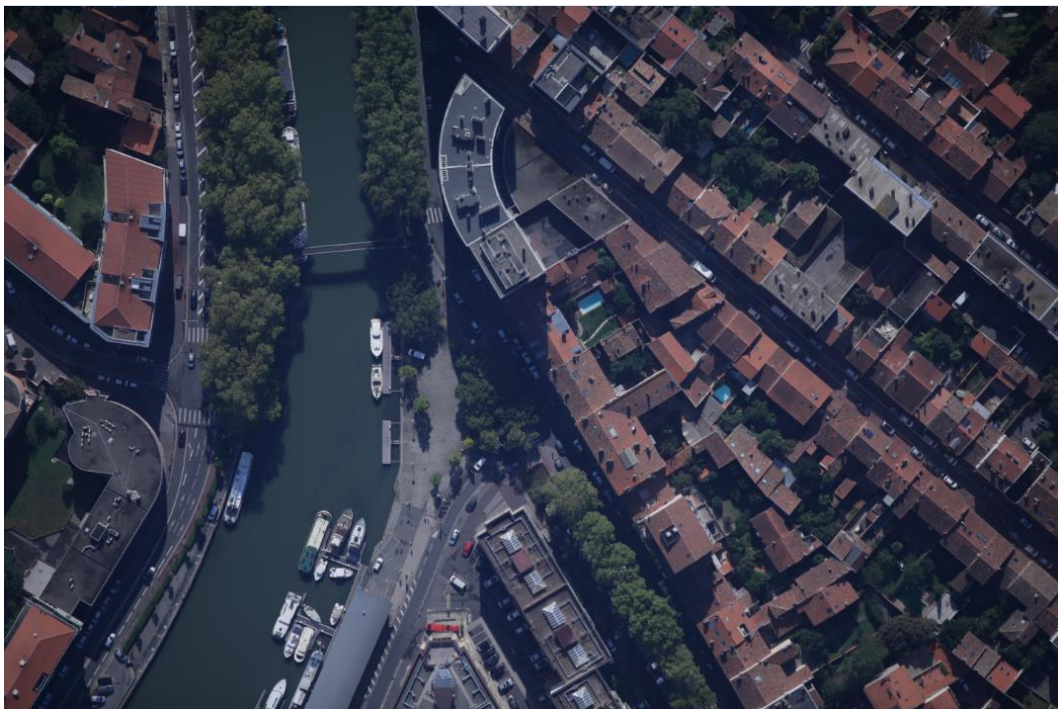




At last part of this Annex we are going to calculate the entropy of order 0, 1 and 2 for all the images. In all the cases we are going to see that our compression algorithm has better rate than those limits, therefore we can assume that our predictor is estimating in a really good way the pixels. By the other hand, if we will calculate the entropy on higher orders the rate of our compression algorithm will be worse, as we said the conditional entropy is a theoretical limit that cannot be improved.

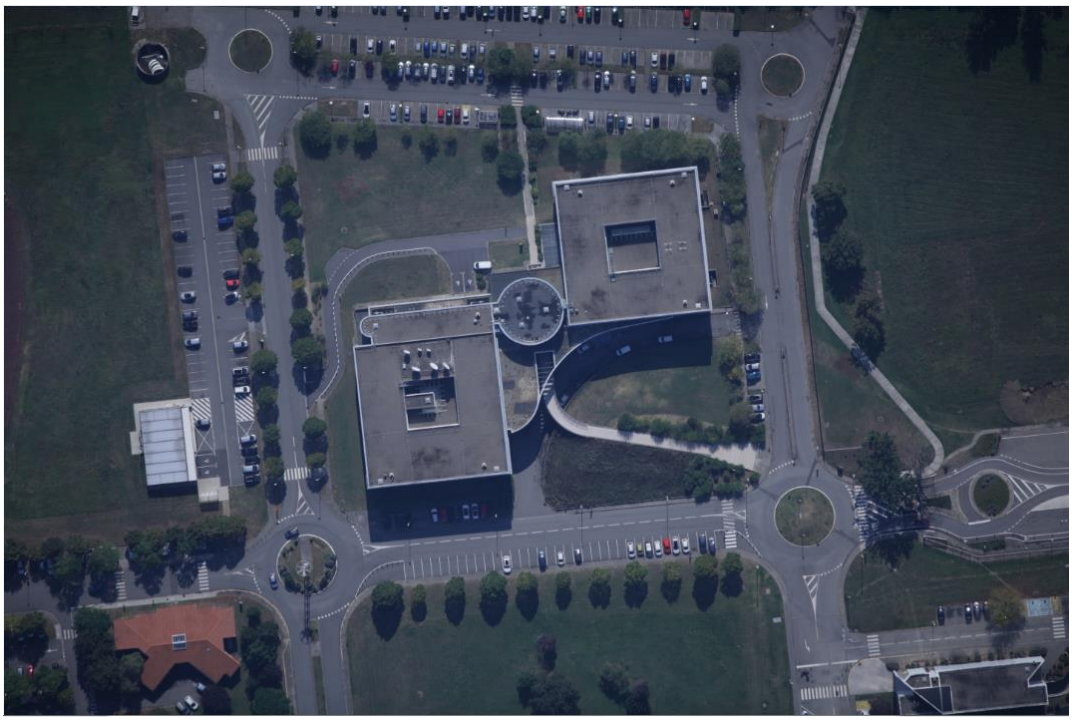
Finally, we define the calculated tables.
Order 0 – Consider the source, as a without memory source
Order 1-1 – We consider that we have West pixel a priori.
Order 1-2 – We consider that we have North pixel a priori.
Order 1-3 – We consider that we have North-West pixel a priori.
Order 2-1 – We consider that we have West and North-West pixels a priori.
Order 2-2 – We consider that we have North-West and North pixels a priori.
Order 2-2 – We consider that we have West and North pixels a priori.

Image Toulouse 1



Entropy	*Order 0*	*Order 1-1*	*Order 1-2*	*Order 1-3*	*Order 2-1*	*Order 2-2*	*Order 2-3*
Image 1	*6.6615*	*5.4852*	*5.5041*	*5.3411*	*4.6172*	*4.4950*	*4.5139*

Image Toulouse 2

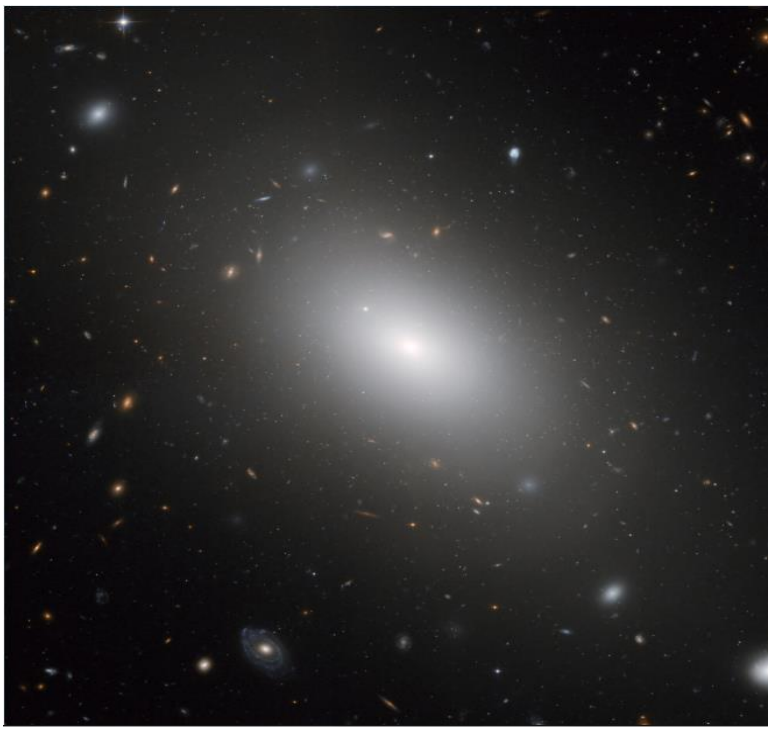


Entropy	*Order 0*	*Order 1-1*	*Order 1-2*	*Order 1-3*	*Order 2-1*	*Order 2-2*	*Order 2-3*
Image 2	*6.5608*	*5.1521*	*5.1525*	*5.0497*	*4.0959*	*4.0188*	*4.0193*

Image Toulouse 3

Entropy	*Order 0*	*Order 1-1*	*Order 1-2*	*Order 1-3*	*Order 2-1*	*Order 2-2*	*Order 2-3*
Image 3	*6.5408*	*5.5402*	*5.5859*	*5.2059*	*4.8240*	*4.5390*	*4.5847*

Image Space 1 – Hubble—12 bits – noise



Entropy	*Order 0*	*Order 1-1*	*Order 1-2*	*Order 1-3*	*Order 2-1*	*Order 2-2*	*Order 2-3*
Image 4	*8.6145*	*5.8894*	*5.8902*	*6.0239*	*5.2084*	*5.2418*	*5.2425*

Image Space 2 – Hubble1—12 bits – noise

Entropy	*Order 0*	*Order 11*	*Order 12*	*Order 13*	*Order 21*	*Order 22*	*Order 23*
Image 5	*5.9844*	*4.6552*	*4.6554*	*4.7169*	*4.3229*	*4.3383*	*4.3385*

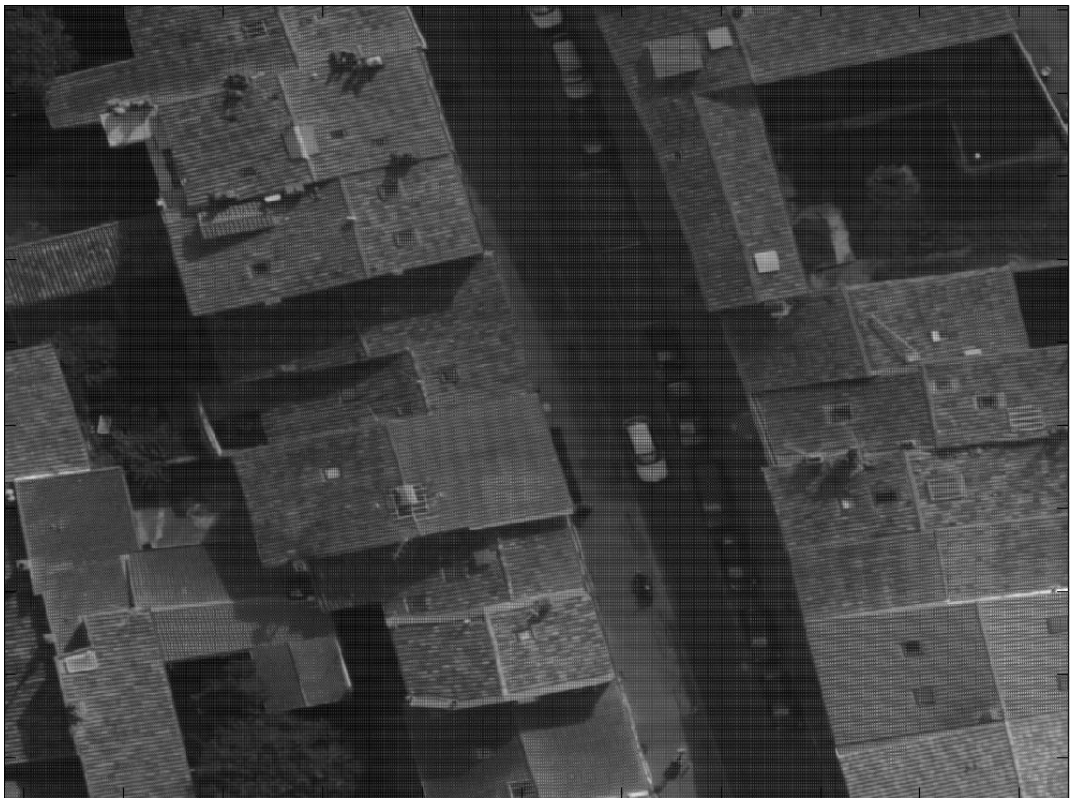
Image Space 3– Space—12 bits – noise

Entropy	*Order 0*	*Order 1-1*	*Order 1-2*	*Order 1-3*	*Order 2-1*	*Order 2-2*	*Order 2-3*
Image 6	*5.0289*	*4.7287*	*4.7258*	*4.7728*	*4.5598*	*4.5716*	*4.5687*

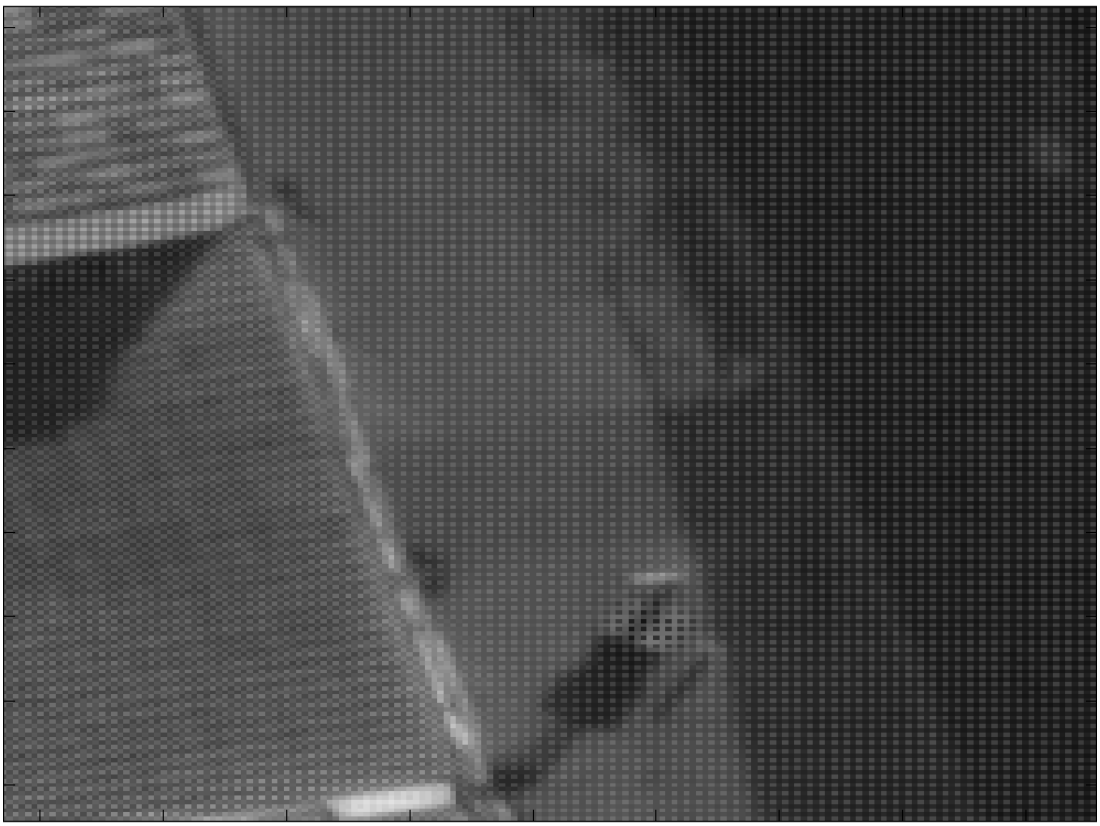
Image Space 4– milky-way—12 bits – noise

Entropy	*Order 0*	*Order 1-1*	*Order 1-2*	*Order 1-3*	*Order 2-1*	*Order 2-2*	*Order 2-3*
Image 7	*7.3048*	*5.7703*	*5.7618*	*5.6866*	*5.3846*	*5.3658*	*5.3572*

The real aspects of the Images, as pictures will be acquired by a CFA captor, seem like:



If we use a zoom; we can notice of the CFA pattern.



Annex 2: Compression Main Code

1.SEGMENTED-FINAL ALGORITHM

```
Select Mode( );
Initialization :
• Errors Address Pointer;
• Num Bytes Address Pointer;
• Image Samples Pointer;
• Segment Samples Pointer
• Residuals Samples Pointer;
• Compressed Segment(0)Pointer;
• Compression Parameters;
• Segment = sqrt(Num_block)
for( i = 0 ; j < Segment; i + +)
    for( j = 0 ; j < Segment; j + +)
        z = j + i * Segment
        Create Segment(Images Samples, Segment, i, j);
        Predict (Segment, Residuals Samples, Mode);
        Num = Encoder (Residuals Samples, Compressed Segment(z), Parameters);
        if (z ≠ 0)
            Compressed Segment(z)Pointer = Compressed Segment(z − 1)Pointer + Num;
        end if
        Num Bytes[z] = Num;
    end for
end for
```

2.EFFICIENT SEGMENTED FINAL ALGORITHM

```
Select Mode( );
Initialization :
• Errors Address Pointer;
• Num Bytes Address Pointer;
• Image Samples Pointer;
• Residuals Samples Pointer;
• Compressed Segment(0)Pointer;
• Compression Parameters;
• Segment = sqrt(Num_block)
for( i = 0 ; j < Segment; i + +)
    for( j = 0 ; j < Segment; j + +)
        z = j + i * Segment
        Create Predict Samples (Images Samples, Residuals, i, j);
        Num = Encoder (Residuals Samples, Compressed Segment(z), Parameters);
        if (z ≠ 0)
            Compressed Segment(z)Pointer = Compressed Segment(z − 1)Pointer + Num;
        end if
        Num Bytes[z] = Num;
    end for
end for
```

Annex 3: Matlab Codes

1.IMAGE GENERATION

```
%% %% open jpg
close all
clear all
clc
%A = imread('Hubble.jpg');
I = imread('Hubble.jpg');
%I = imread('Casper_VP1_1.JPG');


figure;
imagesc(I);

I= double(I(1:2048,1:2048,:));

%% For 132oolbox132_images

Image_red = I(:,:,1);
Image_green = I(:,:,2);
Image_blue = I(:,:,3);

%red modification
Image_red(:,1:2:2048) = 0;
Image_red(2:2:2048,:) = 0;
%blue modification
Image_blue(:,2:2:2048) = 0;
Image_blue(1:2:2048, :) = 0;
%green 132oolbox132tion
Image_green(1:2:2048,2:2:2048) = 0;
Image_green(2:2:2048,1:2:2048) = 0;

I = Image_red +Image_blue +Image_green;


%% Create the Bayer pattern

% Everything less the 50 is noise

Detect1 = (I(:,:,1)>50)*2^4;
Detect2 = (I(:,:,2)>50)*2^4;
Detect3 = (I(:,:,3)>50)*2^4;

Detect1(Detect1 == 0 ) = 1;
Detect2(Detect2 == 0 ) = 1;
Detect3(Detect3 == 0 ) = 1;
% figure ;
% imagesc(A) ;

% Noise generation
sigma = 4 ;
noise = sigma*randn(size(I,1),size(I,2));

Image_red = I(:,:,1).*Detect1 + noise ;
Image_green = I(:,:,2).*Detect2 + noise;
Image_blue = I(:,:,3).*Detect3 + noise;

%red modification
```

```
Image_red(:,1:2:2048) = 0;
Image_red(2:2:2048,:) = 0;
%blue modification
Image_blue(:,2:2:2048) = 0;
Image_blue(1:2:2048,:) = 0;
%green 133oolbox133tion
Image_green(1:2:2048,2:2:2048) = 0;
Image_green(2:2:2048,1:2:2048) = 0;

I = Image_red +Image_blue +Image_green;
I=uint16(I);
figure;
imagesc(I);

%% Create binary

I= I';
I = I(:);

length = size(I,1);

fi = fopen('Hubble12bits.txt', 'a+');

for i=1:length
nybtes=fwrite(fi, I(i),'uint16',0,'ieee-le');

end

fclose(fi);
```

2.SEQUENCE GENERATION

```
%% Generated binary file of sequences of images

clear;
clc;
close all;

%% Generation of 5 differents frames to simulate erros

I = imread('Casper_VP1_3.JPG');
%I = imread('Space1.jpg');

% figure;
% imagesc(I);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 1 iteration

%Generation of three bands

R= I(:,:,1);
G= I(:,:,2);
B= I(:,:,3);

% motion matrix

% M = zeros(5);
% M(end,end) = 0.5;
% M(end-1,end-1) = 0.5;
M = zeros(5);
M(1,1) = 0.5;
```

```
M(2,2) = 0.5;


% motion bands

R1 = conv2(doubleI,M);
G1 = conv2(double(G),M);
B1 = conv2(double(B),M);

I1(:,:,1) = uint8(R1);
I1(:,:,2) = uint8(G1);
I1(:,:,3) = uint8(B1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 2 iteration

% motion matrix

M = zeros(5);
M(end,1) = 0.5;
M(end-1,2) = 0.5;

% motion bands

R2 = conv2(double(R1),M);
G2 = conv2(double(G1),M);
B2 = conv2(double(B1),M);

I2(:,:,1) = uint8(R2);
I2(:,:,2) = uint8(G2);
I2(:,:,3) = uint8(B2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 3 iteration

% motion matrix

M = zeros(4);
M(1,end) = 0.5;
M(2,end-1) = 0.5;

% motion bands

R3 = conv2(double(R2),M);
G3 = conv2(double(G2),M);
B3 = conv2(double(B2),M);

I3(:,:,1) = uint8(R3);
I3(:,:,2) = uint8(G3);
I3(:,:,3) = uint8(B3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 4 iteration

% motion matrix

% M = zeros(6);
% M((end)/2,end) = 1;

M = zeros(6);
M(1,(end)/2) = 1;

% motion bands

R4 = conv2(double(R3),M) ;
G4 = conv2(double(G3),M);
B4 = conv2(double(B3),M);
```

```
I4(:,:,1) = uint8(R4);
I4(:,:,2) = uint8(G4);
I4(:,:,3) = uint8(B4);

%% Generate the Bayers Frame

I = I(1025:3072,1025:3072, :);
I1 = I1(1025-3:3072-3,1025:3072, :);
I2 = I2(1025+4:3072+4,1025-3:3072-3, :);
I3 = I3(1025+10:3072+10,1025+15:3072+15, :);
I4 = I4(1025:3072,1025+5:3072+5, :);


%%% 1 FRAME

%red modification
I(:,1:2:2048,1) = 0;
I(2:2:2048,:,1) = 0;
%blue modification
I(:,2:2:2048,3) = 0;
I(1:2:2048,:,3) = 0;
%green modifcation
I(1:2:2048,2:2:2048,2) = 0;
I(2:2:2048,1:2:2048,2) = 0;

%%% 2 FRAME

%red modification
I1(:,1:2:2048,1) = 0;
I1(2:2:2048,:,1) = 0;
%blue modification
I1(:,2:2:2048,3) = 0;
I1(1:2:2048,:,3) = 0;
%green modifcation
I1(1:2:2048,2:2:2048,2) = 0;
I1(2:2:2048,1:2:2048,2) = 0;

%%% 3 FRAME

%red modification
I2(:,1:2:2048,1) = 0;
I2(2:2:2048,:,1) = 0;
%blue modification
I2(:,2:2:2048,3) = 0;
I2(1:2:2048,:,3) = 0;
%green modifcation
I2(1:2:2048,2:2:2048,2) = 0;
I2(2:2:2048,1:2:2048,2) = 0;

%%% 4 FRAME

%red modification
I3(:,1:2:2048,1) = 0;
I3(2:2:2048,:,1) = 0;
%blue modification
I3(:,2:2:2048,3) = 0;
I3(1:2:2048,:,3) = 0;
%green modification
I3(1:2:2048,2:2:2048,2) = 0;
I3(2:2:2048,1:2:2048,2) = 0;

%%% 5 FRAME
```

```
%red modification
I4(:,1:2:2048,1) = 0;
I4(2:2:2048,:,1) = 0;
%blue modification
I4(:,2:2:2048,3) = 0;
I4(1:2:2048,:,3) = 0;
%green modification
I4(1:2:2048,2:2:2048,2) = 0;
I4(2:2:2048,1:2:2048,2) = 0;

clearvars -except I I1 I2 I3 I4 ;

% Representation

% figure;
% imagesc(I);
% figure;
% imagesc(I1);
% figure;
% imagesc(I2);
% figure;
% imagesc(I3);
% figure;
% imagesc(I4);

I  = I(:,:,1)  + I(:,:,2)  + I(:,:,3);
I1 = I1(:,:,1) + I1(:,:,2) + I1(:,:,3);
I2 = I2(:,:,1) + I2(:,:,2) + I2(:,:,3);
I3 = I3(:,:,1) + I3(:,:,2) + I3(:,:,3);
I4 = I4(:,:,1) + I4(:,:,2) + I4(:,:,3);

Frames(:,:,1) = I;
Frames(:,:,2) = I1;
Frames(:,:,3) = I2;
Frames(:,:,4) = I3;
Frames(:,:,5) = I4;


fi = fopen('imagen3_toulouse_sequence.txt', 'a+');

for j = 1:size(Frames,3)

    % Selection of the frame
    ImagenFinal = Frames( :, :,j)' ;
    %  Adecuate the Image
    ImagenFinal = ImagenFinal( :) ;

    for i=1:length(ImagenFinal)

    nybtes=fwrite(fi, ImagenFinal(i),'uint16',0,'ieee-le');

    end
end
fclose(fi);
```

3.DEMOSAICING METHOD

3.1.METHOD 1

```
%%%%%%%% Demosaicing methods %%%%%%
```

```
%% Bilinear interpolation

clear all;
close all;
clc;

%%%% Creation of Bayer Matrix

Imagen_out_1 = imread('Casper_VP1_1.JPG');
Imagen_out_1 = Imagen_out_1(1:2048,1:2048, :);
% Imagen_out_11 = Imagen_out_1(:,:,1);
% Imagen_out_21 = Imagen_out_1(:,:,2);
% Imagen_out_31 = Imagen_out_1(:,:,3);
figure;
imagesc(Imagen_out_1);
% figure;
% imagesc(Imagen_out_11);
% figure;
% imagesc(Imagen_out_21);
% figure;
% imagesc(Imagen_out_31);

%inizialization
Imagen_in_bayer = Imagen_out_1;
Imagen_out_bayer = Imagen_out_1;
%red modification
Imagen_in_bayer(:,1:2:2048,1) = 0;
Imagen_in_bayer(2:2:2048,:,1) = 0;
%blue modification
Imagen_in_bayer(:,2:2:2048,3) = 0;
Imagen_in_bayer(1:2:2048,:,3) = 0;
%green 137oolbox137tion
Imagen_in_bayer(1:2:2048,2:2:2048,2) = 0;
Imagen_in_bayer(2:2:2048,1:2:2048,2) = 0;
%Representation
% figure;
% imagesc(Imagen_in_bayer);
%
%%%% Definition of the planes

ImagenR = Imagen_in_bayer(:,:,1);
ImagenG = Imagen_in_bayer(:,:,2);
ImagenB = Imagen_in_bayer(:,:,3);

% figure;
% imagesc(ImagenR) ;
% figure ;
% imagesc(ImagenG) ;
% figure ;
% imagesc(ImagenB) ;
%
%%%% Matrix Generation


HR = [1 2 1 ;…
    2 4 2;…
    1 2 1];


HB = [1 2 1;…
    2 4 2;…
    1 2 1];

HG = [0 1 0;…
```

```
    1 4 1;…
    0 1 0];

HR = 0.25*HR;
HB = 0.25*HB;
HG = 0.25*HG;

%%%% bilinear interpolation

ImagenRest = conv2(double(ImagenR),HR,'same');
ImagenBest = conv2(double(ImagenB),HB,'same');
ImagenGest = conv2(double(ImagenG),HG,'same');

Imagen_out_bayer(:,:,1) = uint8(ImagenRest);
Imagen_out_bayer(:,:,2) = uint8(ImagenGest);
Imagen_out_bayer(:,:,3) = uint8(ImagenBest);

%%%% Representation;
figure;
imagesc(Imagen_in_bayer);


figure;
ax(1)=subplot(1,3,1);
imagesc(Imagen_out_1);
title('RGB');
ax(2)=subplot(1,3,2);
imagesc(Imagen_in_bayer);
title('Bayer Pattern');
ax(3)=subplot(1,3,3);
imagesc(Imagen_out_bayer);
title('Demosaicing RGB');
linkaxes(ax);



% Reobtain the real values
ImagenR = double(Imagen_out_1(:,:,1));
ImagenG = double(Imagen_out_1(:,:,2));
ImagenB = double(Imagen_out_1(:,:,3));

ImagenRest = round(ImagenRest);
ImagenGest = round(ImagenGest);
ImagenBest = round(ImagenBest);


errorR = ImagenR - ImagenRest;
errorG = ImagenG - ImagenGest;
errorB = ImagenB - ImagenBest;


figure;
ax(1)=subplot(1,3,1);
imagesc(abs(errorR));
title(,RED ERROR');
ax(2)=subplot(1,3,2);
imagesc(abs(errorG));
title('GREEN ERROR');
ax(3)=subplot(1,3,3);
imagesc(abs(errorB));
title('BLUE ERROR');
linkaxes(ax);
```

```
% calculate the mean-square error

% only with image processing
% mseR = immse(ImagenR,ImagenRest);
% mseG = immse(ImagenG,ImagenGest);
% mseB = immse(ImagenB,ImagenBest);
%
% manual mean-square-error
errorR = errorR.^2;
errorG = errorG.^2;
errorB = errorB.^2;

mseR = sum(sum(errorR))/(size(errorR,1)*size(errorR,2));
mseG = sum(sum(errorG))/(size(errorG,1)*size(errorG,2));
mseB = sum(sum(errorB))/(size(errorB,1)*size(errorB,2));
```

3.2.METHOD 2

```
%% Use Spectral information

clear all;
close all;
clc;

%%%% Creation of Bayer Matrix

Imagen_out_1 = imread('Casper_VP1_1.JPG');
Imagen_out_1 = Imagen_out_1(1:2048,1:2048, :);
% Imagen_out_11 = Imagen_out_1(:,:,1);
% Imagen_out_21 = Imagen_out_1(:,:,2);
% Imagen_out_31 = Imagen_out_1(:,:,3);
figure;
imagesc(Imagen_out_1);
% figure;
% imagesc(Imagen_out_11);
% figure;
% imagesc(Imagen_out_21);
% figure;
% imagesc(Imagen_out_31);

%inizialization
Imagen_in_bayer = Imagen_out_1;
Imagen_out_bayer = Imagen_out_1;
%red modification
Imagen_in_bayer(:,1:2:2048,1) = 0;
Imagen_in_bayer(2:2:2048,:,1) = 0;
%blue modification
Imagen_in_bayer(:,2:2:2048,3) = 0;
Imagen_in_bayer(1:2:2048,:,3) = 0;
%green 139oolbox139tion
Imagen_in_bayer(1:2:2048,2:2:2048,2) = 0;
Imagen_in_bayer(2:2:2048,1:2:2048,2) = 0;
%Representation
% figure;
% imagesc(Imagen_in_bayer);
%
%%%% Definition of the planes

ImagenR = Imagen_in_bayer(:,:,1);
ImagenG = Imagen_in_bayer(:,:,2);
ImagenB = Imagen_in_bayer(:,:,3);

% figure;
```

```
% imagesc(ImagenR) ;
% figure ;
% imagesc(ImagenG) ;
% figure ;
% imagesc(ImagenB) ;
%
%%%% Matrix Generation

HR = [1 2 1 ;…
     2 4 2;…
     1 2 1];

HB = [1 2 1;…
     2 4 2;…
     1 2 1];

HG = [0 1 0;…
     1 4 1;…
     0 1 0];

HR = 0.25*HR;
HB = 0.25*HB;
HG = 0.25*HG;

%%%% bilinear interpolation of Green

% Green
ImagenGest = conv2(double(ImagenG),HG,'same');
ImagenGforRatioB = ImagenGest;
ImagenGforRatioR = ImagenGest;

ImagenGforRatioB(:,2:2:2048) = 0;
ImagenGforRatioB(1:2:2048,:) = 0;
ImagenBest = (double(ImagenB)-double(ImagenGforRatioB));
ImagenBest = conv2(double(ImagenBest),HB,'same');
ImagenBest = ImagenBest + ImagenGest ;

ImagenGforRatioR( :,1 :2 :2048) = 0 ;
ImagenGforRatioR(2 :2 :2048, :) = 0 ;
ImagenRest = (double(ImagenR)-double(ImagenGforRatioR)) ;
ImagenRest = conv2(double(ImagenRest),HR,'same') ;
ImagenRest = ImagenRest + ImagenGest ;

Imagen_out_bayer( :, :,1) = uint8(ImagenRest) ;
Imagen_out_bayer( :, :,2) = uint8(ImagenGest) ;
Imagen_out_bayer(:,:,3) = uint8(ImagenBest);

%%%% Representation;
figure;
imagesc(Imagen_in_bayer);


figure;
ax(1)=subplot(1,3,1);
imagesc(Imagen_out_1);
title('RGB');
ax(2)=subplot(1,3,2);
imagesc(Imagen_in_bayer);
title('Bayer Pattern');
ax(3)=subplot(1,3,3);
imagesc(Imagen_out_bayer);
title('Demosaicing RGB');
linkaxes(ax);
```

```
% Reobtain the real values
ImagenR = double(Imagen_out_1(:,:,1));
ImagenG = double(Imagen_out_1(:,:,2));
ImagenB = double(Imagen_out_1(:,:,3));

ImagenRest = round(ImagenRest);
ImagenGest = round(ImagenGest);
ImagenBest = round(ImagenBest);


errorR = ImagenR - ImagenRest;
errorG = ImagenG - ImagenGest;
errorB = ImagenB - ImagenBest;


figure;
ax(1)=subplot(1,3,1);
imagesc(abs(errorR));
title(,RED ERROR');
ax(2)=subplot(1,3,2);
imagesc(abs(errorG));
title('GREEN ERROR');
ax(3)=subplot(1,3,3);
imagesc(abs(errorB));
title('BLUE ERROR');
linkaxes(ax);


% calculate the mean-square error

% only with image processing
% mseR = immse(ImagenR,ImagenRest);
% mseG = immse(ImagenG,ImagenGest);
% mseB = immse(ImagenB,ImagenBest);

% manual mean-square-error
errorR = errorR.^2;
errorG = errorG.^2;
errorB = errorB.^2;

mseR = sum(sum(errorR))/(size(errorR,1)*size(errorR,2));
mseG = sum(sum(errorG))/(size(errorG,1)*size(errorG,2));
mseB = sum(sum(errorB))/(size(errorB,1)*size(errorB,2));
```

Annex 4: Tables of Results

1.ALGORITHM BAYER MATRIX WITH CCSDS 123

Set of Images of Toulouse – From a jpg format; 8 bits per pixel;

Images Toulouse – Pattern 1 – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.420000*	*1.014000*	*0.406000*	*3.935501*	*2.03*
Image 2	*1.420000*	*1.014000*	*0.406000*	*3.318268*	*2.41*
Image 3	*1.419000*	*1.014000*	*0.405000*	*3.857315*	*2.07*

Images Toulouse – Pattern 1 – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.513000*	*1.014000*	*0.499000*	*4.078476*	*1.96*
Image 2	*1.482000*	*1.014000*	*0.468000*	*3.371323*	*2.37*
Image 3	*1.419000*	*1.014000*	*0.405000*	*3.953598*	*2.02*

Images Toulouse – Pattern 1 – Predictor Full Column – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.309000*	*0.904000*	*0.405000*	*3.957489*	*2.02*
Image 2	*1.310000*	*0.904000*	*0.406000*	*3.380646*	*2.36*
Image 3	*1.325000*	*0.919000*	*0.406000*	*3.927444*	*2.03*

Images Toulouse – Pattern 2 – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.420000*	*1.014000*	*0.406000*	*3.699142*	*2.16*
Image 2	*1.420000*	*1.014000*	*0.406000*	*3.126007*	*2.55*
Image 3	*1.419000*	*1.014000*	*0.405000*	*3.694107*	*2.16*

Images Toulouse – Pattern 2 – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.498000*	*1.014000*	*0.484000*	*3.792938*	*2.11*
Image 2	*1.482000*	*1.014000*	*0.468000*	*3.132416*	*2.55*
Image 3	*1.499000*	*1.007000*	*0.492000*	*3.766998*	*2.02*

Images Toulouse – Pattern 2 – Predictor Full Column – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.309000*	*0.900000*	*0.409000*	*3.697235*	*2.16*
Image 2	*1.304000*	*0.903000*	*0.401000*	*3.129333*	*2.55*
Image 3	*1.316000*	*0.904000*	*0.412000*	*3.694336*	*2.16*

Images Toulouse – No Pattern Band 1 – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.285000*	*0.854000*	*0.431000*	5.650192	*2.03*
Image 2	*1.278000*	*0.856000*	*0.422000*	*5.112686*	*2.41*
Image 3	*1.292000*	*0.858000*	*0.434000*	*5.756714*	*2.07*

Images Toulouse – No Pattern Band 1 – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.364000*	*0.855000*	*0.509000*	*5.620712*	*1.96*
Image 2	*1.355000*	*0.855000*	*0.500000*	*5.084976*	*2.37*
Image 3	*1.368000*	*0.857000*	*0.511000*	*5.725861*	*2.02*

Images Toulouse – No Pattern Band 1 – Predictor Full Column – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.207000*	*0.773000*	*0.434000*	*5.669968*	*2.02*
Image 2	*1.195000*	*0.774000*	*0.421000*	*5.119278*	*2.36*
Image 3	*1.213000*	*0.777000*	*0.436000*	*5.785477*	*2.03*

Set of Images of Space – From a jpg format; 8 bits per pixel;

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.441000*	*1.020000*	*0.421000*	*4.104462*	*1.94*
Hubble 1	*1.446000*	*1.023000*	*0.423000*	*4.179901*	*1.91*
space	*1.450000*	*1.014000*	*0.436000*	*4.760483*	*1.68*
milky	*1.447000*	*1.012000*	*0.435000*	*4.595154*	*1.74*

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.518000*	*1.021000*	*0.497000*	*4.092560*	*1.95*
Hubble 1	*1.514000*	*1.022000*	*0.492000*	*4.135208*	*1.93*
space	*1.524000*	*1.014000*	*0.510000*	*4.785141*	*1.67*
milky	*1.521000*	*1.013000*	*0.508000*	*4.562347*	*1.75*

Images space – Pattern 2 – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.441000*	*1.020000*	*0.421000*	*3.963913*	*2.02*
Hubble 1	*1.437000*	*1.015000*	*0.422000*	*4.135086*	*1.93*
space	*1.442000*	*1.006000*	*0.436000*	*4.708786*	*1.70*
milky	*1.440000*	*1.004000*	*0.436000*	*4.600586*	*1.73*

Images space – Pattern 2– Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.441000*	*1.020000*	*0.421000*	*3.940155*	*1.94*
Hubble 1	*1.505000*	*1.013000*	*0.492000*	*4.092056*	*1.91*
space	*1.518000*	*1.007000*	*0.511000*	*4.719955*	*1.68*
milky	*1.447000*	*1.012000*	*0.435000*	*4.560822*	*1.74*

Set of Images of Space – From a jpg format; 12 bits per pixel + noise ;

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.494000*	*1.019000*	*0.475000*	*6.793793*	*1.76*
Hubble 1	*1.467000*	*1.022000*	*0.445000*	*5.279877*	*2.27*
space	*1.473000*	*1.016000*	*0.457000*	*6.553238*	*1.83*
milky	*1.447000*	*1.012000*	*0.435000*	*6.761948*	*1.77*

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.577000*	*1.018000*	*0.559000*	*6.897705*	*1.74*
Hubble 1	*1.542000*	*1.021000*	*0.521000*	*5.303955*	*2.26*
space	*1.545000*	*1.015000*	*0.530000*	*6.736496*	*1.78*
milky	*1.569000*	*1.016000*	*0.553000*	*6.899841*	*1.73*

Images space – Pattern 2 – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.494000*	*1.019000*	*0.475000*	*6.821472*	*1.75*
Hubble 1	*1.467000*	*1.022000*	*0.445000*	*5.281555*	*2.25*
space	*1.469000*	*1.011000*	*0.458000*	*6.464478*	*1.85*
milky	*1.481000*	*1.007000*	*0.474000*	*6.821472*	*1.75*

Images space – Pattern 2 – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.568000*	*1.011000*	*0.557000*	*6.816055*	*1.74*
Hubble 1	*1.536000*	*1.014000*	*0.522000*	*5.307007*	*2.26*
space	*1.539000*	*1.011000*	*0.528000*	*6.577942*	*1.78*
milky	*1.561000*	*1.008000*	*0.553000*	*6.950989*	*1.73*

Images space – No Pattern 1 Band – Predictor Full Neighbor – Encoder Block

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.336000*	*0.864000*	*0.472000*	*6.671463*	*1.75*
Hubble 1	*1.312000*	*0.868000*	*0.444000*	*5.281677*	*2.25*
space	*1.32100*	*0.868000*	*0.453000*	*6.095596*	*1.85*
milky	*1.336000*	*0.864000*	*0.472000*	*6.704041*	*1.75*

Images space – No Pattern 1 Band – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.427000*	*0.873000*	*0.554000*	*6.736176*	*1.74*
Hubble 1	*1.389000*	*0.869000*	*0.520000*	*5.297745*	*2.26*
space	*1.394000*	*0.867000*	*0.527000*	*6.327164*	*1.78*
milky	*1.424000*	*0.870000*	*0.554000*	*6.794510*	*1.73*

2.INTERPOLATION ALGORITHM WITH CCSDS 123

Green interpolation—No lost

Set of Images of Toulouse – From a jpg format; 8 bits per pixel;

Images Toulouse – Pattern 1 – Predictor Full Neighbor – Encoder Block—Green interpolation –Nolost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.451000*	*1.030000*	*0.421000*	*4.124756*	*1.94*
Image 2	*1.420000*	*1.014000*	*0.406000*	*3.546875*	*2.25*
Image 3	*1.449000*	*1.034000*	*0.415000*	*4.087891*	*1.96*

Images Toulouse – Pattern 2 – Predictor Full Neighbor – Encoder Block—Green interpolation–Nolost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.451000*	*1.030000*	*0.421000*	*4.212997*	*1.93*
Image 2	*1.435000*	*1.030000*	*0.405000*	*3.650970*	*2.19*
Image 3	*1.451000*	*1.030000*	*0.421000*	*4.165070*	*1.92*

Images Toulouse – No-Pattern –Predictor Full Neighbor – Encoder Block-Green interpolation–Nolost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.294000*	*0.889000*	*0.405000*	*3.925552*	*2.03*
Image 2	*1.294000*	*0.889000*	*0.405000*	*3.414719*	*2.34*
Image 3	*1.310000*	*0.905000*	*0.405000*	*3.863510*	*2.07*

Set of Images of Space – From a jpg format; 12 bits per pixel + noise ;

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Block—Green interpolation–Nolost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.529000*	*1.045000*	*0.484000*	*6.974747*	*1.72*
Hubble 1	*1.498000*	*1.045000*	*0.453000*	*5.440353*	*2.20*
space	*1.498000*	*1.045000*	*0.453000*	*6.756424*	*1.78*
milky	*1.497000*	*1.029000*	*0.468000*	*6.920212*	*1.73*

Images space – Pattern 2 – Predictor Full Neighbor – Encoder Block—Green interpolation–Nolost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.427000*	*0.873000*	*0.554000*	*6.736176*	*1.74*
Hubble 1	*1.389000*	*0.869000*	*0.520000*	*5.297745*	*2.26*
space	*1.394000*	*0.867000*	*0.527000*	*6.327164*	*1.78*
milky	*1.497000*	*1.014000*	*0.483000*	*6.979645*	*1.72*

Images space – No-Pattern –Predictor Full Neighbor – Encoder Block-Green interpolation–Nolost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.389000*	*0.905000*	*0.484000*	*6.876785*	*1.74*
Hubble 1	*1.342000*	*0.890000*	*0.452000*	*5.471466*	*2.19*
space	*1.359000*	*0.890000*	*0.469000*	*6.373016*	*1.88*
milky	*1.358000*	*0.889000*	*0.469000*	*6.853439*	*1.75*

Green interpolation—lost (0-1-2)

Set of Images of Toulouse – From a jpg format; 8 bits per pixel;

Images Toulouse – Pattern 1 – Predictor Full Neighbor – Encoder Block—Green interpolation –lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.435000*	*1.029000*	*0.406000*	*3.661041*	*2.18*
Image 2	*1.435000*	*1.029000*	*0.406000*	*3.046753*	*2.62*
Image 3	*1.436000*	*1.034000*	*0.406000*	*3.607285*	*2.21*

Images Toulouse – Pattern 2 – Predictor Full Neighbor – Encoder Block—Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.419000*	*1.014000*	*0.405000*	*3.741867*	*2.13*
Image 2	*1.419000*	*1.014000*	*0.405000*	*3.162567*	*2.52*
Image 3	*1.419000*	*1.014000*	*0.405000*	*3.680847*	*2.17*

Images Toulouse – No-Pattern –Predictor Full Neighbor – Encoder Block-Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.279000*	*0.889000*	*0.390000*	*3.056152*	*2.61*
Image 2	*1.263000*	*0.889000*	*0.374000*	*2.522354*	*3.17*
Image 3	*1.279000*	*0.889000*	*0.390000*	*2.970428*	*2.69*

Set of Images of Space – From a jpg format; 12 bits per pixel + noise ;

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Block—Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.497000*	*1.029000*	*0.468000*	*6.448181*	*1.86*
Hubble 1	*1.485000*	*1.045000*	*0.440000*	*4.932571*	*2.43*
space	*1.491000*	*1.040000*	*0.451000*	*6.256622*	*1.91*
milky	*1.502000*	*1.037000*	*0.465000*	*6.420929*	*1.86*

Images space – Pattern 2 – Predictor Full Neighbor – Encoder Block—Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.498000*	*1.032000*	*0.466000*	*6.410385*	*1.87*
Hubble 1	*1.474000*	*1.036000*	*0.438000*	*4.915054*	*2.44*
space	*1.481000*	*1.032000*	*0.449000*	*6.219299*	*1.92*
milky	*1.498000*	*1.031000*	*0.467000*	*6.482101*	*1.85*

Images space – No-Pattern –Predictor Full Neighbor – Encoder Block-Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.341000*	*0.889000*	*0.452000*	*5.863144*	*2.04*
Hubble 1	*1.326000*	*0.905000*	*0.421000*	*4.455948*	*2.69*
space	*1.342000*	*0.905000*	*0.437000*	*5.366638*	*2.23*
milky	*1.357000*	*0.905000*	*0.452000*	*5.858276*	*2.04*

Green-NonGreen interpolation—lost (0-1-2)

Set of Images of Toulouse – From a jpg format; 8 bits per pixel;

Images Toulouse – Pattern 1 – Predictor Full Neighbor – Encoder Block—Green-NonGreen interpolation –lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.452000*	*1.053000*	*0.399000*	*3.569077*	2,24
Image 2	*1.447000*	*1.054000*	*0.393000*	*2.963211*	2,69
Image 3	*1.454000*	*1.053000*	*0.401000*	*3.490295*	2,29

Images Toulouse – Pattern 2 – Predictor Full Neighbor – Encoder Block— Green-NonGreen interpolation—lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.452000*	*1.053000*	*0.399000*	*3.793365*	2,10
Image 2	*1.447000*	*1.054000*	*0.393000*	*3.196472*	2,50
Image 3	*1.454000*	*1.053000*	*0.401000*	*3.656281*	2,18

Images Toulouse – No-Pattern –Predictor Full Neighbor – Encoder Block- Green-NonGreen interpolation—lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Image 1	*1.279000*	*0.889000*	*0.390000*	*3.056152*	2,61
Image 2	*1.263000*	*0.889000*	*0.374000*	*2.522354*	3,17
Image 3	*1.279000*	*0.889000*	*0.390000*	*2.970428*	2,69

Set of Images of Space – From a jpg format; 12 bits per pixel + noise ;

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Block—Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.516000*	*1.056000*	*0.460000*	*6.105423*	1,96
Hubble 1	*1.492000*	*1.061000*	*0.431000*	*4.554947*	2,63
space	*1.502000*	*1.056000*	*0.446000*	*5.990631*	2,00
milky	*1.514000*	*1.055000*	*0.459000*	*6.092224*	1,96

Images space – Pattern 2 – Predictor Full Neighbor – Encoder Block—Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.510000*	*1.050000*	*0.460000*	*6.125366*	1,95
Hubble 1	*1.485000*	*1.045000*	*0.440000*	*4.564026*	2,62
space	*1.491000*	*1.040000*	*0.451000*	*6.019531*	1,99
milky	*1.507000*	*1.045000*	*0.462000*	*6.176819*	1,94

Images space – No-Pattern –Predictor Full Neighbor – Encoder Block-Green interpolation–lost

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Bits/Pixels*	*Compressed rate*
Hubble	*1.341000*	*0.889000*	*0.452000*	*5.863144*	2,04
Hubble 1	*1.326000*	*0.905000*	*0.421000*	*4.455948*	2,69
space	*1.342000*	*0.905000*	*0.437000*	*5.366638*	2,23
milky	*1.357000*	*0.905000*	*0.452000*	*5.858276*	2,04

3.VIDEO CODER ALGORITHM 1

Set of Images of Toulouse – From a jpg format; 8 bits per pixel;

Images Toulouse – Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Residual Generation duration(sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Sequence Image 1	*8.268000*	*0.468000*	*4.758000*	*3.042000*	*2.332050*	3,43
Sequence Image 2	*8.268000*	*0.468000*	*4.758000*	*3.042000*	*2.356140*	3,39
Sequence Image 3	*8.330000*	*0.483000*	*4.758000*	*3.089000*	*2.752649*	2,90

Set of Images of Space – From a jpg format; 12 bits per pixel + noise ;

Images space –Predictor Full Neighbor – Encoder Sample

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Sequence Hubble	---	---	---	*6.781860*	1,76
Sequence Hubble 1	---	---	---	*5.299469*	2,26
Sequence space	---	---	---	*6.588135*	1,82
Sequence milky	---	---	---	*6.770111*	1,77

4.SEGMENTED-FINAL ALGORITHM

Segmentation 8x8 – Mission Segmentation

Set of Images of Toulouse – From a jpg format; 8 bits per pixel;

Images Toulouse – Pattern 1 – Predictor Full Neighbor – Encoder Block – Segmentation 8x8

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Image 1	---	---	---	*4.040268*	1,98
Image 2	---	---	---	*3.475647*	2,30
Image 3	---	---	---	*3.975494*	2,01

Images Toulouse – Pattern 2 – Predictor Full Neighbor – Encoder Block – Segmentation 8x8

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Image 1	---	---	---	*4.171082*	1,91
Image 2	---	---	---	*3.704636*	2,15
Image 3	---	---	---	*4.133331*	1,93

Images Toulouse – No Pattern – Predictor Full Neighbor – Encoder Block – Segmentation 8x8

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Image 1	---	---	---	*5.535645*	1,44
Image 2	---	---	---	*5.201172*	1,53
Image 3	---	---	---	*5.683090*	1,40

Set of Images of Space – From a jpg format; 12 bits per pixel + noise ;

Images space – Pattern 1 – Predictor Full Neighbor – Encoder Block— Segmentation 8x8

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Hubble	---	---	---	*6.781860*	1,76
Hubble 1	---	---	---	*5.299469*	2,26
space	---	---	---	*6.588135*	1,82
milky	---	---	---	*6.770111*	1,77

Images space – Pattern 2 – Predictor Full Neighbor – Encoder Block— Segmentation 8x8

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Hubble	---	---	---	*6.694031*	1,79
Hubble 1	---	---	---	*5.283936*	2,27
space	---	---	---	*6.499557*	1,84
milky	---	---	---	*6.831573*	1,75

Images space – No-Pattern –Predictor Full Neighbor – Encoder Block- Segmentation 8x8

Images	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Hubble	---	---	---	*6.657608*	1,80
Hubble 1	---	---	---	*5.264893*	2,27
space	---	---	---	*6.103516*	1,96
milky	---	---	---	*6.676956*	1,79

Set of Images of Toulouse – From a jpg format; 8 bits per pixel;

Images Toulouse 1 – Pattern 1 – Predictor Full Neighbor – Encoder Block – Segmentation 2x2

Image1	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Block1	*0.359000*	*0.265000*	*0.094000*	*4.053467*	1,97
Block2	*0.359000*	*0.250000*	*0.109000*	*3.846619*	2,07
Block3	*0.359000*	*0.250000*	*0.109000*	*4.241821*	1,88
Block4	*0.359000*	*0.265000*	*0.094000*	*3.587646*	2,22
Total	---	---	---	*3.932388*	2,03

Images Toulouse 1 – Pattern 1 – Predictor Full Neighbor – Encoder Block – Segmentation 2x2

Image1	*Compression duration (sec)*	*Prediction duration (sec)*	*Encoding duration (sec)*	*Average Bits/Pixels*	*Compressed rate*
Block1	*0.094000*	*0.062000*	*0.032000*	*3.91284*	2,04
Block2	*0.093000*	*0.062000*	*0.031000*	*4.603271*	1,73
Block3	*0.094000*	*0.063000*	*0.031000*	*4.607910*	1,73
Block4	*0.093000*	*0.062000*	*0.031000*	*2.992676*	2,67
Block5	*0.078000*	*0.063000*	*0.015000*	*3.362793*	2,37
Block6	*0.078000*	*0.062000*	*0.016000*	*4.427002*	1,80
Block7	*0.094000*	*0.078000*	*0.016000*	*4.763672*	1,67
Block8	*0.093000*	*0.078000*	*0.015000*	*3.149658*	2,53
Block9	*0.094000*	*0.063000*	*0.031000*	*5.082764*	1,57
Block10	*0.093000*	*0.062000*	*0.031000*	*4.820068*	1,65
Block11	*0.094000*	*0.063000*	*0.031000*	*4.636230*	1,72
Block12	*0.094000*	*0.062000*	*0.032000*	*2.820313*	2,83
Block13	*0.093000*	*0.062000*	*0.031000*	*3.142822*	2,54
Block14	*0.078000*	*0.063000*	*0.015000*	*3.888184*	2,05
Block15	*0.078000*	*0.062000*	*0.016000*	*4.717285*	1,69
Block16	*0.078000*	*0.063000*	*0.015000*	*2.383789*	3,35
Total	---	---	---	*3.956955*	2,02

Annex 5: XSDB Commands

1.CONNECT

Usage

```
connect [Options]
```

Options

Option	Description
-host <host name/ip>	Name/IP address of the host machine
-port <port num>	TCP port number
-url <url>	URL description of hw_server
-list	List open connections
-set <channel-id>	Set active connection
-new	Create a new connection, even one exist to the same url

Example

- Connect to hw_server on host localhost and port 3121

```
connect -host localhost -port 3121
```

- Identical to previous example

```
connect -url tcp:localhost:3121
```

2.ADD BREAKPOINT

Usage

bpadd <Options>

Options

Option	Description
-addr <breakpoint-address>	Specify the address at which the Breakpoint should be set.
-file <file-name>	Specify the file-name in which the Breakpoint should be set.
-line <line-number>	Specify the line-number within the file, where Breakpoint should be set.
-type <breakpoint-type>	Specify the Breakpoint type. Type can be one of the values below: • auto - Auto - Breakpoint type is chosen by hw_server. This is the default type • hw - Hardware Breakpoint • sw - Software Breakpoint
-mode <breakpoint-mode>	Specify the access mode that will trigger the breakpoint. Mode can be a bitwise OR of the values below: • 0x1 - Triggered by a read from the breakpoint location. • 0x2 - Triggered by a write to the breakpoint location. • 0x4 - Triggered by an instruction execution at the breakpoint location. This is the default for Line and Address breakpoints • 0x8 - Triggered by a data change (not an explicit write) at the breakpoint location.

Example

- Set a Breakpoint at address 0x100000. Breakpoint type is chosen by hw_server.

```
bpadd -addr 0x100000
```

- Set a Hardware Breakpoint at test.c:23

```
bpadd -file test.c -line 23 -type hw
```

- Set a Read_Write Watchpoint on variable fooVar

```
bpadd -addr &fooVar -type hw -mode 0x3
```

3.REMOVE BREAKPOINT

Usage

```
bpremove <id-list>
```

Options

Option	Description
<id-list>	List of breakpoint IDs, which are returned by the bpadd command. Breakpoint IDs can also be obtained by using the bplist command.

Example

- Remove Breakpoint 0

  ```
  bpremove 0
  ```

- Remove Breakpoints 1 and 2

  ```
  bpremove 1 2
  ```

4.DOWNLOAD

Download elf/binary files to the active target

Usage

dow [options] <file>

Options

Option	Description
[-clear] <file>	Download ELF file <file> to active target clear option can be used to clear non-loadable sections like bss, during ELF download
-data <file> <addr>	Download binary file<file> to active target address specified by <addr>

5.FPGA

Configure FPGA with a bitstream. FPGA device should be selected through the 'targets' command before running 'fpga' command.

Usage

```
fpga [options] <bitstream-file>
```

Options

Option	Description
-file <bitstream-file>	Specify file containing bitstream.
-partial	Configure FPGA without first clearing current configuration.

6. READ

Usage

```
mrd [options] <address> [num]
```

Options

Option	Description
-force	Overwrite access protection. By default accesses to reserved and invalid address ranges are blocked
-size <access-size>	Access size can be one of the values below: b = Bytes accesses h = Half-word accesses w = Word accesses Default access size is w. Address will be aligned to access-size before reading memory
-value	Return a Tcl list of values, instead of displaying the result on console
-bin	Return data read from the target in binary format
-file <file-name>	Write binary data read from the target to <file-name>
-arm-dap	Access ARM DAP address space DAP Address range is 0x0 - 0xfffc, where the higher 8 bits select an AP and lower 8 bits are the register address for that AP
-arm-ap <ap-num>	Access ARM MEM-AP address space ap-num selects a AP within the DAP address can be any address that is accessible over that AP

7. WRITE

Usage

- Write <num> words from list of <values> to active target memory address specified by <address>

  ```
  mwr [options] <address> <values> [num]
  ```

- If <num> words is not specified, all the <values> from the list are written sequentially from the address specified by <address>. If <num> is greater than the size of the <values> list, the last word in the list is filled at the remaining address locations
- Read <num> values from a binary file and write to active target memory address specified by <address>

  ```
  mwr [options] <address> [num]
  ```

- If <num> words is not specified, all the data from the file is written sequentially from the address specified by <address>.

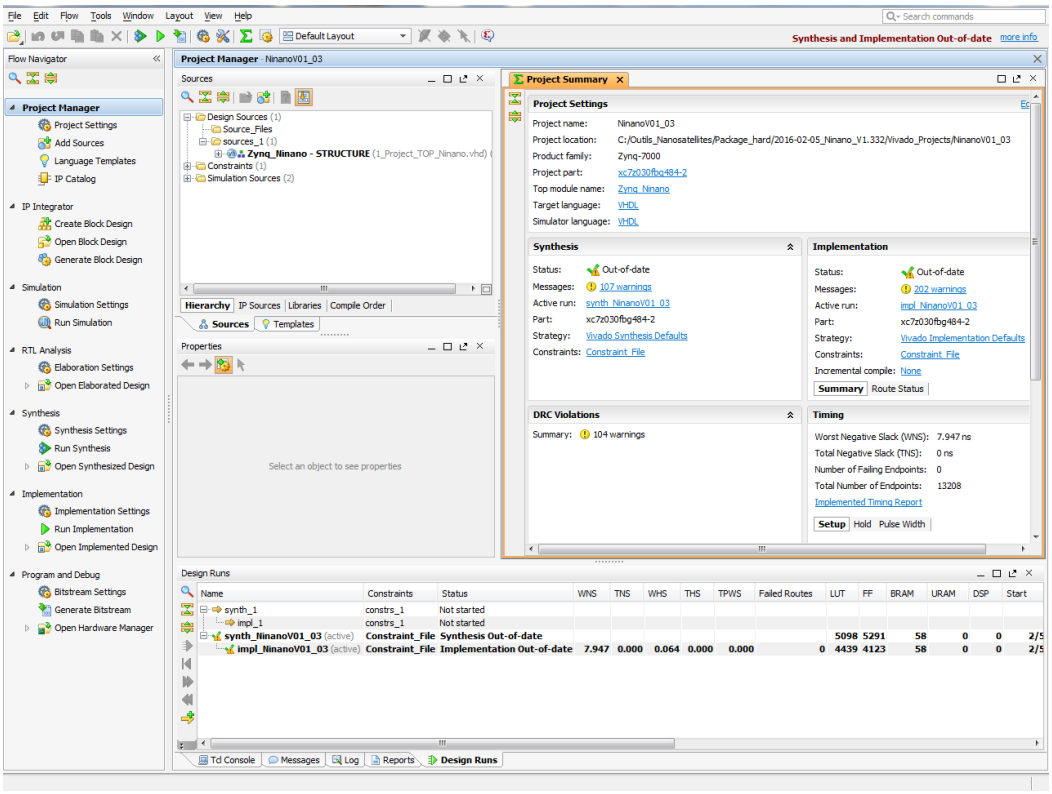
Options

Option	Description
-force	Overwrite access protection. By default accesses to reserved and invalid address ranges are blocked.
-size <access-size>	Access size can be one of the values below: b = Bytes accesses h = Half-word accesses w = Word accesses Default access size is w Address will be aligned to access-size before writing to memory
-bin	Read binary data from a file and write it to target address space.
-file <file-name>	File from which binary data is read before writing to target address space.
-arm-dap	Access ARM DAP address space. DAP Address range is 0x0 - 0xfffc, where the higher 8 bits select an AP and lower 8 bits are the register address for that AP.
-arm-ap <ap-num>	Access ARM MEM-AP address space. <ap-num> selects a AP within the DAP. address can be any address that is accessible over that AP

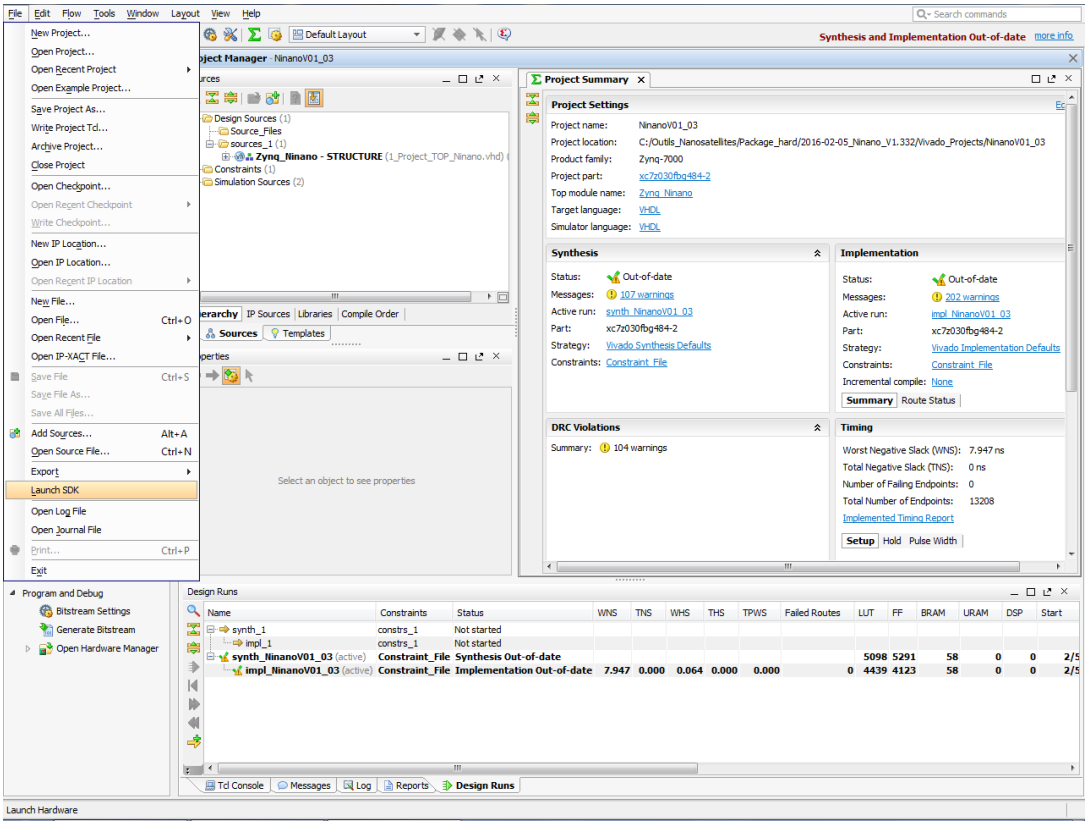
Annex 6: Execution Examples

1.SDK

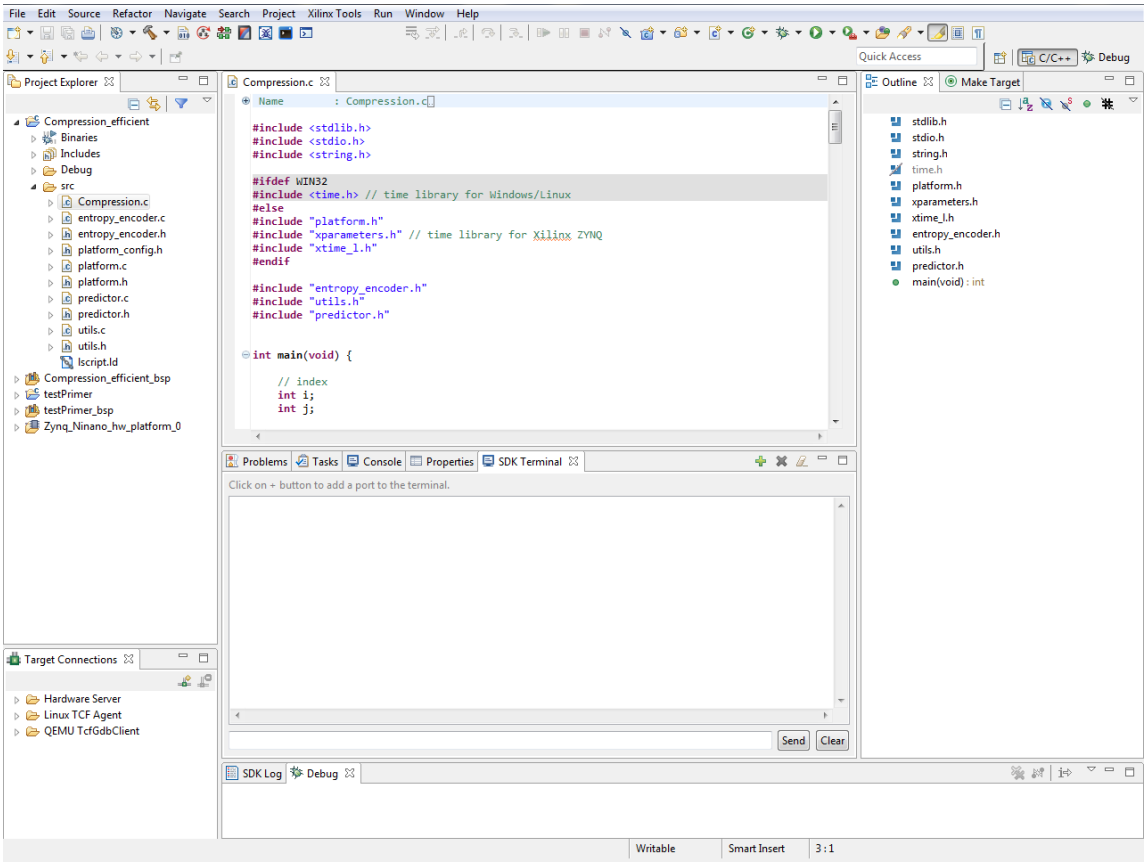
We start the Vivado Project



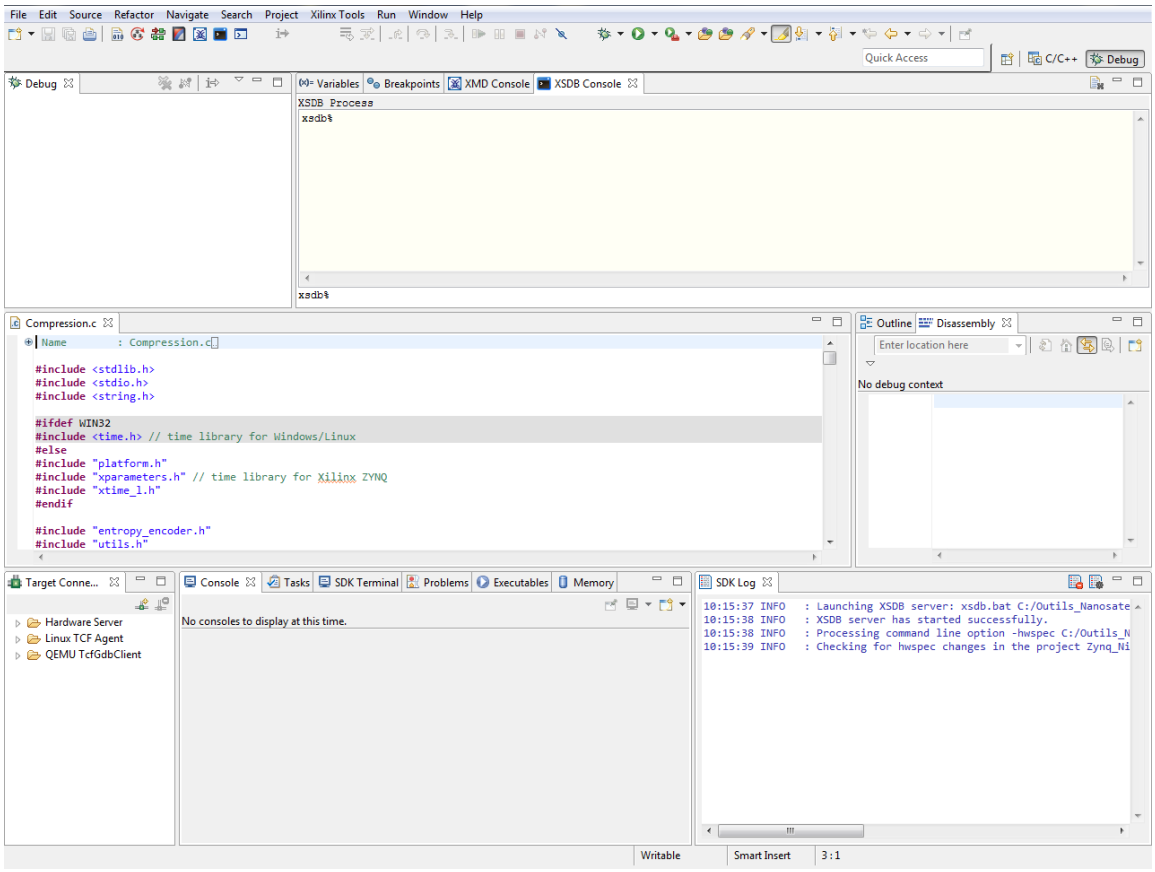
And we launch the SDK



A platform similar to eclipse will appear, we can develop our code on SDK platform



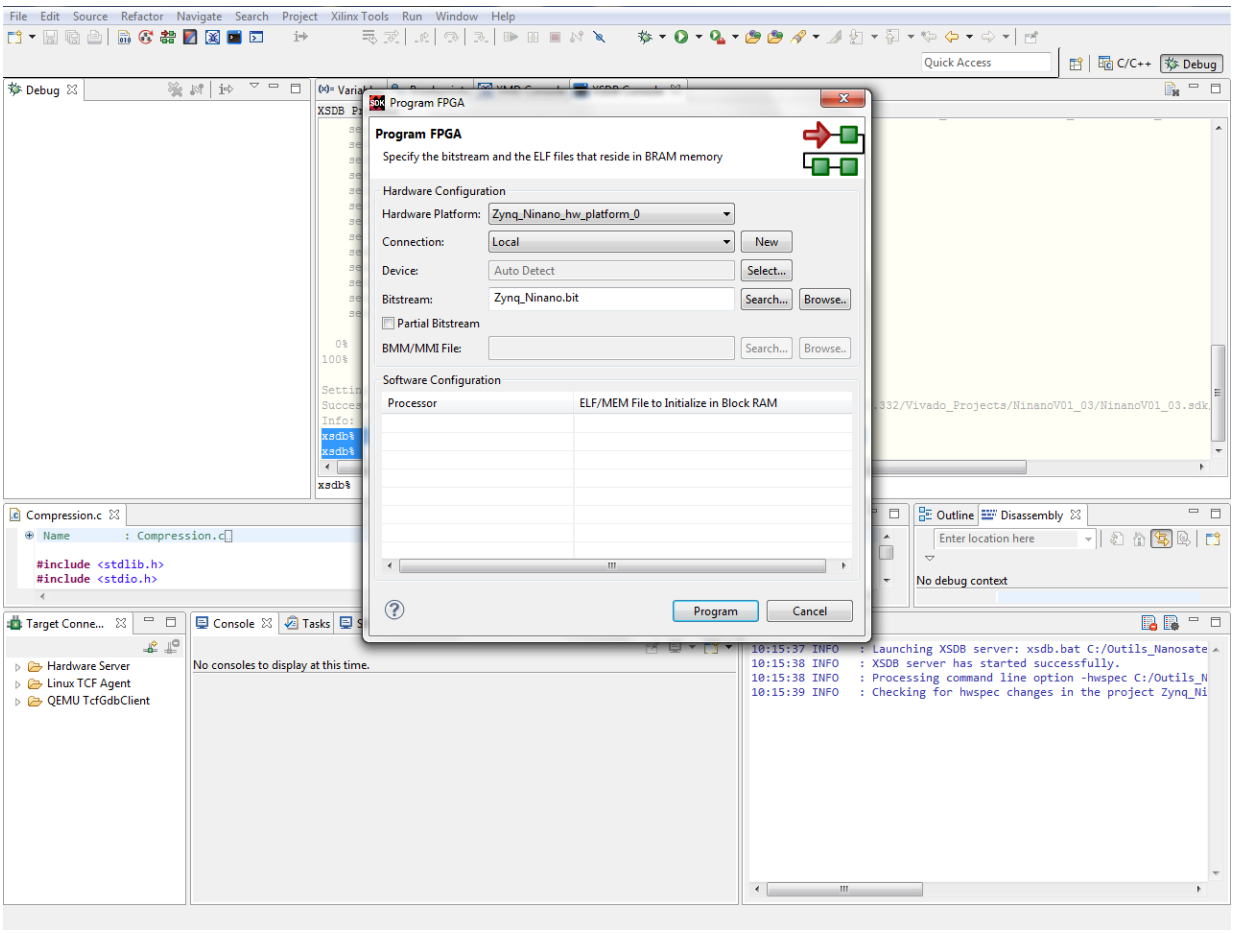
To load an image in memory we can do directly by the XSDB, to do that we must go to the debug tab and open the XSDB console

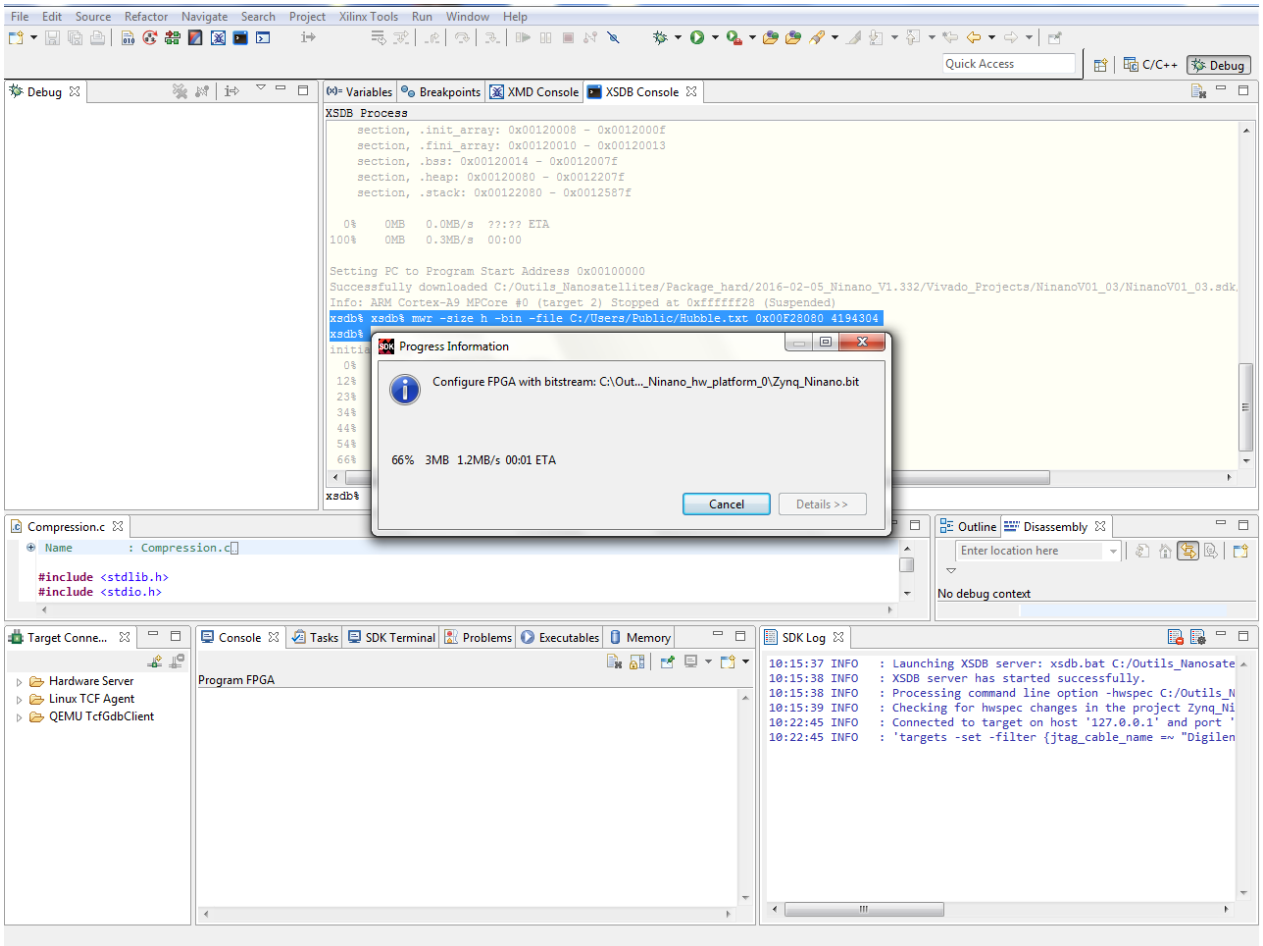


Inside the XSDB we must introduce the next code:

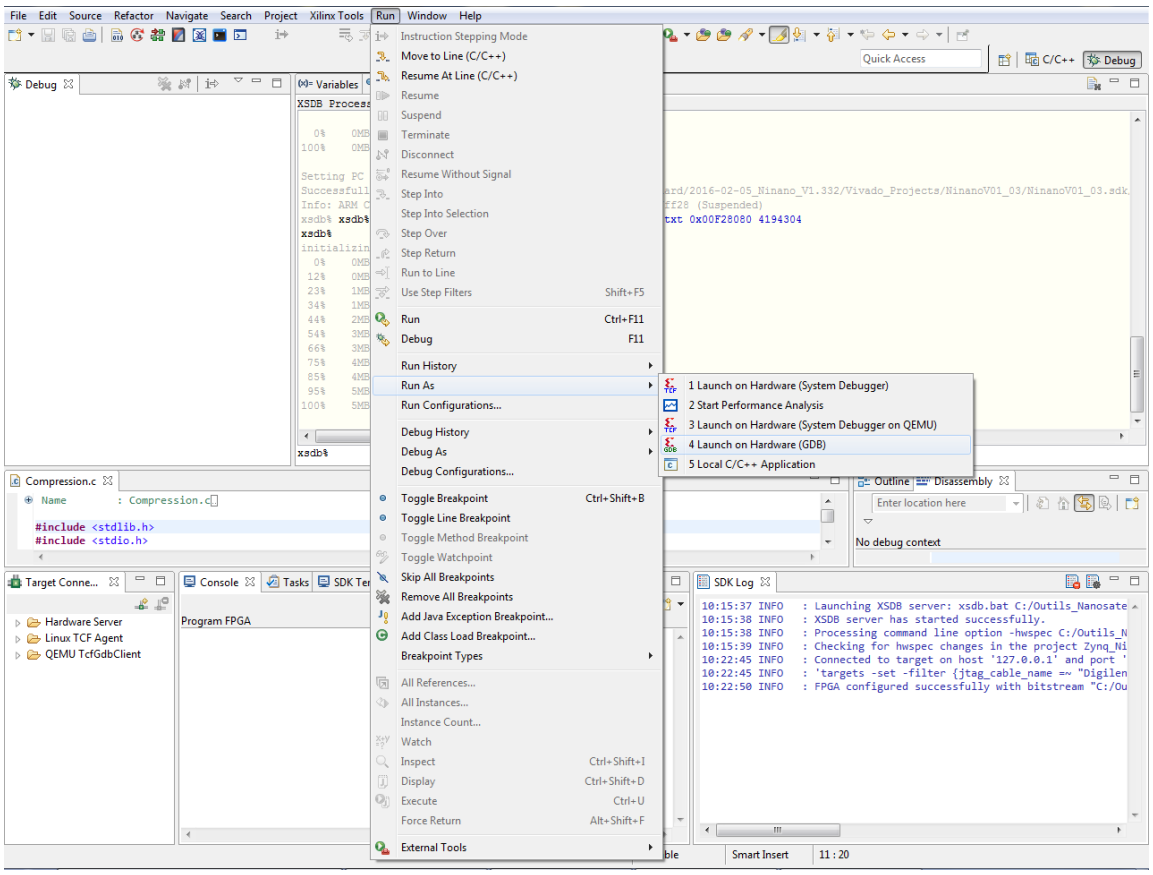
```
xsdb% connect
attempting to launch hw_server
****** Xilinx hw_server v2015.4
  **** Build date : Nov 17 2015-18:01:07
    ** Copyright 1986-1999, 2001-2015 Xilinx, Inc. All Rights Reserved.
INFO: hw_server application started
INFO: Use Ctrl-C to exit hw_server application
****** Xilinx hw_server v2015.4
  **** Build date : Nov 17 2015-18:01:07
    ** Copyright 1986-1999, 2001-2015 Xilinx, Inc. All Rights Reserved.
INFO: hw_server application started
INFO: Use Ctrl-C to exit hw_server application
INFO: To connect to this hw_server instance use url: TCP:127.0.0.1:3121
tcfchan#1
xsdb% targets
  1  APU
     2  ARM Cortex-A9 MPCore #0 (Running)
     3  ARM Cortex-A9 MPCore #1 (Running)
  4  xc7z030
xsdb% target 2
xsdb% source C:/Outils_Nanosatellites/Package_hard/2016-02-
05_Ninano_V1.332/Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/Zynq_Ninano_hw_pl
atform_0/ps7_init.tcl
xsdb% ps7_init
ps7_post_config*
xsdb% ps7_post_config
xsdb% dow C:/Outils_Nanosatellites/Package_hard/2016-02-
05_Ninano_V1.332/Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/testPrimer/Debug/
testPrimer.elf
Downloading Program -- C:/Outils_Nanosatellites/Package_hard/2016-02-
05_Ninano_V1.332/Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/testPrimer/Debug/
testPrimer.elf
      section, .text: 0x00100000 - 0x00117eb7
      section, .init: 0x00117eb8 - 0x00117ecf
      section, .fini: 0x00117ed0 - 0x00117ee7
      section, .rodata: 0x00117ee8 - 0x00118a3b
      section, .data: 0x00118a40 - 0x0011934b
      section, .eh_frame: 0x0011934c - 0x001193bf
      section, .mmu_tbl: 0x0011c000 - 0x0011ffff
      section, .ARM.exidx: 0x00120000 - 0x00120007
      section, .init_array: 0x00120008 - 0x0012000f
      section, .fini_array: 0x00120010 - 0x00120013
      section, .bss: 0x00120014 - 0x0012007f
      section, .heap: 0x00120080 - 0x0012207f
      section, .stack: 0x00122080 - 0x0012587f
  0%    0MB   0.0MB/s  ??:?? ETA
100%    0MB   0.3MB/s  00:00
Setting PC to Program Start Address 0x00100000
Successfully downloaded C:/Outils_Nanosatellites/Package_hard/2016-02-
05_Ninano_V1.332/Vivado_Projects/NinanoV01_03/NinanoV01_03.sdk/testPrimer/Debug/
testPrimer.elf
Info: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0xffffff28 (Suspended)
xsdb% mwr -size h -bin -file C:/Users/Public/Hubble.txt 0x10000000 4194304
```

After load the image, we can load our code at the FPGA. To do that we must click the bottom Program FPGA

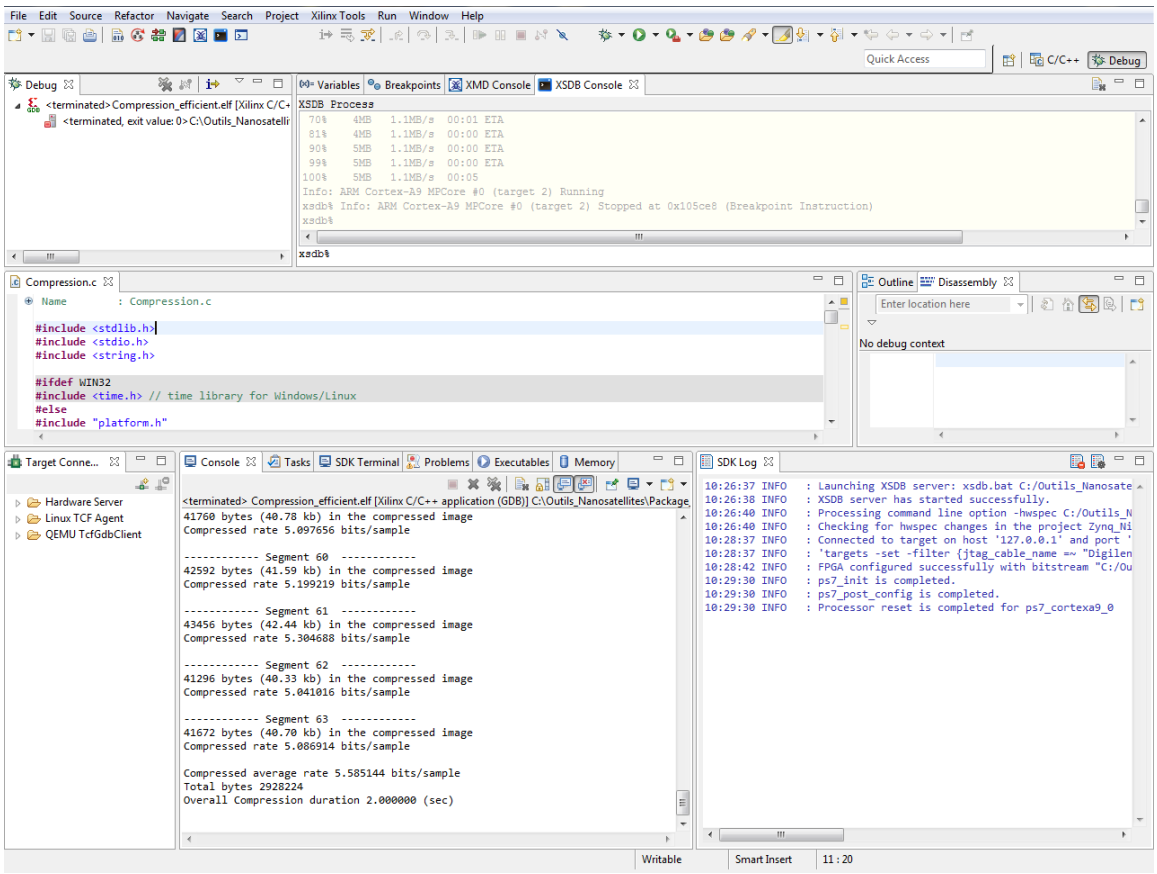




Finally to run the code we run as and launch on hardware.



Results will be shown at console



2.XTRATUM

The first step is developing and compiling the code inside the XtratuM environment, to do that we have use Aventurine, Linux Server. To compile the file we are going to create a "makefile", where we include all the compiling options. Since we are carrying out cross-compiling we must be carefully with libraries, compiling options….

Next figure show the "makefile" to generate the executable file – *resident_sw*

```
# XAL_PATH: path to the XTRATUM directory
XAL_PATH=../..
export XTRATUM_PATH = /lvcugen/TESTS/compression/ARM/XM_hello_world/2.0.4/xm2/xm
CONTAINER = container.bin
RSW = resident_sw

#all: entropy_encoder.o $(CONTAINER) $(RSW)
all: utils.o entropy_encoder.o predictor.o partition.o container.bin partition0.asm partition0.symb resident_sw resident_sw.asm resident_sw.symb


include $(XAL_PATH)/common/rules.mk

LOCAL_TARGET_CFLAGS = -I/lvcugen/TOOLS/compilateur/Xilinx-ARM-CC-4.8.3/lin/arm-xilinx-eabi/include
LOCAL_TARGET_CFLAGS += -I/lvcugen/TOOLS/compilateur/Xilinx-ARM-CC-4.8.3/lin/lib/gcc/arm-xilinx-eabi/4.8.3/include
LOCAL_TARGET_CFLAGS += -I$(XAL_PATH)/include
LOCAL_TARGET_CFLAGS += -I/lvcugen/ZYNQ//CONF/XM2/2.0.4/xm2/xm/include
LOCAL_TARGET_CFLAGS += -march=armv7-a -mcpu=cortex-a9 -mfpu=vfpv3 -nostdinc -nostdlib
LOCAL_TARGET_CFLAGS += -O2

LOCAL_TARGET_LDFLAGS = -L/lvcugen/TOOLS/compilateur/Xilinx-ARM-CC-4.8.3/lin/arm-xilinx-eabi/lib -lm
#LOCAL_TARGET_LDFLAGS += -T/lvcugen/ZYNQ/CONF/XM2/2.0.4/xm-src-2.0.4/user/xal/lib/loader.lds
LOCAL_TARGET_LDFLAGS += -u start -u xmImageHdr -T../../lib/loader.lds -L../../../../user/libxm -L../../lib -L/lvcugen/ZYNQ/CONF/XM2/2.0.4/xm2/xm/lib -
L/lvcugen/ZYNQ/CONF/XM2/2.0.4/xm2/xal/lib --start-group `/lvcugen/TOOLS/compilateur/Xilinx-ARM-CC-4.8.3/lin/bin/arm-xilinx-eabi-gcc -print-libgcc-file-
name -march=armv7-a -mcpu=cortex-a9 -mfpu=vfpv3` -lxm -lxal --end-group


TARGET_OBJDUMP = /lvcugen/TOOLS/compilateur/Xilinx-ARM-CC-4.8.3/lin/bin/arm-xilinx-eabi-objdump

# XMLCF: path to the XML configuration file
XMLCF=xm_cf.$(ARCH).xml

# PARTITIONS: partition files (xef format) composing the example
SRCS := $(sort $(wildcard *.c))
OBJS := $(patsubst %.c,%.o, $(SRCS)) $(patsubst %.S,%.o, $(ASRCS))

PARTITIONS=partition0.xef

#%.o: %.c
#	$(TARGET_CC) -c $< -o $@ $(LOCAL_TARGET_CFLAGS)

#entropy_encoder.o: utils.o
#	$(TARGET_CC) -c entropy_encoder.c

partition.o: partition.c
	$(TARGET_CC) -o partition.o -c partition.c $(LOCAL_TARGET_CFLAGS)

predictor.o: predictor.c utils.h
	$(TARGET_CC) -o predictor.o -c predictor.c $(LOCAL_TARGET_CFLAGS)

entropy_encoder.o: entropy_encoder.c utils.h
	$(TARGET_CC) -o entropy_encoder.o -c entropy_encoder.c $(LOCAL_TARGET_CFLAGS)

utils.o: utils.c
	$(TARGET_CC) -o utils.o -c utils.c $(LOCAL_TARGET_CFLAGS)

partition0: $(OBJS) $(XMLCF)
	$(TARGET_LD) -o $@ $(OBJS) $(LOCAL_TARGET_LDFLAGS) -Ttext=$(shell $(XPATHSTART) 0 $(XMLCF))


PACK_ARGS=-h $(XMCORE):xm_cf.xef.xmc \
	-p 0:partition0.xef

$(CONTAINER): $(PARTITIONS) xm_cf.xef.xmc
	$(XMPACK) check xm_cf.xef.xmc $(PACK_ARGS)
	$(XMPACK) build $(PACK_ARGS) $@

%.asm: %
	$(TARGET_OBJDUMP) -D $< > $@

%.symb: %
	$(TARGET_OBJDUMP) -x $< > $@
```

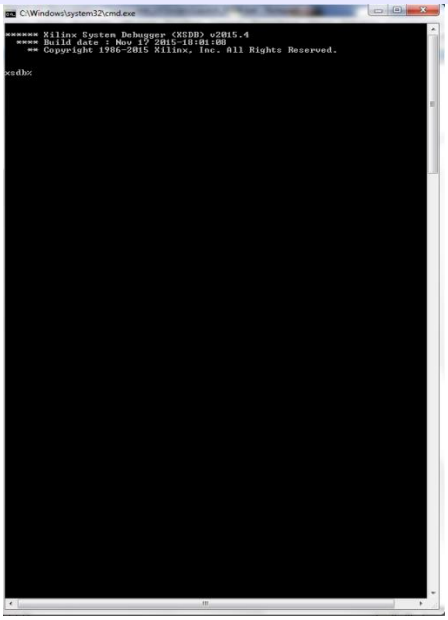
The XML file is shown in the next figure

```
<SystemDescription xmlns="http://www.xtratum.org/xm-arm-2.x" version="1.0.0" name="hello_world">
    <HwDescription>
        <MemoryLayout>
            <Region type="rom" start="0x0" size="1MB" />
            <Region type="sdram" start="0x00100000" size="1023MB" />
        </MemoryLayout>
        <ProcessorTable>
            <Processor id="0" frequency="80Mhz">
                <CyclicPlanTable>
                    <Plan id="0" majorFrame="1000ms">
                        <Slot id="0" start="0ms" duration="999ms" partitionId="0" />
                    </Plan>
                </CyclicPlanTable>
            </Processor>
        </ProcessorTable>
        <Devices>
            <Uart id="0" baudRate="230400" name="Uart" />
            <Uart id="1" baudRate="230400" name="Uart2" />
        </Devices>
    </HwDescription>
    <XMHypervisor console="Uart">
        <PhysicalMemoryArea size="20MB" />
    </XMHypervisor>
    <PartitionTable>
        <Partition id="0" name="Partition0" flags="boot fp" console="Uart">
            <PhysicalMemoryAreas>
                <Area start="0x0FFC0000" size="252KB"/>
                <Area start="0x10000000" size="8MB" flags="uncacheable read-only" mappedAt="0x10000000"/>
                <Area start="0x14000000" size="128KB"/>
                <Area start="0x14100000" size="6MB"/>
                <Area start="0x17200000" size="4KB"/>
            </PhysicalMemoryAreas>
        </Partition>
    </PartitionTable>
</SystemDescription>
```

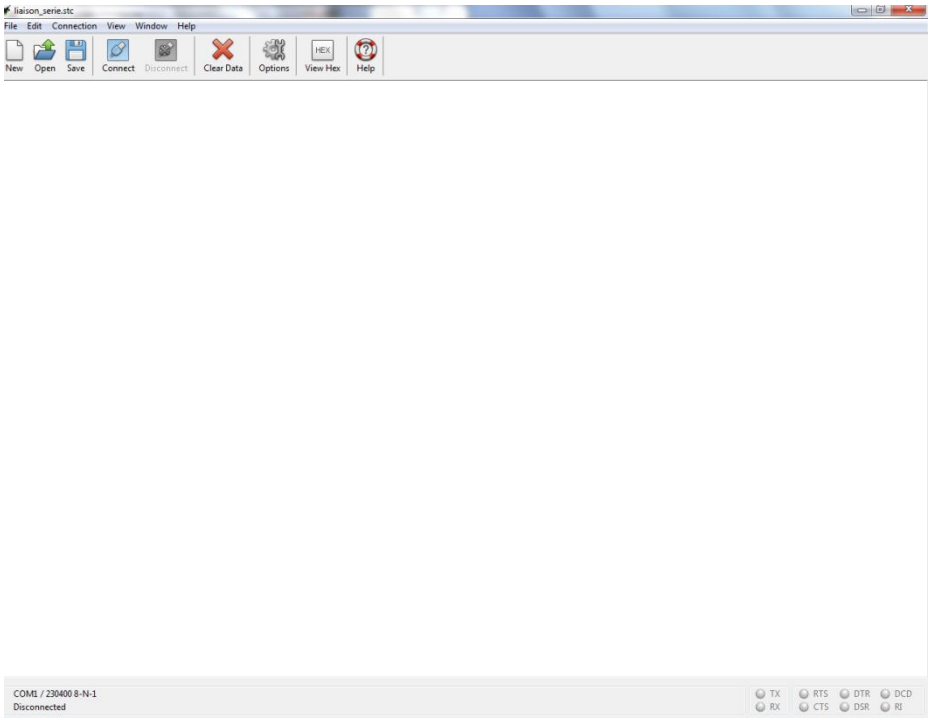
After that, we can compile through the command “make clean all”. Once we have already generated the *resident_sw,* we are going to connect the board and follow the next steps.

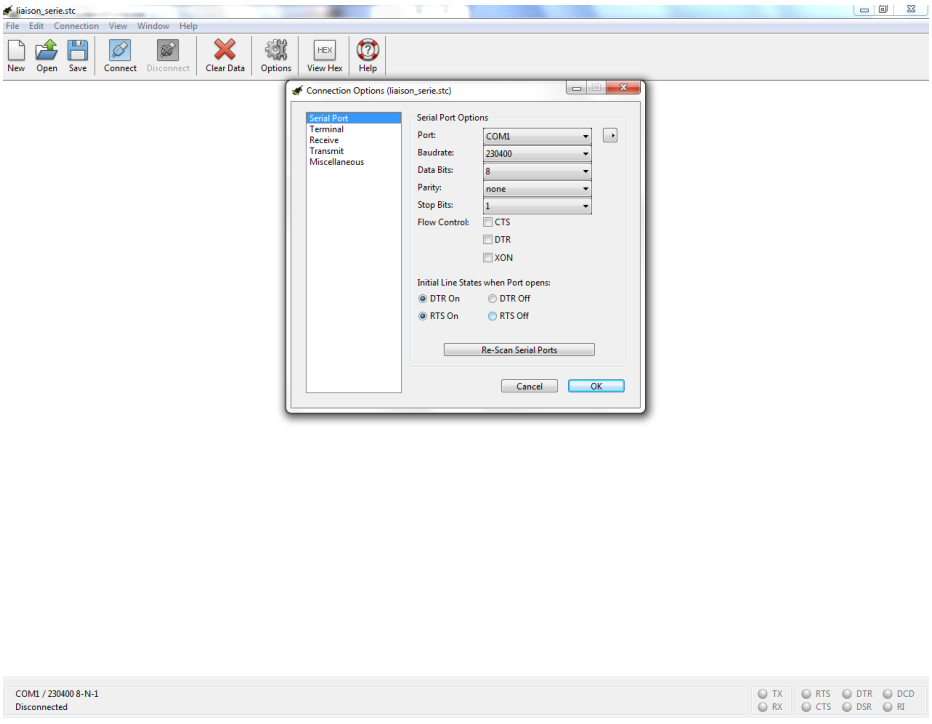
First, we create the next file. The goal of this file is to open the XSDB Console Line.

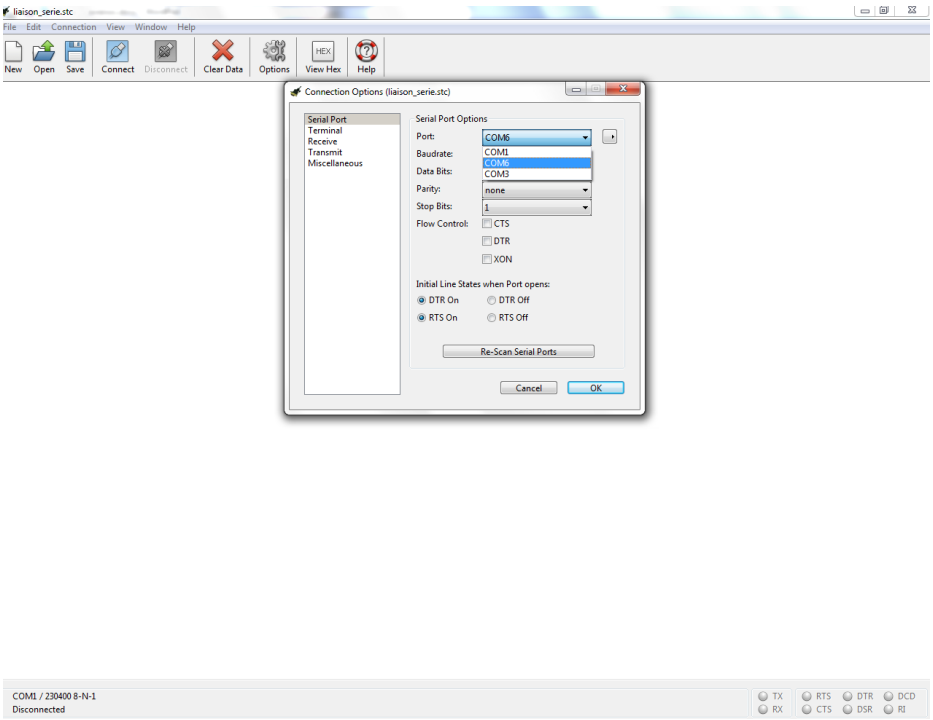
```
@echo off

set XMD=C:\Outils_Nanosatellites\Xilinx\SDK\2015.4\bin\xsdb.bat

%XMD%
```



To have the feedback from the board we are going to open the program CoolTerm. Inside the program, we must configure the UART; the port we are going to use is the number 6. Moreover, the rate has to be exactly the same between the XML file and the rate configuration inside the program. As in the XML we have decided 230400 as baud Rate, we must select that rate.

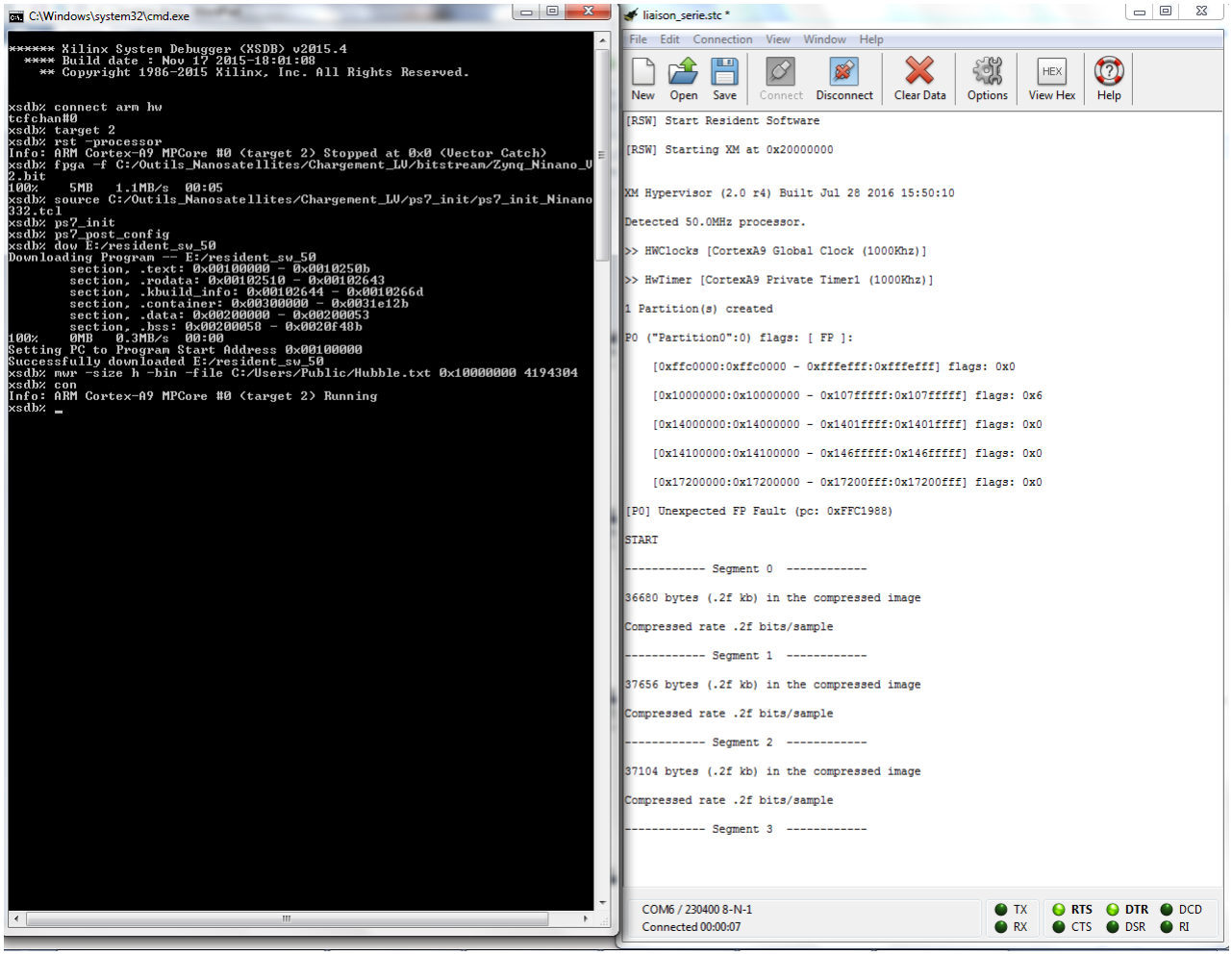




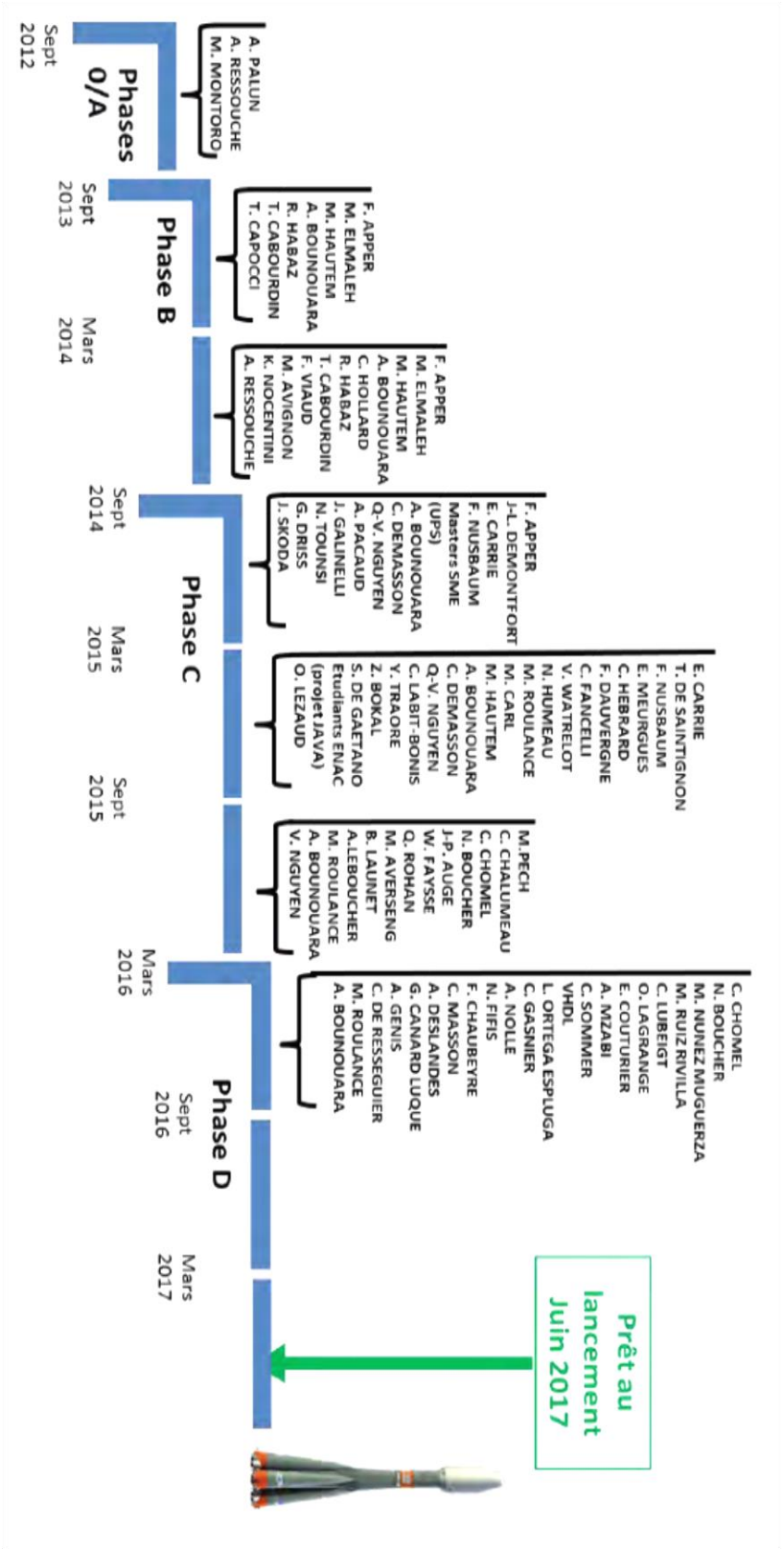


After that, we must load the image inside the memory zone and execute the code. To that, we write the next instructions.

Finally, we connect the CoolTermn and we write the instruction “con” for execute the program.


C:\Windows\system32\cmd.exe
****** Xilinx System Debugger (XSDB) v2015.4
**** Build date : Nov 17 2015-18:01:08
** Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.
xsdb% connect arm hw
tcfchan#0
xsdb% target 2
xsdb% rst -processor
Info: ARM Cortex-A9 MPCore #0 (target 2) Stopped at 0x0 (Vector Catch)
xsdb% fpga -f C:/Outils_Nanosatellites/Chargement_LV/bitstream/Zynq_Ninano_V2.bit
100% 5MB 1.1MB/s 00:05
xsdb% source C:/Outils_Nanosatellites/Chargement_LV/ps7_init/ps7_init_Ninano332.tcl
xsdb% ps7_init
xsdb% ps7_post_config
xsdb% dow E:/resident_sw_50
Downloading Program -- E:/resident_sw_50
section, .text: 0x00100000 - 0x0010250b
section, .rodata: 0x00102510 - 0x00102643
section, .kbuild_info: 0x00102644 - 0x0010266d
section, .container: 0x00300000 - 0x0031e12b
section, .data: 0x00200000 - 0x00200053
section, .bss: 0x00200058 - 0x0020f48b
100% 0MB 0.3MB/s 00:00
Setting PC to Program Start Address 0x00100000
Successfully downloaded E:/resident_sw_50
xsdb% mwr -size h -bin -file C:/Users/Public/Hubble.txt 0x10000000 4194304
xsdb% con
Info: ARM Cortex-A9 MPCore #0 (target 2) Running
xsdb%
liaison_serie.stc *
File Edit Connection View Window Help
New Open Save Connect Disconnect Clear Data Options View Hex Help
[RSW] Start Resident Software
[RSW] Starting XM at 0x20000000
XM Hypervisor (2.0 r4) Built Jul 28 2016 15:50:10
Detected 50.0MHz processor.
>> HWClocks [CortexA9 Global Clock (1000Khz)]
>> HwTimer [CortexA9 Private Timer1 (1000Khz)]
1 Partition(s) created
P0 ("Partition0":0) flags: [FP]:
[0xffc0000:0xffc0000 - 0xfffefff:0xfffefff] flags: 0x0
[0x10000000:0x10000000 - 0x107fffff:0x107fffff] flags: 0x6
[0x14000000:0x14000000 - 0x1401ffff:0x1401ffff] flags: 0x0
[0x14100000:0x14100000 - 0x146fffff:0x146fffff] flags: 0x0
[0x17200000:0x17200000 - 0x17200fff:0x17200fff] flags: 0x0
[P0] Unexpected FP Fault (pc: 0xFFC1988)
START
------------ Segment 0 ------------
36680 bytes (.2f kb) in the compressed image
Compressed rate .2f bits/sample
------------ Segment 1 ------------
37656 bytes (.2f kb) in the compressed image
Compressed rate .2f bits/sample
------------ Segment 2 ------------
37104 bytes (.2f kb) in the compressed image
Compressed rate .2f bits/sample
------------ Segment 3 ------------
COM6 / 230400 8-N-1
Connected 00:00:07
TX RX RTS CTS DTR DSR DCD RI

Annex 7: Project Students



Students March 2016 - September 2016