TIAGO CÉSAR KATCIPIS

INSERÇÃO DE METADADOS REFERENTES A DETECÇÃO DE PADRÕES NO H.264

Florianópolis

15 de Julho de 2011

TIAGO CÉSAR KATCIPIS

INSERÇÃO DE METADADOS REFERENTES A DETECÇÃO DE PADRÕES NO H.264

Trabalho de Conclusão de Curso submetido ao Programa de graduação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.

Orientador: Antônio Augusto Fröhlich

Coorientador: Paulo Jorge Câmara Pizarro

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis

15 de Julho de 2011

AGRADECIMENTOS

Primeiramente gostaria de agradecer à minha amada esposa e melhor amiga Stephanie, que além de me mostrar o real significado do amor e me apresentar a felicidade de uma maneira que eu nao conhecia, cooperou no desenvolvimento do trabalho desde o seu início, suportando amorosamente a minha ausência, se sobrecarregando com trabalho (inclusive o de revisar esse texto) permitindo que eu me dedicasse integralmente ao desenvolvimento do mesmo.

Aos meus pais agradeço por todo o esforço e sacrifício que fizeram para que eu pudesse chegar aqui, sem eles eu certamente não teria oportunidade de realizar este trabalho nem de viver todas as experiências maravilhosas que tive até hoje.

Ao professor Guto agradeço pela ajuda na escolha de um tema que além de interessante também é relacionado ao meu trabalho, e pela orientação em momentos críticos, onde o caminho a se seguir não era tão claro.

Pela oportunidade de desenvolver este trabalho em parceria com a Dígitro agradeço ao Rafael Pina que acreditou na proposta do trabalho e ao Paulo Pizarro, que além de acreditar no trabalho, co-orientou ele, fornecendo ajuda valiosa em decisões críticas ao longo do projeto e na formulação do texto final.

Agradeço ao Mateus Ludwich pela ajuda fundamental no desenvolvimento do trabalho, ajudando a escolher o melhor rumo a seguir em momentos críticos, esclarecendo dúvidas a respeito do software de referência do H.264, revisando o texto, revisando apresentações, apontando melhoramentos que podiam ser feitos. Certamente o resultado não teria sido o mesmo sem a sua ajuda. Pelas correções na apresentação e no texto final agradeço também ao Hugo Marcondes.

Ao Alexis Tourapis devo um agradecimento pela ajuda no entendimento de estruturas de dados e funções importantes do codificador do software de referência do H.264.

Acima de tudo agradeço a Jeová Deus, o dador de ”toda boa dádiva e todo presente perfeito”. (Tiago 1:17)

RESUMO

Para realizar o armazenamento e transmissão de vídeo tem se utilizado cada vez mais codificadores, dentre esses codificadores se destaca o MPEG 4 parte 10, também conhecido como H.264. Este trabalho integra um classificador Haar ao codificador de referência do padrão MPEG 4 parte 10 para realizar a detecção de objetos e explora o algoritmo de estimativa de movimento do codificador para realizar o *tracking* do objeto. Os objetos detectados e as informações de tracking são representados na forma de metadados e são transportados no bitstream do vídeo utilizando mensagens *Supplemental Enhancement Information*.

No decodificador de referência esses metadados são recuperados e apresentados de forma satisfatória a fim de realizar a constatação do bom funcionamento da implementação de detecção de objetos no codificador. Os testes realizados mostram que o codificador gerou vídeos em conformidade com o padrão MPEG 4 parte 10, junto com um desempenho computacional satisfatório. Os resultados obtidos no decodificador, ao recuperar os metadados e apresentá-los, também foram satisfatórios mostrando a viabilidade de construir um sistema de *tracking* de objetos embutido no codificador.

ABSTRACT

To store and transmit video the use of encoders has been increasing, MPEG 4 part 10, also known as H.264, stands out among these encoders. This work integrates the Haar classifier to the reference MPEG 4 part 10 encoder to detect objects and explores the motion estimation algorithm of the encoder to track the detected objects. Detected objects and tracking information are represented in the form of metadata and bitstream are transported in the video messages using Supplemental Enhancement Information.

In the reference decoder, metadata is retrieved and presented in order to satisfactorily carry out the verification of the proper functioning of the object detection implemented in the encoder. The tests showed that the generated video is in accordance with the MPEG 4 part 10, together with a satisfactory computational performance. The results obtained in the decoder, retrieving the metadata and presenting them, were also satisfactory, showing the viability of building a object tracking system embedded on the encoder.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

H.264 ..... *Padrão de compressão de vídeo*
MPEG-4 Parte 10 ..... *Padrão de compressão de vídeo, também conhecido como H.264*
H.263 ..... *Padrão de compressão de vídeo*
VLC ..... *Variable Length Coding*
VCL ..... *Video Coding Layer*
AVC ..... *Advanced Video Coding, também conhecido como H.264*
CCD ..... *Charged Coupled Device*
4:2:0(amostragem) *Método de amostragem: Componentes de crominância possuem metade da resolução horizontal e vertical do componente de luminância*
4:2:2(amostragem) *Método de amostragem: Componentes de crominância possuem metade da resolução horizontal do componente de luminância*
4:4:4(amostragem) *Método de amostragem: Componentes de crominância possuem mesma resolução que o componente de luminância*
CODEC ..... *Par de COder / DECoder, codificador e decodificador*
OpenCV ..... *Open Computer Vision*
JM ..... *Joint Model*
SEI ..... *Supplemental Enhancement Information*
MPEG ..... *Motion Picture Experts Group, um comitê da ISO/IEC*
MPEG-2 ..... *Padrão de compressão multimedia*
MPEG-2 TS ..... *MPEG-2 Transport Stream*
MPEG-4 ..... *Padrão de compressão multimedia*
ISO ..... *International Standards Organization*
ISO/IEC ... *International Standards Organization/International Electrotechnical Commission*
ITU-T ..... *International Telecommunication Union*
NAL ..... *Network Abstraction Layer*
NALU ..... *Network Abstraction Layer Unit*
RBSP ..... *Raw Byte Sequence Payload*
UUID ..... *Universally Unique Identifier*
QPel ..... *Quarter Pel refinement*
FPGA ..... *Field-programmable Gate Array*

SUMÁRIO

1 INTRODUÇÃO

Atualmente tem se tornado comum, em soluções de segurança, o uso de detecção de padrões como detecção facial, detecção de objetos específicos, cercas virtuais, alarmes, etc. Dispositivos de gravação de vídeo em alta definição estão se tornando cada vez mais acessíveis, estando presentes até mesmo em celulares, porém como é inviável dispor de longos trechos de vídeo em alta resolução sem compactação pois estes consomem um grande espaço de armazenamento e não é possível transmiti-los em larga escala com os meios de comunicação existentes, esses vídeos em alta definição são usualmente compactados.

A maior parte dos algoritmos de identificação de objetos e algoritmos biométricos trabalham com informações não codificadas, nesse caso, o processamento de múltiplos vídeos exigiria que esses vídeos fossem decodificados primeiro e então processados. Alguns algoritmos de identificação de objetos trabalham com informações codificadas, um exemplo é o detector de faces proposto em [1], mas na conclusão do mesmo é possível se observar que apesar do resultado ser satisfatório, o processamento em um identificador de objetos utilizando informações brutas foi superior (no caso a comparação foi realizada com o classificador Haar do OpenCV).

Considerando que a busca por objetos de interesse fosse realizada no vídeo compactado (não sendo necessário decodificar o vídeo), em um caso de uso como o de um aeroporto onde existem muitas câmeras de segurança, ainda seria necessário um grande poder computacional para realizar a busca de objetos de interesse em todos vídeos ao mesmo tempo, já que normalmente se espera que um sistema de segurança tenha um tempo de resposta rápido.

Dessa maneira é interessante obter a maior quantidade possível de metadados a respeito de objetos de interesse, na fonte do vídeo, de maneira integrada ao processo de codificação, reaproveitando ao máximo qualquer informação que o processo de codificação possa fornecer. Ao invés de analisar os vídeos, será realizada uma análise dos metadados. Se um vídeo muito longo não possui nenhum metadado, não será necessário analisá-lo. Como metadados pode-se citar a presença de um objeto de interesse (sua posição e tamanho), padrões de movimento (útil em cercas virtuais) ou o objeto de interesse em alta resolução não compactado (facilita o

processamento posterior desse objeto em um algoritmo biométrico).

Ao utilizar ao máximo as informações que o próprio codificador gera para realizar a identificação de padrões é interessante transportar os metadados gerados dentro do próprio bitstream do vídeo, isso facilita o desenvolvimento de um chip codificador na solução, pois não é necessário que os metadados encontrados sejam enviados a aplicação para serem transportados de outra maneira.

No presente trabalho será realizado um estudo da integração de um algoritmo de detecção de padrões com as informações de estimativa de movimento calculadas pelo codificador MPEG 4 parte 10, gerando um *tracker* de objetos, capaz de enviar também objetos de interesse não compactados.

1.1 OBJETIVOS

1.1.1 *OBJETIVO GERAL*

Modificar um codificador H.264 para torná-lo capaz de detectar e realizar o *tracking* de objetos, utilizando um detector de padrões e informações de estimativa de movimento, e inserir essas informações como metadados no próprio bitstream de vídeo.

1.1.2 *OBJETIVOS ESPECÍFICOS*

- Detectar e enviar objetos de interesse não comprimidos na forma de metadados, úteis para um pós-processamento.
- Realizar o *tracking* de objetos de interesse durante o processo de codificação do vídeo, utilizando informações de estimativa de movimento calculadas pelo próprio codificador para auxiliar o *tracking*.
- Inserir os metadados obtidos no processo de codificação diretamente no bitstream do vídeo H.264 sem alterar o vídeo e a conformidade dele com o padrão.
- Recuperar os metadados no processo de decodificação e apresentá-los de forma útil.

1.2 ESTRUTURA DO TRABALHO

Os capítulos que seguem estão organizados da seguinte maneira. No capítulo 2 apresenta-se a fundamentação teórica do trabalho, sendo dividida em 3 seções principais. A seção 2.1 traz

uma visão dos conceitos gerais de vídeo digital, da necessidade de compressão do mesmo e de compressão de vídeo em geral. A seção 2.2 traz conceitos gerais a respeito de codificação de vídeo. A seção 2.3, realiza um estudo do padrão de compressão de vídeo MPEG-4 parte 10, focando na inserção de metadados no bitstream. Na seção 2.4 é descrito em detalhes o algoritmo de detecção de objetos utilizado no trabalho.

O capítulo 3 descreve o desenvolvimento prático do trabalho, sendo dividido em 6 seções. A seção 3.1 fala sobre o uso prático das mensagens SEI *Unregistered Userdata* no software de referência, incluindo testes de inserção e recuperação das mensagens no bitstream. A seção 3.2 fala a respeito do módulo *extracted_metadata*, onde são definidos os tipos de metadados utilizados ao longo do trabalho e como eles são serializados e desserializados.

A seção 3.3 explica o módulo *metadata_extractor*, onde é realizada tanto a detecção como o *tracking* de objetos. A seção 3.4 explica em detalhes as alterações realizadas no codificar de referência para integrar o codificador com os módulos *extracted_metadata* e *metadata_extractor*. A seção 3.5 detalha as alterações realizadas no decodificador de referência para realizar a apresentação dos metadados inseridos no bitstream.

O capítulo 4 descreve e apresenta os resultados dos testes realizados no sistema proposto.

O capítulo 5 faz as considerações finais e conclui o trabalho.

2 REVISÃO TEÓRICA

2.1 VÍDEO DIGITAL

Como consta em [2], vídeo digital é a representação de uma cena do mundo real, amostrada espacialmente e temporalmente. Tipicamente uma cena é amostrada em um certo ponto no tempo para produzir um quadro, que representa a cena inteira naquele ponto no tempo. A amostragem é realizada em intervalos (ex. 1/25 ou 1/30 vezes por segundo).

Uma cena do mundo real costuma ser composta de múltiplos objetos, cada um com suas próprias características como forma, profundidade, textura e iluminação. A cor e o brilho de uma cena pode variar com diferentes graus de lisura ao longo da cena. As características de uma cena mais relevantes para o processamento e compressão de vídeo incluem características espaciais como variação de textura na cena, quantidade de forma dos objetos, cor, etc, e características temporais como o movimento de um determinado objeto, mudanças na iluminação e no movimento da câmera.

Usualmente uma cena é amostrada espacialmente sendo representada como uma grade retangular ou quadrada, o número de pontos amostrados no momento da captura influência a qualidade final da imagem. Essa quantidade de pontos amostrados normalmente é chamada de resolução da imagem.

Uma vídeo em movimento é formado por se tirar ”retratos” retangulares do sinal analógico que está sendo capturado em intervalos de tempo periódicos. Reproduzir uma série desses retratos ou quadros produz uma aparência de movimento. Uma taxa de amostragem alta fornece uma aparência de movimento mais suave, mas requer que mais amostras sejam capturadas e armazenadas.

2.1.1 *ESPAÇOS DE COR*

A maior parte das aplicações de vídeo precisam representar cores, portanto precisam de um mecanismo de captura e representação de informações de cor. Uma imagem monocromática

requer apenas um número para indicar o brilho ou luminância de cada amostra espacial. Porém imagens coloridas necessitam de no mínimo três números por pixel para representar a cor de forma precisa. O método escolhido para representar brilho, luminância ou luma e cor é descrito como espaço de cor.

Um dos espaços de cor mais conhecidos é o RGB, nesse formato uma amostra de imagem colorida é representada por três números que indicam a porção de vermelho, verde e azul, as 3 cores primárias. Combinar essas três cores pode produzir qualquer outra cor, dessa maneira o espaço de cor RGB consegue capturar e exibir imagens coloridas com sucesso. Capturar uma imagem RGB envolve filtrar os componentes vermelho, verde e azul de cada cena sem sensores separados (normalmente são utilizados CCDs na captura de imagens). Displays coloridos exibem uma imagem RGB por iluminarem o componente vermelho, verde e azul de cada pixel de acordo com a intensidade de cada componente de cor. De uma distancia normal, esses componentes separados se misturam e formam a aparência de uma cor de verdade.

Apesar de também ser muito utilizado, o espaço de cor RGB não é o mais bem adaptado a visão humana, que é mais sensível a luminância do que a cor, dessa maneira o RGB acaba ignorando informações importantes para a visão humana (luminância) e armazenando outras menos importantes (cores). É possível guardar uma cor de maneira mais eficiente por separar a informação de luminância da informação de cor e representar a informação de luminância (também chamada de luma) com um resolução mais alta.

O espaço de cor Y:Cr:Cb normalmente é utilizado para representar cores de forma eficiente, otimizando as informações para serem mais relevantes à visão humana. Nesse espaço de cor o Y representa a luminância e pode ser calculado como a média ponderada dos valores do canal R,G e B.

$$Y = krR + kgG + kbB \tag{2.1}$$

onde k são os pesos para cada canal.

A informação de cor pode ser representada como componentes de diferença de cor (crominância ou croma). Onde cada componente de crominância é a diferença entre R,G ou B e a luminância Y:

$$Cr = R - Y \tag{2.2}$$

$$Cb = B - Y \tag{2.3}$$

$$Cg = G - Y \tag{2.4}$$

Como Cr + Cb + Cg é uma constante somente dois dos valores de crominância precisam ser armazenados ou transmitidos, já que o terceiro componente pode ser calculado a partir dos outros dois. No Y:Cr:Cb somente o luma (Y) e o croma vermelho e azul são transmitidos. Os componentes de croma Cr e Cb podem ser representados com uma resolução menor que o luma já que o olho humano é menos sensível à cor do que a luz. Dessa maneira reduzimos a quantidade necessária para representar os componentes de crominância sem ter efeitos muito óbvios na qualidade visual.

O YCbCr possui três modos de amostragem suportados pelo H.264, esses modos são o 4:2:0, 4:2:2, 4:4:4. No 4:4:4 para cada quatro amostras de luminância temos quatro Cr e quatro Cb, no 4:2:2, também conhecido como YUY2, os componentes de crominância possuem a mesma resolução vertical que a luminância mas metade da resolução horizontal, 4:2:2 significa assim que para cada quatro amostras de luminância na direção horizontal existem duas amostras de Cr e duas de Cb. 4:2:2 normalmente é utilizado para reprodução de vídeos em alta qualidade de cor.

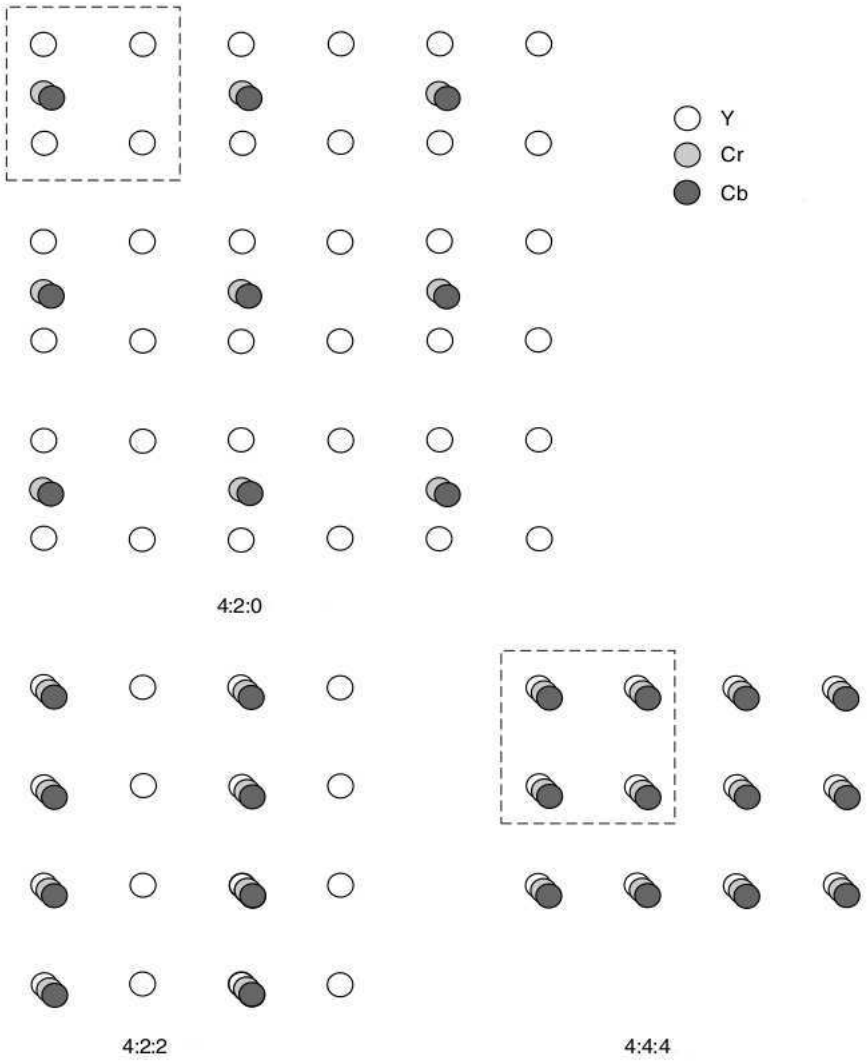

Figura 1: Padrões de amostragem 4:2:0, 4:2:2, 4:4:4 (progressivo).

O formato mais popular para codificação de vídeo, normalmente utilizado em aplicações como vídeo conferência, televisão digital e DVD é o 4:2:0, também conhecido como YV12, Cr e Cb possuem metade da resolução horizontal e vertical do Y. Cada componente de cor possui um quarto das amostras que existem no componente luma, dessa maneira um vídeo 4:2:0 necessita da metade das amostras necessárias em um vídeo 4:4:4 ou RGB. As diferenças entre cada um dos formatos pode ser melhor vista em []

2.2 CODIFICAÇÃO DE VÍDEO DIGITAL

Como consta em [2], compressão é o ato ou processo de compactar dados em um menor número de bits que o original. Compressão de vídeo é o processo de converter vídeo digital em um formato mais adequado para envio ou armazenamento. A compressão baseia-se na remoção

de redundância, removendo assim componentes que não são necessários para a reprodução do vídeo, normalmente utiliza-se uma abordagem de compactação com perda, buscando remover redundância temporal e espacial.

No domínio temporal existe uma alta correlação entre quadros que foram capturados em instantes similares de tempo, principalmente quando a taxa de amostragem é alta, existindo muita informação redundante que pode ser removida entre esses quadros. No domínio espacial existe uma correlação entre os pixels que estão perto um do outro, sendo mais uma fonte de informações redundantes que pode vir a ser comprimida.

Por isso normalmente os modelos de CODEC, como o H.264, MPEG-2 Vídeo, MPEG-4 Visual, H.263, VC-1, etc, utilizam predição e/ou compensação de movimento para remover redundância inerente tanto no próprio quadro (espacial) como entre quadros (temporal). Este trabalho é realizado no módulo de predição que tem por objetivo formar uma predição do quadro e então subtrair a predição deste mesmo quadro, gerando assim uma amostra residual do quadro. A predição pode ser formada a partir de quadros que já foram processados (nesse caso ela é temporal) ou a partir de amostras já processadas do mesmo quadro (nesse caso ela é espacial). Para realizar a predição o codificador deve utilizar apenas informações que estão disponíveis ao decodificador, ou seja, dados que já foram codificados e transmitidos, senão o decodificador não será capaz de reconstruir o quadro, pois ele necessita recriar a mesma predição feita no codificador para adicioná-la ao resíduo recebido e gerar o quadro.

Com exceção de regiões descobertas e mudanças drásticas de iluminação, a maior parte das diferenças que ocorrem entre os quadros de um vídeo se dá pelo movimento dos pixels entre os quadros, dessa maneira é possível calcular a trajetória de cada pixel entre sucessivos quadros do vídeo, gerando uma matriz de vetores de movimento para cada pixel. Porém realizar com precisão esse cálculo para cada pixel exige um esforço computacional muito grande, se tornando impraticável. Para contornar esse problema a estimativa e compensação de movimento é realizada em macro blocos e não por pixel.

2.2.1 *PREDIÇÃO DE UM MACROBLOCO UTILIZANDO COMPENSAÇÃO DE MOVIMENTO*

O macrobloco, que normalmente corresponde a uma região de 16 x 16 pixels de um quadro, é a unidade básica do processo de predição por compensação de movimento em vários padrões de codificação de vídeo como MPEG-1, MPEG-2, MPEG-3, MPEG-4 Visual, H.261, H.263 e H.264. Um macrobloco no formato de vídeo mais comum, que é o 4:2:0, é formado por uma região de 16 x 16 pixels, onde temos 256 amostras de luminância organizadas em 4 blocos de 8 x 8 pixels, 64 amostras de crominância vermelha em um bloco de 8 x 8 pixels e 64 amostras de

crominância azul também arranjadas como um bloco de 8 x 8. No H.264 cada quadro de vídeo é processado como um conjunto de macroblocos.

2.2.2 *ESTIMATIVA DE MOVIMENTO*

A estimativa de movimento de um macrobloco envolve,a partir de um quadro de referência previamente selecionado, encontrar uma região 16 x 16 neste quadro que seja o mais semelhante possível com o macrobloco atual. O quadro de referência deve ser um quadro que já foi codificado e deve ser anterior ou posterior ao quadro atual em relação a ordem de amostragem. A partir da posição do macrobloco corrente define-se uma área de busca por uma outra região que se assemelhe a ele no quadro de referência, dentro dessa área de busca procura-se a região de 16 x 16 que mais se assemelha com o macrobloco.

2.2.3 *COMPENSAÇÃO DE MOVIMENTO*

As amostras de luminância e crominância da região selecionada no quadro de referência são subtraídas do macrobloco corrente para produzir um macrobloco residual que é codificado e transmitido junto com um vetor de movimento descrevendo a posição da região selecionada em relação a posição do macrobloco corrente.

Existem variações no processo de estimativa e compensação de movimento. O quadro de referência pode ser um quadro anterior, posterior ou uma combinação da predição de dois ou mais quadros que já foram codificados. Se um quadro posterior ao corrente é selecionado, é necessário codificar esse quadro antes que o corrente, ou seja os quadros terão de ser codificados fora da ordem de apresentação. Quando ocorre uma mudança brusca entre o quadro de referência e o quadro corrente, por exemplo uma grande área foi descoberta, pode ser mais eficiente codificar o macrobloco sem realizar a compensação de movimento. Dessa maneira o codificador pode utilizar predição interna (espacial) ou a predição externa com compensação de movimento para cada macrobloco de acordo com a situação do quadro corrente.

2.3 MPEG 4 PARTE 10

O MPEG 4 Parte 10 (também conhecido como H.264 ou AVC) é um padrão de compressão de vídeo documentado e publicado por duas organizações padronizadoras, a ITU-T e a ISO/IEC [3]. Um codificador H.264 funciona realizando predição, transformação e codificação do resultado final, produzindo assim um bitstream. O decodificador realiza o processo complementar,

decodificando a informação, realizando uma transformada inversa e reconstrução dos quadros previamente codificados.

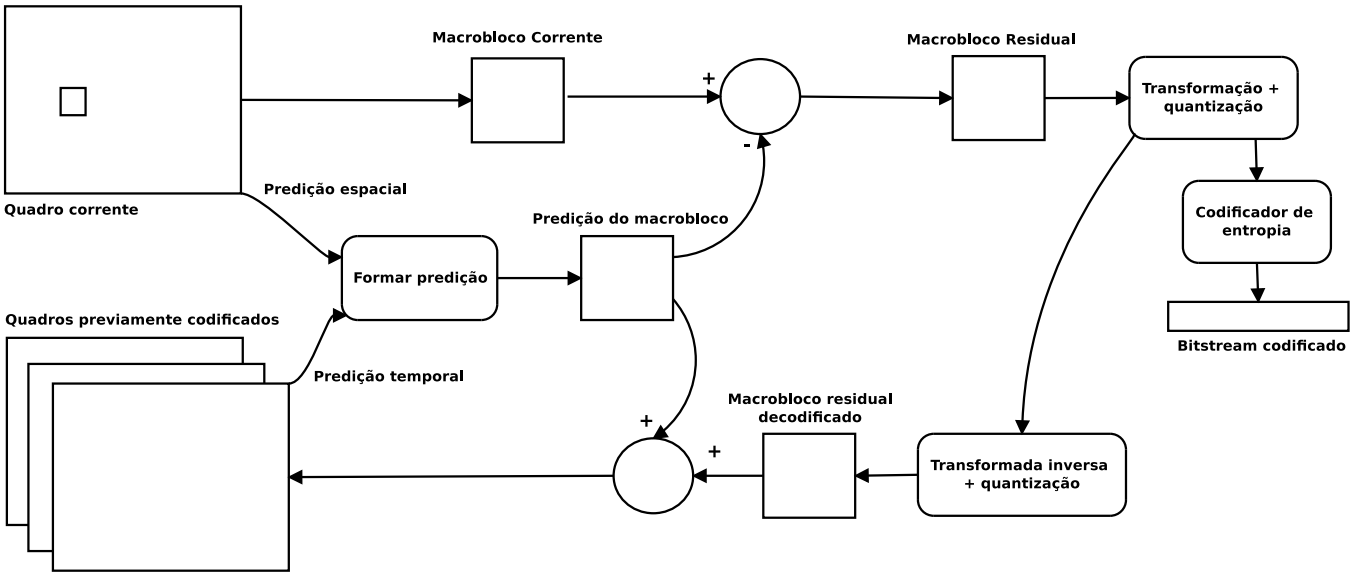

Figura 2: Típico codificador H.264

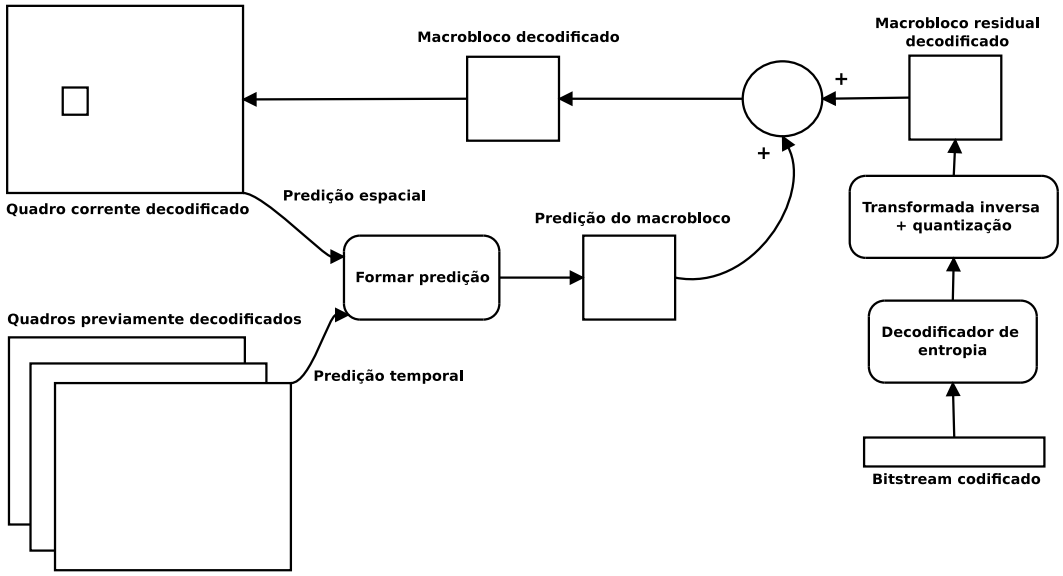

Figura 3: Típico decodificador H.264

Todas as informações produzidas pelo codificador tem de ser codificadas e comprimidas em um bitstream que possa ser armazenado/transmitido para ser decodificado posteriormente. Essas informações incluem:

- Coeficientes gerados pela transformação (quantizados).
- Informação que habilita o decoder a recriar a predição.

- Informação da estrutura dos dados comprimidos, e das ferramentas de compressão utilizadas no processo de codificação.
- Informação a respeito da sequência de vídeo como um todo.

Esses valores e parâmetros, também conhecidos como elementos de sintaxe, são convertidos em código binário utilizando codificação de tamanho variado (VLC) e/ou codificação aritmética. Cada um desses métodos produz uma representação binária eficiente e compacta das informações.

2.3.1 *SINTAXE GERAL*

H.264 provê uma sintaxe clara de como representar o vídeo comprimido e informações relacionadas ao vídeo, essa sintaxe é hierárquica. No nível mais alto, uma sequência H.264 consiste de uma série de ”pacotes” ou Network Abstraction Layer Units (NALUs).

Um NALU (Network Abstraction Layer Unit) contem um RBSP (Raw Byte Sequence Payload), que é uma sequência de bytes contendo os elementos da sintaxe. No H.264 os elementos da sintaxe são códigos binários de tamanho variado, portanto uma sequência de elementos de sintaxe dentro de um NALU não vai necessariamente possuir um número de bits divisivel por 8, bits com valor zero são adicionados ao final do RBSP para garantir que a quantidade de bits é dívisivel por 8 (esta técnica é chamada de *padding*).

Cada NALU consiste de um cabeçalho de 1 byte, seguido de um stream de bytes contendo informações de controle ou vídeo codificado. O cabeçalho indica o tipo do NALU e a ”importância” dele. NALUs utilizados como referência, por exemplo na predição de quadros futuros, são considerados como de alta prioridade, já que a perda de um desses poderia dificultar o processo de decodificação. Já os NALUs que não são utilizados como referência são considerados como tendo uma prioridade mais baixa. Esse tipo de informação pode ser utilizada para priorizar os NALUs mais importantes no momento da transmissão.

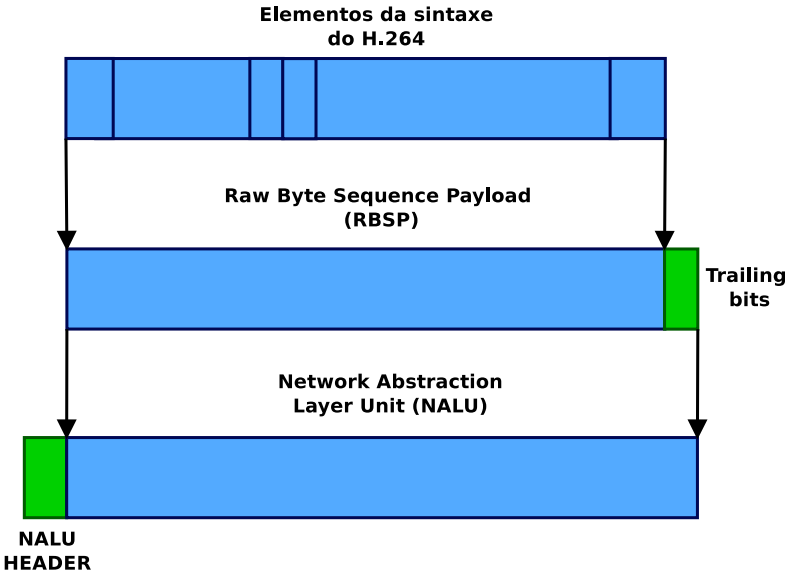

Figura 4: Encapsulamento de elementos de sintaxe do H.264 em um NALU

De acordo com [4] página 563, um NALU pode ser transmitido usando um protocolo de transporte, onde cada NALU se torna o payload do pacote (orientado a pacote), ou em um stream de bytes, onde cada NALU será transmitido sequencialmente em uma série de bytes (orientado a byte stream), utilizando como prefixo um código de inicio (start code) de 3 bytes indicando que está começando um novo NALU no bitstream.

Sempre que ocorre de uma sequência de 3 bytes ser similar ao start code, é inserido um emulation prevention byte para prevenir que o decoder confunda essa sequência de bytes com um start code. Quando se utiliza orientação a pacote é desnecessária a utilização do start code e do emulation prevention byte. A sintaxe completa de um NALU pode ser encontrada em [3] seção 7.3.1 e sua semântica na seção 7.4.1.

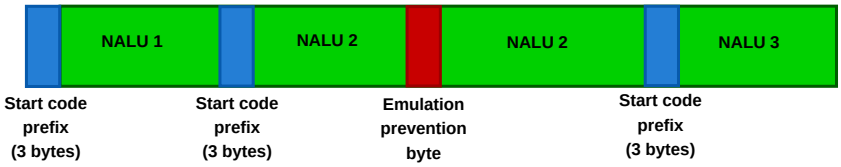

Figura 5: Exemplo de 3 NALUs sendo transmitidos/armazenados com orientação a bitstream

Figura 6: Exemplo de 3 NALUs sendo transmitidos/armazenados com orientação a pacotes

Como consta em [4] página 564, NALUs podem ser classificados como VCL (pertencem ao video coding layer) ou não-VCL NALUs (não pertencem a Video Coding Layer).

NALUs VCL possuem as amostras de vídeo (slices) enquanto que NALUs não-VCL possuem informações adicionais associadas ao vídeo como parâmetros (Parameter Sets) que fornecem informações de controle ao decodificador ou SEI (Supplemental Enhancement Information).

Os Parameter Sets existentes são os SPS (Sequence Parameter Sets), que podem ser aplicados a uma inteira sequência de vídeo, e os PPS (Picture Parameter Sets) que são aplicáveis somente a um ou mais quadros de uma determinada sequência, mas não todos.

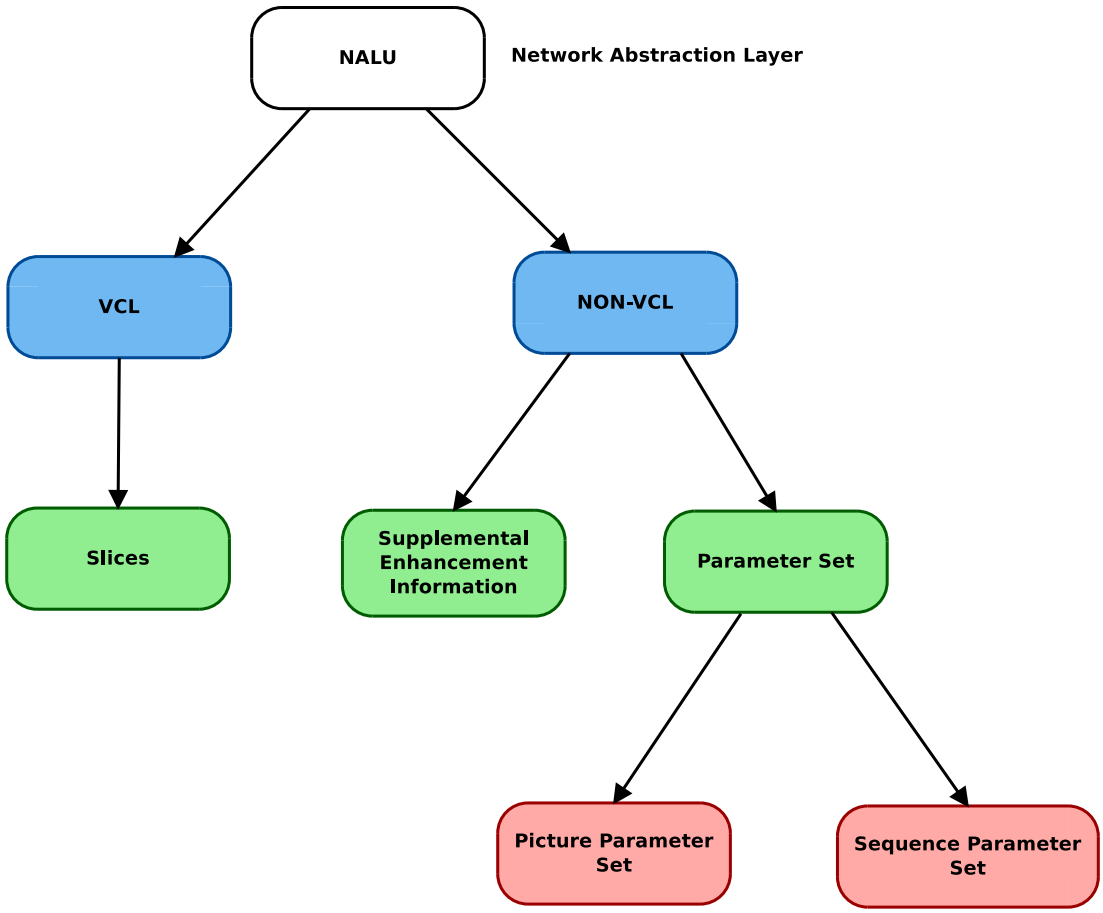

Figura 7: Possíveis classificações de um NALU

2.3.2 *SUPPLEMENTAL ENHANCEMENT INFORMATION*

Como consta em [3] cláusula 7.4.2.3, SEI (Supplemental Enhancement Information) NALUs contêm informações que não são essenciais ao processo de decodificação mas que podem ser utilizadas para auxiliar a decodificação. Um NALU do tipo SEI pode conter uma ou mais mensagens SEI dentro do seu RBSP. A sub-cláusula 7.3.2.3 define a sintaxe do RBSP de um NALU do tipo SEI, mais informações sobre SEI podem ser encontrados no anexo D da recomendação.

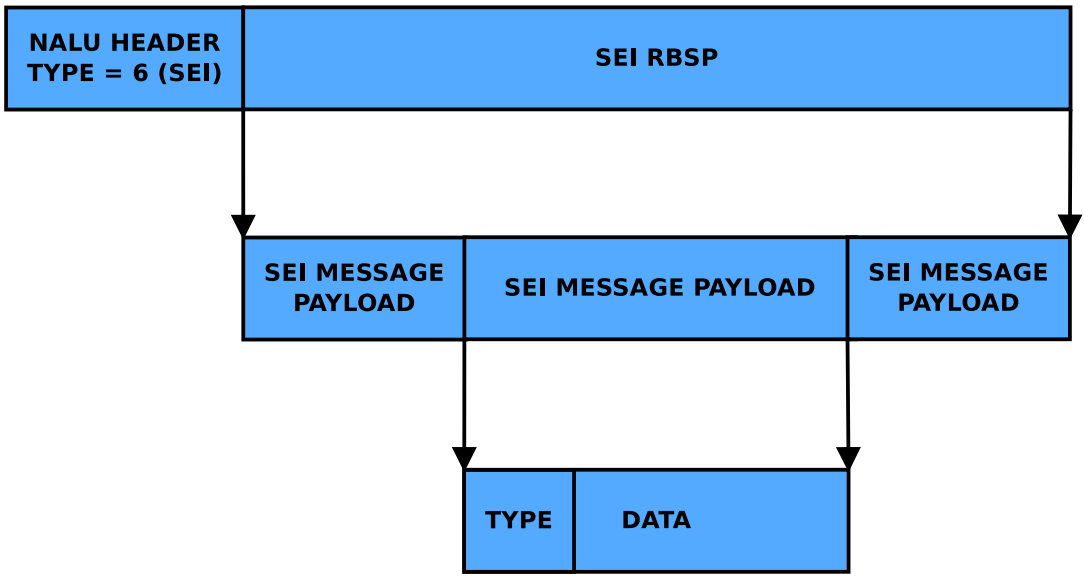

Figura 8: Exemplo de mensagens SEI encapsuladas em um SEI NALU

Não existe um limite máximo no tamanho de um SEI NALU, porém existem algumas limitações de buffer que podem resultar em um limite implícito no tamanho do SEI NALU (tabela A.1 do anexo A da [3] traz os limites que podem existir de acordo com o nível/perfil que o codificador implemente).

A sub-cláusula 7.3.2.3.1 da recomendação [3] define a sintaxe de uma mensagem SEI e a sub-cláusula 7.4.2.3.1 define a sua semântica, nestas sub-cláusulas pode se encontrar informações mais detalhadas de como o decodificador consegue obter o tamanho das mensagens SEI.

Uma mensagem SEI é composta basicamente do tipo do payload e o próprio payload. Normalmente as mensagens SEI são utilizadas para auxiliar o processo de decodificação, por isso existem vários tipos de mensagens pré-definidas.

Uma mensagem SEI do tipo *UserData* pode carregar qualquer tipo de dados de aplicação. Existem dois tipos de mensagem SEI *UserData*, a registrada e a não registrada, ambas podem possuir dados que não são definidos na recomendação [3]. Na mensagem SEI *UserData* registrada a mensagem deve ser precedida por um código registrado na ITU-T e a mensagem deve seguir a sintaxe e a semântica estipuladas por esse código informado.

Já a mensagem SEI *UserData* não registrada necessita de um UUID (Universally unique identifier, ou seja um identificador único para a mensagem) precedendo a mensagem e o conteúdo da mensagem não precisa obedecer nenhuma sintaxe ou semântica específica. A sintaxe do payload de uma mensagem SEI *UserData* não registrada pode ser encontrada em [3] na seção D.1.6 e a definição semântica na seção D.2.6.

Uma lista completa de todos os tipos de mensagens SEI se encontra em [3] na seção D.1.

2.4 CLASSIFICADOR HAAR - OPENCV

2.4.1 *INTRODUÇÃO*

OpenCV inclui diversas implementações de algoritmos de aprendizagem de máquina (uma lista completa dos algoritmos disponíveis encontra-se em [5] páginas 462 e 463), que é um sub-campo da inteligência artificial que visa transformar dados brutos em informação, tornando possível que a máquina responda a perguntas como: Que outra informação é similar a esta? Existe um carro nessa imagem?

Um desses algoritmos de aprendizagem de máquina é o classificador Haar, esse algoritmo é uma implementação da técnica conhecida como detector Viola-Jones [6] e que depois foi estendido por Rainer Lienhart e Jochen Maydt [7]. Ele é um classificador supervisionado (para mais informações sobre tipos de classificadores veja [5] páginas 460 e 461), ou seja, após treiná-lo é necessário apenas fornecer a imagem ao classificador e ele irá rotular ela como contendo ou não contendo o objeto de interesse.

2.4.2 *TEORIA DO CLASSIFICADOR VIOLA-JONES*

O detector Viola-Jones utiliza uma forma de AdaBoost ([5] página 497) mas o organiza como uma árvore, onde cada nodo desse é um classificador ”Multitree AdaBoosted”, desenvolvido para ter uma alta (digamos 99,9%) taxa de detecção (poucos falsos negativos), ás custas de uma baixa (perto de 50%) taxa de rejeição (grande quantidade de falsos positivos).

A árvore, que é organizada como uma cascata, funciona como uma cascata de rejeição, para cada nodo na cascata, um ”não pertence a classe” encerra a computação e o algoritmo responderá que não existe o objeto de pesquisa na região determinada para busca. A região só é classificada como possuindo um objeto de determinada classe se a computação ocorrer com sucesso por toda a cascata.

Em casos onde o objeto de interesse é raro (por exemplo uma face em uma foto), cascatas de rejeição aceleram o processo de busca por um objeto, já que a maior parte das regiões de busca será marcada como não possuindo o objeto rapidamente, sem ter de percorrer a maior parte da cascata. Em cada nodo da cascata de rejeição existe um conjunto de classificadores fracos, que são árvores de decisão. O conjunto de árvores em cada nodo é acelerado com a técnica de Boosting (mais informações sobre a técnica de Boosting podem ser encontradas em

[5] página 495).

Essas árvores normalmente possuem profundidade um, sendo também chamadas de *decision stumps*, são o caso mais simples de árvores de decisão o qual consiste de um único nodo de decisão e duas folhas. Cada ”decision stump” deve tomar apenas uma decisão, da seguinte forma: ”Está o valor V de um determinado recurso R acima ou abaixo de um limite L”; Nesse caso um ”sim” indica que o objeto de interesse está presente, um ”não” indica a sua ausência.

A quantidade de recursos Haar que o classificador Viola-Jones utiliza em cada classificador fraco pode ser configurado no momento do treinamento, normalmente se utiliza apenas um recurso (uma árvore com apenas uma divisão) ou no máximo 3 recursos. Incrementar esses classificadores iterativamente (técnica de Boosting) constrói um classificador que é a soma ponderada dos classificadores fracos. Ao analisar pela primeira vez o conjunto de dados, o classificador aprende o limite que melhor classifica a entrada. A técnica de Boosting utiliza os erros para realizar o calculo ponderado de w1. Cada vetor de recursos é reavaliado como tendo menos peso (se ele foi classificado corretamente) ou mais peso (se ele foi classificado incorretamente).

Pode parecer estranho diminuir o peso do vetor de dados quando ele é classificado corretamente, e aumentar o peso quando a classificação falha. A razão do treinamento ocorrer assim é que a técnica de Boosting tem como objetivo corrigir os pontos onde existem ”problemas” e ignorar os pontos que já funcionam bem. Depois que o nodo é ensinado, os dados sobreviventes da parte superior da cascata são usados para treinar o próximo nodo, e assim por diante.

Um detalhe importante é o uso de recursos Haar como entrada do algoritmo, em vez de dados brutos. Esses recursos Haar são basicamente um limite aplicado às somas e diferenças de regiões retangulares da imagem. Para acelerar o cálculo desses recursos é utilizada a técnica de imagem integral (para mais detalhes veja [5] página 182) que acelera o cálculo do valor de regiões retangulares e dessas mesmas regiões rotacionadas em 45 graus.

Viola e Jones organizaram cada grupo classificador boosted em nodos, formando uma cascata de rejeição, como mostrado na figura 9. Na figura , cada nodo F contem uma inteira cascata boosted de grupos de *decision stumps* (ou árvores) treinadas utilizando recursos Haar. Normalmente os nodos são ordenados do menos complexo ao mais complexo, assim o custo computacional é minimizado (os nodos mais simples são executados primeiro) ao analisar regiões da imagem que com certeza não possuem o objeto de interesse.

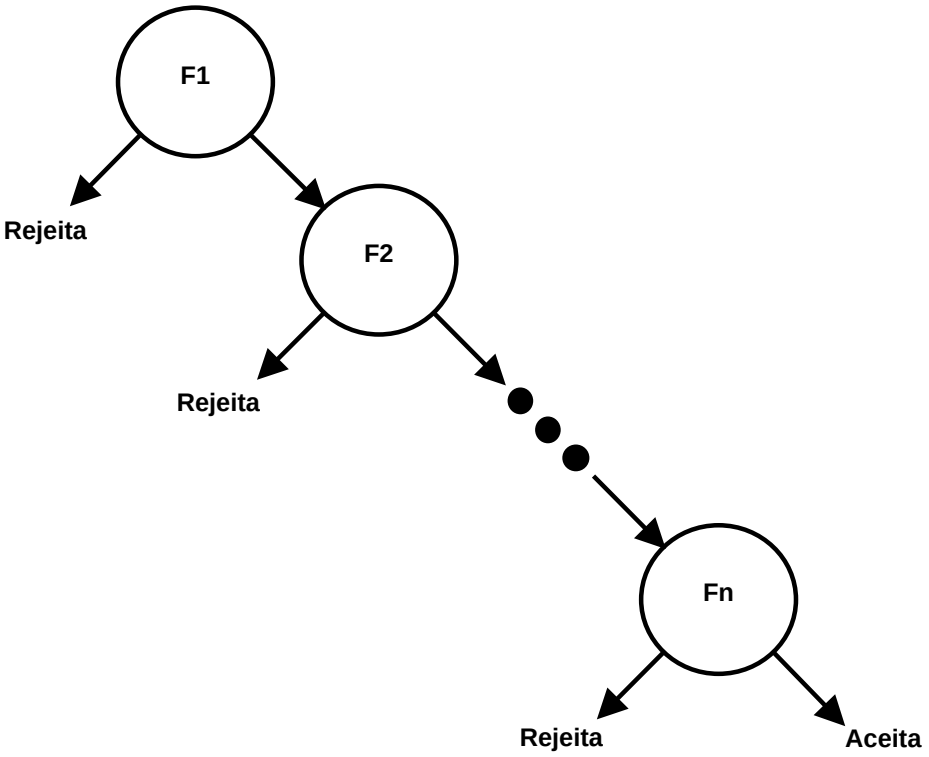

Figura 9: Cascata de rejeição utilizada no classificador Viola-Jones: Cada nodo representa um classificador ”Multitree AdaBoosted” treinado para raramente perder o objeto de interesse, rejeitando porém apenas uma pequena parte das regiões que não representam o objeto de interesse. Até chegar o final da cascata a maior parte das regiões que não representam o objeto de interesse já foram descartadas.

2.4.3 *USO DO CLASSIFICADOR*

O classificador Haar funciona bem em objetos como faces frontais, olhos, boca, traseira de carros, lateral de carros, frente de carros; Mas objetos como o perfil de um rosto (ou qualquer outro objeto em que o contorno seja a característica principal de identificação), já não funciona tão bem, principalmente porque essa perspectiva insere variações no padrão que os recursos em forma de bloco do Haar não conseguem lidar muito bem.

Por exemplo, para modelar a curva de um rosto visto de lado (seu contorno) é necessário inserir no modelo informações a respeito do fundo da imagem, para realizar essa modelagem de maneira adequada seria necessário um conjunto de treinamento muito maior, com diferentes fundos na imagem, ou rostos de lado com fundos diferentes do modelado seriam rejeitados. Ou seja, o contorno de um objeto é difícil de modelar nesse classificador porque os recursos Haar funcionam em bloco, dessa maneira o classificador é obrigado a aprender as variações do fundo da imagem que formam a borda do objeto.

Para treinar o classificador é necessário ter um grande conjunto de dados, com boa qua-

lidade, bem segmentado, para detecção de objetos rígidos onde o contorno não é a principal característica do objeto. O treinamento desse classificador é lento, porém graças a modelagem em cascata de rejeição ele é bem rápido para realizar a detecção do objeto em si.

Por grande conjunto de dados entende-se milhares de imagens com o objeto de interesse e dezenas de milhares de objetos sem o objeto de interesse. Por boa qualidade dos dados entende-se que os dados não devem conter o mesmo objeto em perspectivas diferentes, para cada perspectiva diferente é melhor treinar um classificador especifíco para aquela perspectiva.

Por bem segmentado entende-se imagens que representam corretamente o objeto sem variações desnecessárias, isso faria com que o classificador corrigisse essa variação fictícia, levando a resultados imprecisos. Por exemplo, ao treinar o classificador para detectar faces frontais, se a localização dos olhos no rosto variar demais (algo fora do normal) o classificador vai assumir que a localização dos olhos não é fixa e que é normal que os olhos possam estar em qualquer posição, isso degrada muito o desempenho já que o classificador precisa se ajustar a fatores que não existem nos dados reais.

3 PROJETO

Este capítulo descreve o desenvolvimento prático do projeto, que é composto da detecção e *tracking* de objetos dentro do software de referência, da utilização dos vetores calculados na estimativa de movimento para auxiliar no *tracking* de objetos e da inserção dos metadados extraídos no bitstream. Primeiro são apresentados resultados práticos da inserção de mensagens *SEI Userdata Unregistered* no bitstream de um vídeo codificado. Em seguida serão apresentados os novos módulos adicionados para realizar a extração e o armazenamento dos metadados e como eles foram integrados ao software de referência. Por fim será apresentada uma avaliação dos resultados obtidos.

3.1 INSERINDO USERDATA NO BITSTREAM

O H.264 possui uma sintaxe específica para o envio de mensagens não necessárias ao processo de decodificação, embutidas no bitstream de vídeo, são mensagens SEI (*Supplemental Enhanced Information*). Ao longo dessa seção será estudada a utilização prática das mensagens SEI do tipo *UserData* não registrada em pontos diferentes no processo de codificação.

3.1.1 *AMBIENTE DE TESTE*

Todos os testes foram realizados com o mesmo trecho de vídeo obtido a partir do site http://media.xiph.org/video/derf.

O vídeo possui a seguinte configuração:

- quadros por segundo = 29.97.
- total de quadros = 300.
- comprimento = 176 pixels.
- altura = 144 pixels.

- espaço de cor = YUV, 4:2:0.

Para testar a conformidade do bitstream gerado foi utilizado o decodificador de referência, após utilizar o decodificador de referência para transformar o bitstream H.264 em vídeo raw foi utilizado o Gstreamer para exibir o vídeo e constatar visualmente que o vídeo foi decodificado sem problemas.

O bitstream também foi testado diretamente no Gstreamer, pra verificar se o bitstream tocaria em um player usual (muitos players open source utilizam gstreamer como framework de multimedia).

Gstreamer utiliza internamente o ffmpeg para fazer o decoding do bitstream H.264.

3.1.2 *ENVIANDO UMA MENSAGEM SEI USERDATA*

Inicialmente foi utilizada a opção *GenerateSEIMessage* do próprio codificador, onde é possível ativar o envio de uma mensagem e configurar a mensagem que será enviada. Mas isso serve apenas para testes bem simples já que apenas uma mensagem é enviada ao longo de todo o bitstream de vídeo, e em forma de texto. Essa função é invocada no módulo *filehandle*, por isso ele foi escolhido como ponto de partida para testes de inserção de mensagens SEI no bitstream.

A partir do que foi observado na presente implementação de envio de mensagens SEI não registradas foi desenvolvida uma nova função que consegue criar uma mensagem SEI com qualquer tipo de dado, não sendo necessariamente texto. Para testar essa nova função foram feitas pequenas alterações no módulo filehandle do codificador, na função *start_sequence*, e foi adicionado um novo módulo *udata_gen* para possibilitar a criação de mensagens SEI a partir de um char* e do seu tamanho.

Também foram realizadas alterações no decodificador para realizar a extração da mensagem SEI, para isso foi criado um novo módulo *udata_parser* que faz a extração dos dados a partir de um SEI NALU que possui uma mensagem SEI do tipo *UserData* não registrada.

O módulo *udata_parser* é chamado dentro do módulo sei (*sei.c*) do decodificador (*ldecod*), na função *InterpretSEIMessage* que trata a recepção de mensagens SEI. Nesse ponto as informações são enviadas para o *stdout*, para constatar se as mensagens foram extraídas corretamente.

Com essas duas alterações prontas foi possível testar o envio de um *SEI User Data* não registrado contendo informação que não é necessariamente texto (nesse caso a ideia é que fossem

parâmetros extraídos do video).

Foi constatado que:

- As informações persistiram no bitstream e foram extraídas com sucesso pelo decodificador de referência modificado.
- O decodifica de referência realizou a decodificação do vídeo com sucesso, gerando um arquivo de vídeo bruto válido com todo o vídeo original.
- O bitstream H.264 gerado mostrou ser compatível com Gstreamer.

Portanto não seria um problema enviar dados de qualquer natureza utilizando mensagens *SEI User Data* não registradas, o decodificador de referência continuou funcionando sem problemas e a mensagem permaneceu íntegra. Até mesmo um decodificador de terceiro (ffmpeg + Gstreamer) funcionou com o bitstream gerado.

3.1.3 *ENVIANDO DIVERSAS MENSAGENS SEI USERDATA DE TAMANHO VARIADO*

Após ter enviado uma mensagem com sucesso o próximo passo foi enviar várias mensagens no mesmo bitstream, com tamanhos variados. Porém surgiram dois problemas, a quantidade de mensagens *SEI Userdata* que podem ser enviadas no mesmo bitstream e o tamanho máximo delas. De acordo com [3] não existe um limite explicito nem na quantidade, nem no tamanho das mensagens SEI.

Porém na prática, além dos limites implícitos que podem existir por causa do nível de conformidade do codificador, muitos decodificadores possuem limites no que eles suportam que estão relacionados com a maneira como eles foram implementados e não com a recomendação em si.

Alguns decodificadores possuem limites de quantas mensagens SEI podem receber, outros simplesmente descartam mensagens SEI. Como a variedade de decodificadores é muito grande, o foco da implementação será o codificador/decodificador de referência.

No codificador de referência existe um limite explicito no tamanho máximo de um NALU, esse limite é definido no arquivo *lcommon/inc/nalucommon.h* como *MAXNALUSIZE*. O valor desse limite é 64000 bytes. Ao longo dos testes esse será o tamanho máximo utilizado.

Para testar o envio de múltiplos SEI NALUS com mensagens *SEI Userdata* de tamanhos variados, respeitando o limite de 64000 bytes, foi criada uma nova função no módulo *udata_gen* que gera uma *string* maior a cada nova chamada.

Ao atingir o tamanho limite a função retorna ao tamanho inicial e reinicia o processo de aumento da string. Dessa maneira é possível enviar diversas mensagens todas de tamanho diferente, explorando desde mensagens de tamanho pequeno como de mensagens que cheguem perto do tamanho máximo.

Foi feita uma alteração no módulo *filehandle* do codificador, para criar e escrever no bitstream 1000 mensagens de tamanho variado. Nenhuma alteração foi necessária no decodificador.

Foi constatado que:

- Os 1000 SEI NALUs persistiram no bitstream e foram extraídos com sucesso pelo decodificador de referência modificado.
- O decodifica de referência realizou a decodificação do vídeo com sucesso, gerando um arquivo de vídeo bruto válido com todo o vídeo original.
- O bitstream H.264 gerado com os 1000 SEI NALUS mostrou ser compatível com Gstreamer.

Portanto não seria um problema enviar múltiplos SEI NALUs no mesmo bitstream, e de tamanho variado, desde que se respeite o tamanho máximo definido pelo software de referência. A única limitação deste teste é que os SEI NALUs são escritos no bitstream sequencialmente, não estão intercalados com os NALUs de vídeo e são todos escritos no inicio do bitstream, antes dos quadros de vídeo.

Foi realizado um teste para constatar se múltiplos SEI NALUs vão funcionar bem intercalados com quadros de vídeo. Para isso foram utilizadas as mesmas funções já existentes nos testes anteriores, mas ao invés de os SEI NALUs serem inseridos no módulo *filehandle* eles são inseridos no módulo *lencod*, após a geração de cada quadro.

Como o vídeo consiste em 300 quadros, foram inseridos exatamente 300 SEI NALUs, sempre que um quadro é escrito no bitstream um SEI NALU de tamanho variável é escrito após ele.

Foi constatado que:

- Os 300 SEI NALUs persistiram no bitstream e foram extraídos com sucesso pelo decodificador de referência modificado.
- O decodificador de referência realizou a decodificação do vídeo com sucesso, gerando um arquivo de vídeo bruto válido com todo o vídeo original.

- O bitstream H.264 gerado com os 300 SEI NALUS mostrou ser compatível com Gstreamer.

Utilizando mensagens *SEI Userdata* não registradas é possível enviar dados que não fazem parte do processo de decodificação, de tamanhos variados (respeitando o tamanho máximo estipulado pelo software de referência), em posições variadas dentro do bitstream e em quantidades variadas. O código fonte de todos os módulos alterados no software de referência e dos novos módulos, que foram utilizados para realizar esse teste, se encontra no apêndice A.

3.2 MÓDULO EXTRACTED_METADATA

Esta seção explica detalhadamente a implementação do módulo *extracted_metadata*. Ele é escrito em linguagem C e foi adicionado no diretório *lcommon* do software de referência já que ele é utilizado tanto pelo codificador como pelo decodificador. Todas as funções são declaradas no cabeçalho *lcommon/inc/extracted_metadata.h* e implementadas em *lcommon/src/extracted_metadata.c*.

Apesar de ser escrito em C, foi utilizada uma abordagem orientada a objetos. A orientação a objetos é feita internamente de uma maneira bem simples, a classe pai possui uma lista de ponteiros de função e na construção da classe filho ela fornece a implementação para essas funções.

A classe pai pode realizar alguma atividade antes ou depois da chamada da função implementada pela classe filho. Dessa maneira fica fácil a classe pai executar código que é comum a todas as subclasses, antes de chamar a função implementada pela subclasse. Isso foi especialmente útil na implementação da serialização, cada implementação de metadado deve se preocupar apenas com a serialização dos seus dados, sem se preocupar com os dados da classe pai, já que a classe pai irá serializar esses dados antes de chamar a função de serialização da classe filho, simplificando a implementação de novos metadados.

Funções da família *hton*, definidas em *arpa/inet.h*, foram utilizadas na serialização dos metadados para garantir que ao serializar os dados os bytes estejam de acordo com o padrão para rede. Na desserialização, funções da família *ntoh* foram utilizadas para garantir que os bytes ordenados de acordo com o padrão para rede fossem convertidos para a ordenação de *bytes* do *host*.

O padrão utilizado nos nomes das funções é *nome_do_tipo_método*. Por exemplo a implementação em C do método *new* da classe ExtractedMetadataBuffer será *extracted_metadata_buffer_new*.

Por classe entende-se a estrutura definida com o determinado nome, e sempre ao chamar uma função que é método desta classe, o primeiro parâmetro é um ponteiro para a estrutura que representa a classe. Ao longo dessa seção será feita referência apenas ao nome dos métodos, e não ao nome completo da função que está implementada em C.

Na figura 10 pode ser visto o diagrama de classe dos objetos definidos no módulo. A descrição de cada um dos objetos são apresentados nessa seção.

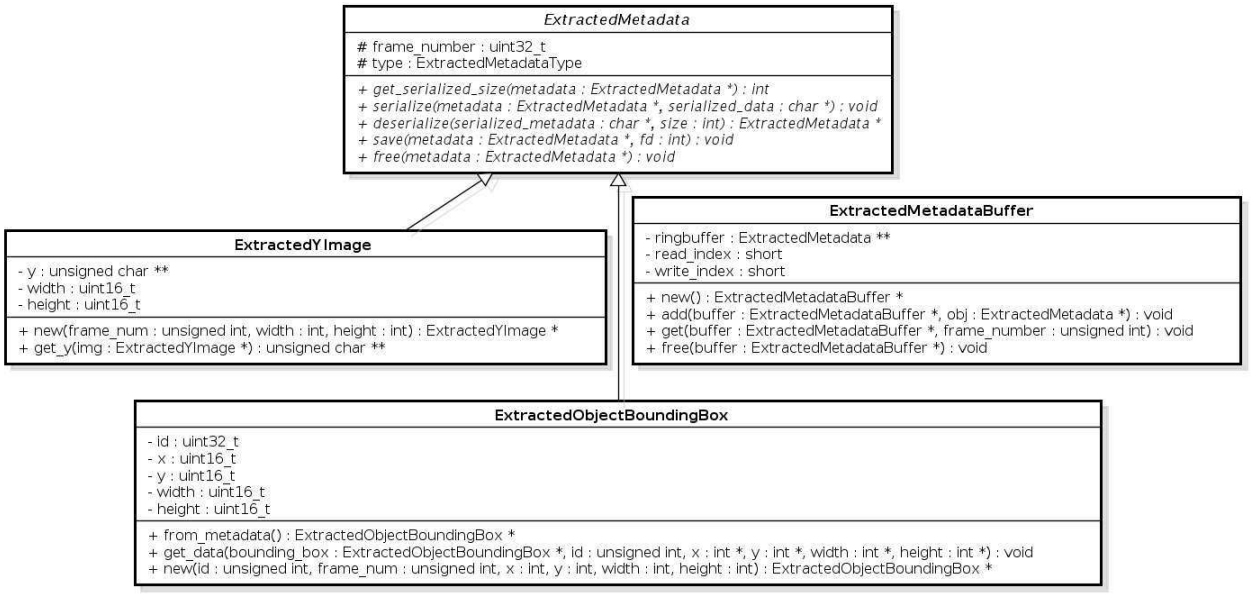

Figura 10: Diagrama de classe dos objetos definidos no módulo extracted_metadata.

A implementação deste módulo encontra-se no apêndice B.

3.2.1 *EXTRACTED_METADATA*

Esta classe abstrai a serialização, desserialização, salvamento e destruição dos diferentes tipos de metadados. O método *serialize* é responsável pela serialização, que é utilizada na inserção de um metadado no bitstream (utilizado em conjunto com o módulo *udata_gen*).

O método *get_serialized_size* é utilizado para auxiliar a alocação de memória necessária pelo processo de serialização, dessa maneira fica a cargo de quem está serializando o metadado alocar a memória necessária para a serialização e liberar a memória após o uso.

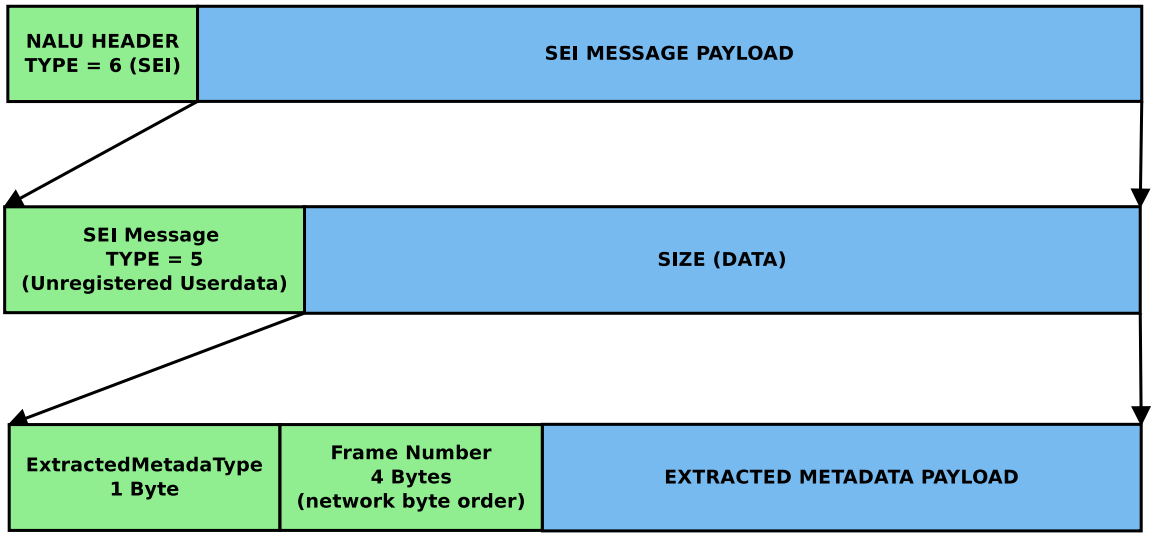

Figura 11: Metadado serializado dentro de um NALU do tipo SEI. Cabeçalho em verde, *payload* em azul.

Como pode ser visto na figura 11, um metadado serializado sempre começa com o seu tipo e o número do quadro (em ordem de apresentação) do qual ele foi extraído. O payload do metadado será definido pela classe que herdar *ExtractedMetadata*.

O método *save* salva o metadado em um file descriptor, sendo útil para realizar depuração. O método *free* libera todos os recursos utilizados pelo metadado.

3.2.2 *EXTRACTED_Y_IMAGE*

Esta classe herda *ExtractedMetadata* e representa o plano luma de um objeto de interesse, é composto basicamente do plano luma com o seu comprimento e altura. O plano luma é um *array* bidimensional de bytes, onde cada byte é o valor de um pixel. A primeira dimensão do array é a linha (Y) que vai de 0 á altura - 1. A segunda dimensão é a coluna (X), que vai de 0 á comprimento - 1.

Ao detectar um objeto de interesse é possível extrair apenas o objeto e inserir ele no bitstream. O objeto pode ser recuperado no decodificador e salvo em um arquivo. Como ele é composto apenas do plano luma só é possível visualizar o objeto monocromático. Muitos algoritmos trabalham com imagens monocromáticas, esse metadado pode ser utilizado para preservar no bitstream o objeto original antes que parte das informações dele possam ser destruídas pelo processo de codificação, dependendo do nível de perda do processo da codificação.

Em alguns casos para economizar largura de banda na transmissão, ou espaço no armazenamento, o vídeo é obtido em alta definição mas tem seu tamanho reduzido, utilizando esse

metadado seria possível dispor de um banco de objetos de interesse em alta resolução apesar dos vídeos terem seu tamanho reduzido.

Para criar um objeto *ExtractedYImage* basta utilizar o método *new* passando como parâmetro o número do quadro, seu comprimento e sua altura. Pela altura e comprimento informados será alocado o plano, que pode ser obtido através do método *get_y*, tendo obtido o plano basta escrever nele o plano luma do objeto e o metadado estará pronto para ser serializado e inserido no bitstream.

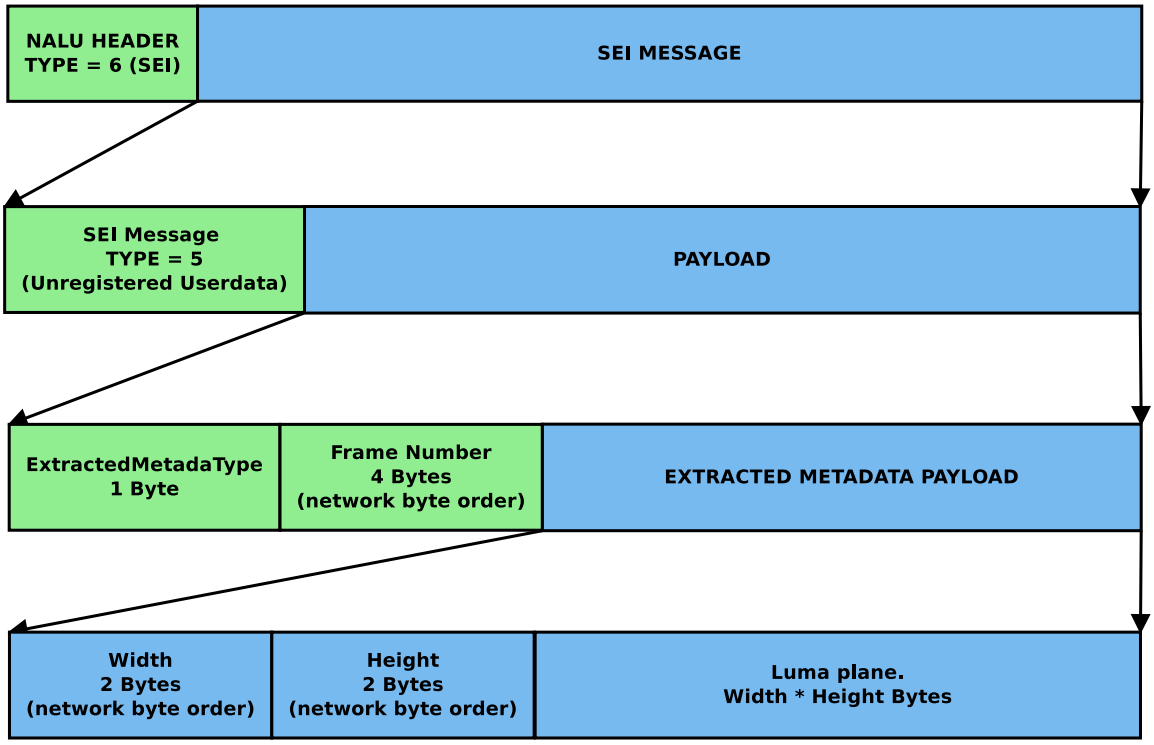

Figura 12: Objeto ExtractedYImage serializado dentro de um NALU do tipo SEI. Cabeçalho em verde, *payload* em azul.

3.2.3 *EXTRACTED_OBJECT_BOUNDING_BOX*

Esta classe herda *ExtractedMetadata* e representa uma caixa delimitadora envolta de um objeto de interesse. Ela é composta de um identificador único do objeto, as coordenadas (x, y) da caixa, sua altura e seu comprimento.

Ao ocorrer a detecção de um objeto de interesse é possível inserir no bitstream um metadado *ExtractedObjectBoundingBox*, ao ser recuperado no decodificador, utilizando o identificador único, é possível realizar a indexação de todos os objetos diferentes detectados ao longo do vídeo. Como a classe pai *ExtractedMetadata* possui o número do quadro em ordem de apresentação, é possível dizer em que momento do vídeo o objeto foi detectado pela primeira

vez e qual o momento que ele foi identificado pela última vez, tendo assim todo o intervalo de tempo em que o objeto esteve no vídeo.

Com as informações da caixa delimitadora também é possível desenhar a caixa diretamente no vídeo, facilitando a constatação visual de que um objeto de interesse está presente no trecho de vídeo. Se para cada quadro que possui o objeto de interesse for inserido um metadado *ExtractedObjectBoundingBox* no bitstream será possível realizar o tracking do objeto ao longo do vídeo.

Para criar um objeto *ExtractedObjectBoundingBox* basta utilizar o método *new* passando como parâmetro o identificador do objeto, o número do quadro (em ordem de apresentação), a coordenada x, a coordenada y, seu comprimento e sua altura. Após a construção basta serializar e inserir o objeto *ExtractedObjectBoundingBox* no bitstream.

Ao recuperar o metadado do bitstream o método *get_data* pode ser utilizado para se obter informações como o identificador único do objeto, suas coordenadas e seu tamanho.

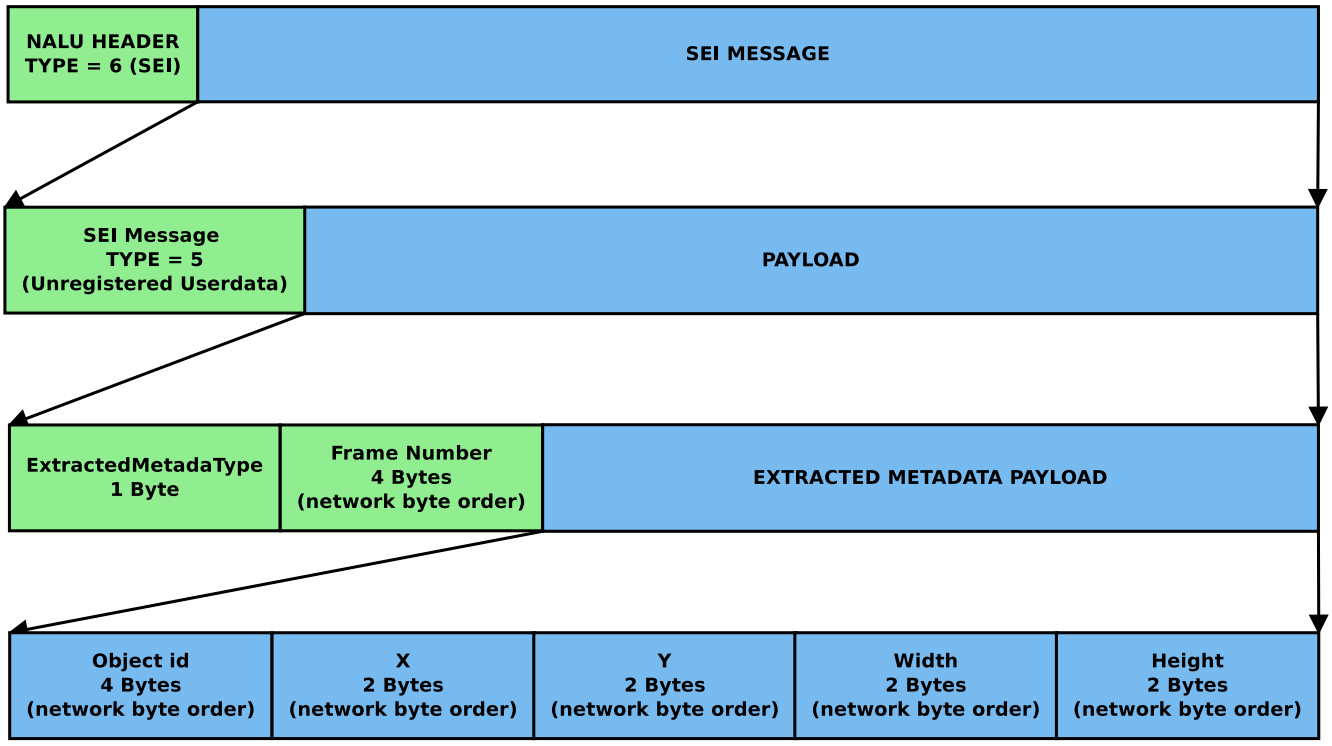

Figura 13: Objeto ExtractedObjectBoundingBox serializado dentro de um NALU do tipo SEI. Cabeçalho em verde, *payload* em azul.

3.2.4 *EXTRACTED_METADATA_BUFFER*

Esta classe é utilizada para bufferizar diversos metadados extraídos no processo de decodificação. O método *add* adiciona um novo metadado no buffer, os metadados podem ser recuperados do

buffer utilizando o método *get*, neste método deve ser informado o número do quadro em ordem de apresentação, se houver algum metadado referente a este quadro ele será retornado, se não existir o método retornará *NULL*.

O buffer foi construído porque nem sempre os metadados ficam no bitstream exatamente antes do quadro ao qual eles pertencem. Dependendo do vídeo vários metadados podem ser inseridos no bitstream antes que um quadro seja escrito no bitstream, dessa maneira é necessário bufferizar os metadados no decodificador até terminar a decodificação de um quadro e chegar o momento de apresentá-lo. Ao terminar a decodificação o buffer pode ser consultado para verificar se existe algum metadado referente ao quadro que será apresentado.

Para tornar a bufferização eficiente e simples não existe nenhum tipo de ordenação dos metadados dentro do buffer, ao se obter o metadado para um determinado quadro, qualquer metadado que for referente a um quadro anterior será descartado pelo buffer. Portanto um metadado deve ser gravado no bitstream antes do quadro do qual ele foi extraído. E os metadados devem sempre ser gravados no bitstream na ordem correta (os metadados referentes a quadros menores primeiro). Isso é fácil de garantir já que no codificador temos total controle de quando o metadado é gravado no bitstream, o que já não é verdade para os quadros que dependem de vários outros processos internos do codificador.

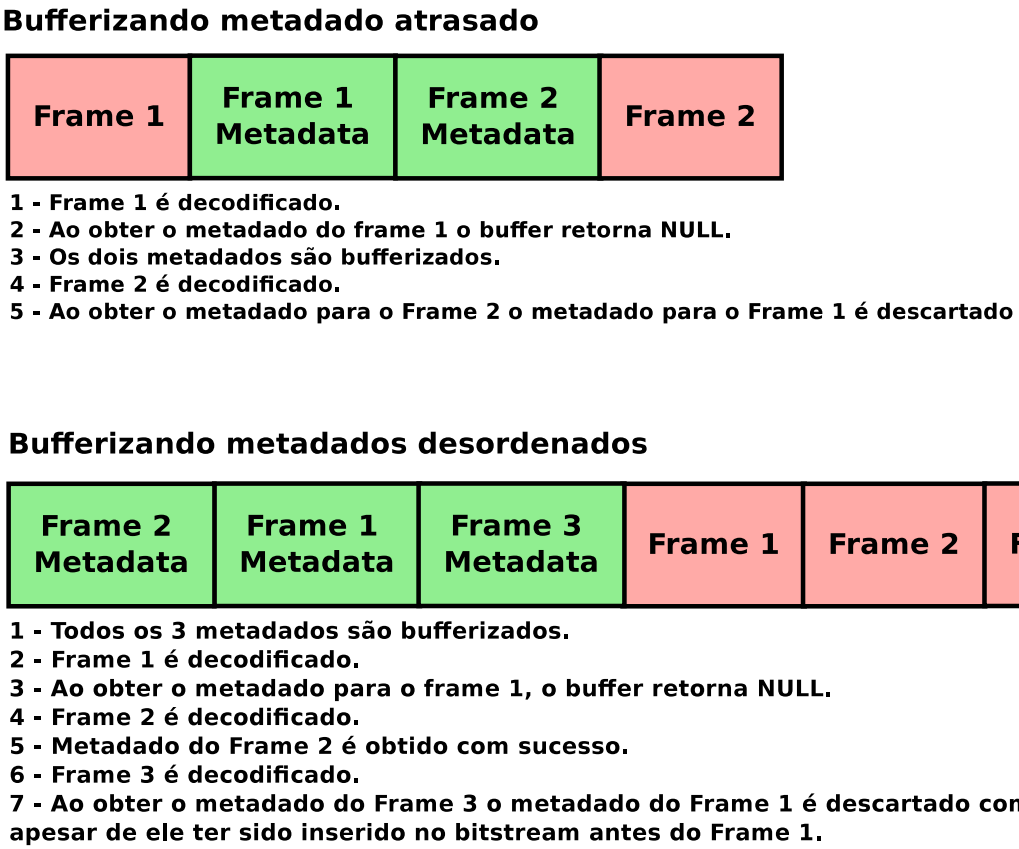

Figura 14: Exemplo de problemas que podem ocorrer na recuperação dos metadados se eles forem inseridos de maneira errada no bitstream. Metadados em verde, quadros em vermelho.

A figura 14 exemplifica alguns problemas que podem ocorrer ao bufferizar metadados que não foram inseridos no bitstream de maneira adequada.

3.3 MÓDULO METADATA_EXTRACTOR

Este módulo é responsável pela extração de metadados a partir de quadros brutos e das informações de estimativa de movimento fornecidas. Na atual implementação ele pode extrair metadados do tipo *ExtractedObjectBoundingBox* e *ExtractedYImage*.

É escrito em linguagem C e se encontra no diretório *lencod* do software de referência já que ele é utilizado tanto pelo codificador como pelo decodificador. Todas as funções são declaradas no cabeçalho *lencod/inc/metadata_extractor.h* e implementadas em *lencod/src/metadata_extractor.c*.

Nele também foi utilizada a mesma abordagem orientada a objetos utilizada no módulo *extracted_metadata* e as mesmas regras foram utilizadas para os nomes das funções que representam métodos da classe *MetadataExtractor*.

O uso do classificador Haar na implementação do extrator de metadados será comentado nessa seção. A implementação completa do módulo encontra-se no apêndice C.

3.3.1 *METADATA_EXTRACTOR*

É o único objeto exportado publicamente pelo módulo e possui todo o contexto da extração de metadados. O método *new* cria um novo extrator de metadados, na criação várias características do extrator podem ser configuradas por meio dos seguintes parâmetros:

- min_width - Comprimento mínimo do objeto de interesse.
- min_height - Altura mínima do objeto de interesse.
- search_hysteresis - Histerese (em quadros) para realização da busca de um novo objeto de interesse.
- tracking_hysteresis - Histerese (em quadros) para realizar a confirmação de que o objeto de interesse do qual está sendo feito *tracking* ainda se encontra no vídeo, e corrigir possíveis erros causados pela estimativa de movimento.
- training_file - Arquivo de treinamento que será utilizado ao criar o classificador Haar desse extrator. Ele define qual é o objeto de interesse.

O termo histerese tem diferentes significados de acordo com o contexto e a área em que é utilizado, pode ser utilizado na biologia, tratamento de materiais, mecânica, entre outros. No presente trabalho a histerese se refere a quantidade de quadros necessários para realizar a execução do classificador Haar, gerando um atraso configurável na detecção de um novo objeto, ou ao confirmar a existência de um objeto previamente detectado.

O método *extract_raw_object* extrai do quadro o plano luma do objeto de interesse (se ele existir no quadro) e retorna como um objeto *ExtractedMetadata*. Para realizar a extração é necessário passar os seguintes parâmetros:

- extractor - Objeto *MetadataExtractor*.
- frame_number - Número do quadro (em ordem de apresentação) do qual será extraído o metadado.
- y - Plano luma, array bidimensional[y][x] de bytes, onde y vai de 0 á altura - 1 e x vai de 0 á comprimento - 1.

- width - Comprimento do quadro.
- height - Altura do quadro.

Se o objeto de interesse não se encontrar no quadro, esse método retorna *NULL*. As configurações de histerese e as informações de estimativa de movimento não são utilizadas nesse método.

O método *extract_object_bounding_box* extrai do quadro uma caixa delimitadora que representa a área onde se encontra o objeto de interesse no quadro e retorna como um objeto *ExtractedMetadata*. Para realizar a extração é necessário passar os seguintes parâmetros:

- extractor - Objeto *MetadataExtractor*.
- frame_number - Número do quadro (em ordem de apresentação) do qual será extraído o metadado.
- y - Plano luma, array bidimensional[y][x] de bytes, onde y vai de 0 à altura - 1 e x vai de 0 ao comprimento - 1.
- width - Comprimento do quadro.
- height - Altura do quadro.

Se o objeto de interesse não se encontrar no quadro, esse método retorna *NULL*. As configurações de histerese e as informações de estimativa de movimento são utilizadas nesse método de acordo com o estado interno do extrator. O comportamento do método depende do estado interno do extrator:

- Não realizando *tracking* - Será consultada a histerese de busca, se a histerese foi ultrapassada, o classificador Haar será utilizado para realizar a busca do objeto de interesse neste quadro. Se o objeto existir, o extrator cria um objeto *TrackedBoundingBox* que será utilizado internamente para realizar o *tracking* do objeto, passa para o estado *Realizando tracking* e retorna um objeto *ExtractedObjectBoundingBox*. Se o objeto não existir o extrator reinicia a histerese de busca, continua no estado atual e retorna *NULL*. Se a histerese não foi ultrapassada o extrator continua no estado atual e retorna *NULL*.
- Realizando *tracking* - Será consultada a histerese de *tracking*, se a histerese foi ultrapassada, o classificador Haar será utilizado para verificar se o objeto ainda se encontra no quadro atual, se o objeto não for encontrado o extrator volta ao estado *Não realizando*

tracking e retorna NULL. Se o objeto de interesse está no quadro o objeto *TrackedBoundingBox* é atualizado de acordo com a posição encontrada pelo classificador Haar e um *ExtractedObjectBoundingBox* é retornado. Se a histerese não foi ultrapassada será retornado um objeto *ExtractedObjectBoundingBox* utilizando as informações de estimativa de movimento para calcular a posição atual do objeto *TrackedBoundingBox*.

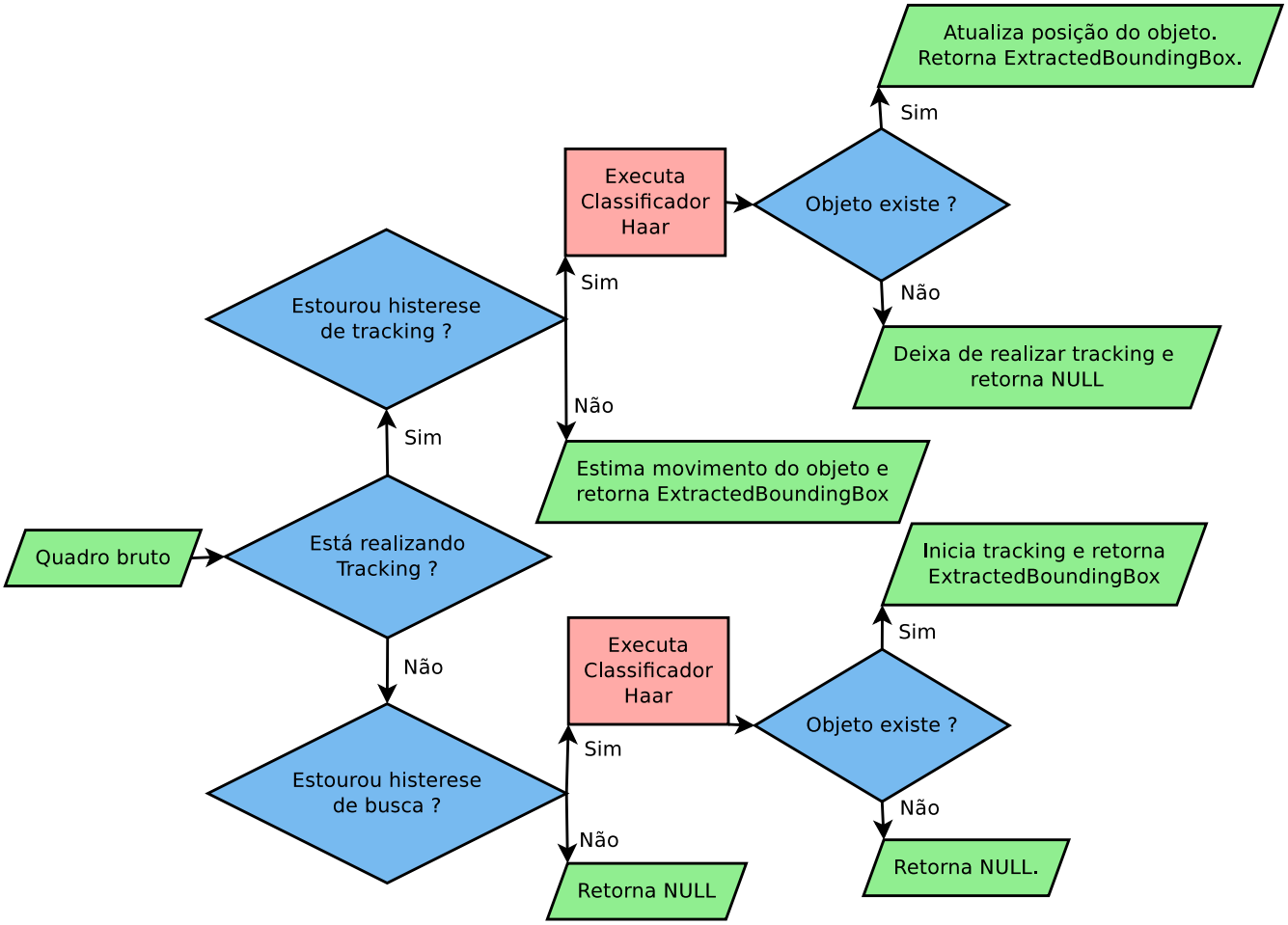

Figura 15: Fluxograma do método extract_object_bounding_box.

As histereses configuradas, junto com as informações de estimativa de movimento, visam utilizar os algoritmos já existentes no codificador para reduzir o custo computacional do *tracking* de um objeto no vídeo. Ao invés de realizar o processamento de todos os quadros brutos no classificador Haar para realizar o *tracking* do objeto ao longo do vídeo, a histerese diminui o custo computacional por atualizar a posição do objeto em intervalos fixos. Os vetores de movimento calculados pelo codificador são utilizados para suavizar o *tracking*, estimando a nova posição do objeto enquanto a histerese de *tracking* não estoura.

Por exemplo, em um vídeo com 300 quadros sem nenhum objeto de interesse, com uma histerese de busca de 10 quadros, o classificador Haar será chamado apenas 30 vezes, ao invés

de 300 vezes. A configuração é especialmente útil em vídeos com *framerate* muito alto, já que determinados objetos têm uma velocidade de locomoção esperada em um vídeo, logo não é necessário processar todos os quadros.

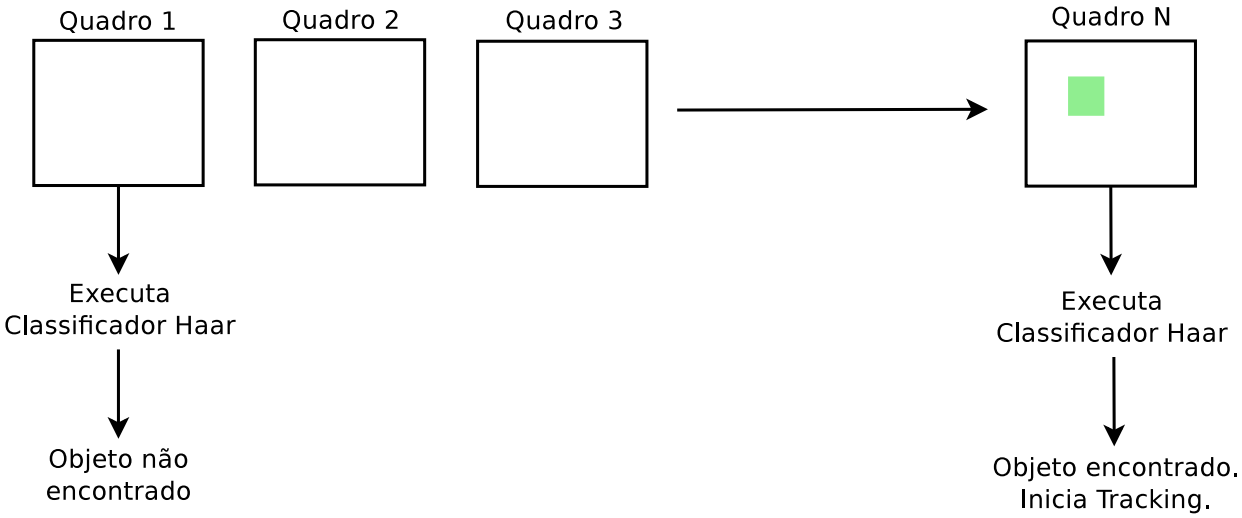

Figura 16: Exemplo do funcionamento da histerese de busca.

O mesmo se dá quando o objeto já foi detectado, não é necessário executar o classificador em todos os quadros, os vetores de estimativa de movimento nos dão uma ideia aproximada da posição do objeto, até que seja alcançada a histerese de *tracking*.

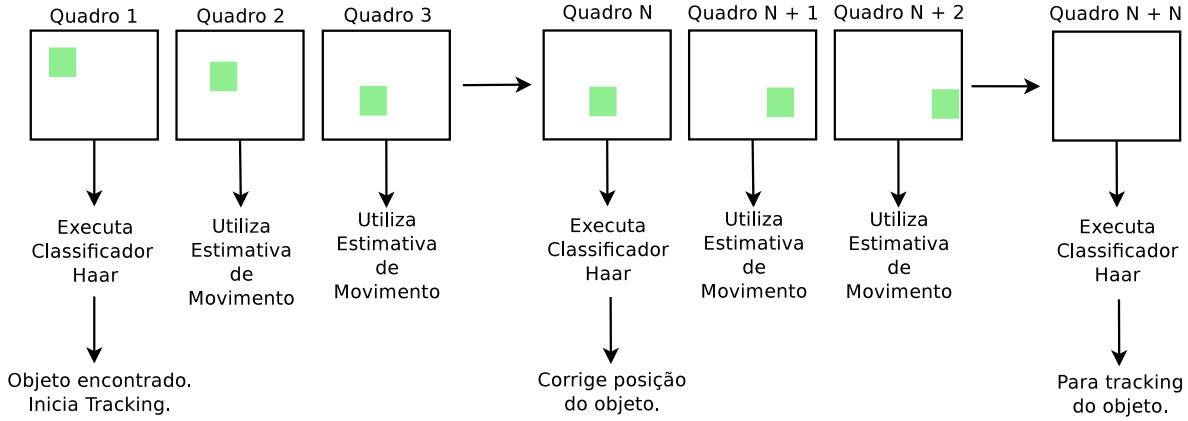

Figura 17: Exemplo do funcionamento da histerese de *tracking*.

Cada extrator só pode realizar a monitoração de um objeto de interesse, se existirem diversos objetos de interesse no quadro ele irá detectar apenas um e realizar o *tracking* desse objeto. Criar vários extratores para detectar o mesmo tipo de objeto geraria problemas como detectar várias vezes o mesmo objeto, mas seria possível criar vários extratores para detectar objetos diferentes e utilizá-los em paralelo.

3.3.2 *UTILIZANDO O CLASSIFICADOR HAAR*

O classificador Haar foi escolhido como algoritmo de detecção de padrões a ser utilizado nesse trabalho. Como ele ficou completamente oculto na implementação do módulo *metadata_extractor* é possível alterar a implementação do módulo, utilizando outro algoritmo de detecção de padrões, sem realizar qualquer alteração na API pública do módulo, ou simplesmente removê-lo. A única limitação da API atual é que ela trabalha apenas com as informações de luminância, para trabalhar com algoritmos que utilizem informações de cor será necessária uma alteração na API do módulo atual.

A implementação do classificador Haar do OpenCV foi escolhida para realizar a detecção de objetos por ser bem documentada e fácil de utilizar, sem contar que o OpenCV já dispõe de uma série de outras funções que facilitam trabalhar com o quadro bruto. Outra vantagem é que ele trabalha com imagens em escala de cinza, independente da configuração do vídeo bruto o plano de luminância sempre possui a maior resolução possível. Técnicas que utilizam informações de cor teriam a desvantagem de ter de trabalhar com informações de cor com resolução reduzida.

Outra motivação é a possibilidade de acelerar o classificador Haar diretamente em hardware, como pode ser visto em [8], onde a parte mais complexa do classificador Haar (90% do tempo de processamento) foi implementando em hardware, o que se alinha com a ideia de construir um chip codificador H.264 com detecção de objetos integrada em tempo real.

A principal função que é utilizada na detecção de um objeto é *cvHaarDetectObjects(const CvArr* image, CvHaarClassifierCascade* cascade, CvMemStorage* storage, double scale_factor, int min_neighbors, int flags, CvSize min_size)*.

O primeiro parâmetro é a imagem onde será realizada a busca, em escala de cinza, que será obtida através do plano luma original do codificador. O segundo parâmetro é o classificador que é criado no método *new* do objeto *MetadataExtractor*. O terceiro parâmetro é o buffer de trabalho do classificador que também é criado no método *new*.

O classificador busca pelo objeto de interesse em todas as escalas possíveis, o quarto parâmetro significa o quão grande será o pulo entre cada escala, um valor alto vai tornar o classificador mais rápido porém aumentará as chances de falso negativo. O quinto parâmetro é um controle para prevenir falsos positivos. Na área onde se encontra o objeto de interesse costuma ocorrer várias detecções com sucesso para o mesmo objeto, já que os pixels ao redor da área e em escalas diferentes indicam que existe o objeto de interesse. Por exemplo, o valor 3 indica que o classificador só considerará o objeto como presente se ocorrerem 3 detecções

sobrepostas na mesma área.

O sexto parâmetro são as flags do processo de busca, existem 4 flags que podem ser compostas com o operador OR:

- CV_HAAR_DO_CANNY_PRUNING - Fará com que regiões planas da imagem (sem linhas) sejam ignoradas.
- CV_HAAR_SCALE_IMAGE - Faz com que o classificador escalone a imagem em vez de escalonar ele mesmo.
- CV_HAAR_FIND_BIGGEST_OBJECT - Faz com que o classificador retorne apenas o maior objeto detectado (se houver mais de um).
- CV_HAAR_DO_ROUGH_SEARCH - Só pode ser utilizado em conjunto com a flag CV_HAAR_FIND_BIGGEST_OBJECT, faz com que o classificador encerre a busca assim que ele encontrar o objeto de interesse (nesse caso o maior).

O sétimo parâmetro é o tamanho mínimo que o objeto deve possuir para ser detectado pelo classificador. Quanto maior for o tamanho mínimo mais rápido o classificador irá executar, mas maior será a possibilidade de ocorrer falso negativo.

3.3.3 *REALIZANDO ESTIMATIVA DE MOVIMENTO*

A estimativa de movimento de um objeto é realizada por um trabalho em conjunto entre o método *extract_object_bounding_box* e o método *add_motion_estimation_info*. Um *MetadataExtractor* recebe as informações de estimativa de movimento através do método *add_motion_estimation_info*, essas informações por sua vez serão utilizadas na execução do método *extract_object_bounding_box*. O método *add_motion_estimation_info* recebe os seguintes parâmetros:

- extractor - Objeto *MetadataExtractor*.
- block_x - X do bloco, em coordenada de bloco.
- block_y - Y do bloco, em coordenada de bloco.
- x_motion_estimation - Estimativa de movimento para a coordenada x do bloco.
- y_motion_estimation - Estimativa de movimento para a coordenada y do bloco.

Se o extrator não estiver realizando *tracking* de um objeto a chamada para este método é ignorada. Se o extrator está realizando *tracking*, será avaliado se este bloco faz parte da caixa delimitadora (representada pela classe *TrackedBoundingBox*) do objeto de interesse. Se qualquer parte do bloco estiver dentro da caixa delimitadora, as informações de estimativa de movimento dele serão utilizadas para estimar o movimento da caixa delimitadora do objeto de interesse.

Essa abordagem é eficiente já que bastam algumas comparações bem simples para verificar se a área do bloco tem uma interseção com a área da caixa delimitadora do objeto de interesse, porém a caixa delimitadora nem sempre representa perfeitamente o objeto (o objeto teria de ter a forma de uma caixa também), dessa maneira acaba-se utilizando a estimativa de movimento de blocos que fazem parte da caixa delimitadora mas não do objeto, gerando um erro que se acumula a cada estimativa de movimento do objeto. Isso limita o uso da técnica, sendo necessária uma histerese de *tracking* pequena dependendo do quanto o objeto se movimenta e da sua caixa delimitadora.

No software de referência utiliza-se um modelo simplificado de estimativa de movimento, o tamanho dos blocos utilizados na estimativa de movimento sempre possuem um tamanho de 4 x 4 pixels, independente do modo escolhido. Dessa maneira para obter as coordenadas reais do bloco basta multiplicar as coordenadas do bloco por 4. Tendo as coordenadas reais do bloco fica fácil determinar se o bloco está dentro ou não da caixa delimitadora do objeto de interesse.

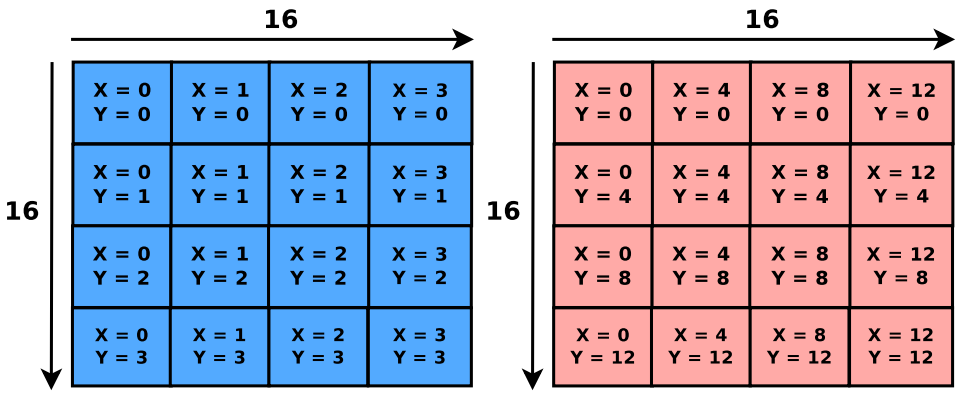

Figura 18: Comparação entre as coordenadas dos blocos (em azul) da estimativa de movimento e as coordenadas reais (em vermelho) que elas representam.

É importante ressaltar que o tamanho fixo 4 x 4 para os blocos da estimativa de movimento não é verdade para todos os codificadores. Alguns codificadores possuem uma hierarquia e seria necessário derivar a estimativa de movimento para um bloco dependendo do modo utilizado.

O valor da estimativa de movimento é informado em unidades de QPel (Quarter Pel refinement), onde um quarto das amostras são sub-amostras interpoladas, causadas por frações de vetores de movimento. Mais detalhes a respeito desse processo podem ser encontrados em [3] subcláusula 8.4.2.2. Um vetor x negativo indica um movimento para a direita, y negativo indica um movimento para baixo, x positivo indica um movimento para a esquerda e um y positivo um movimento para cima.

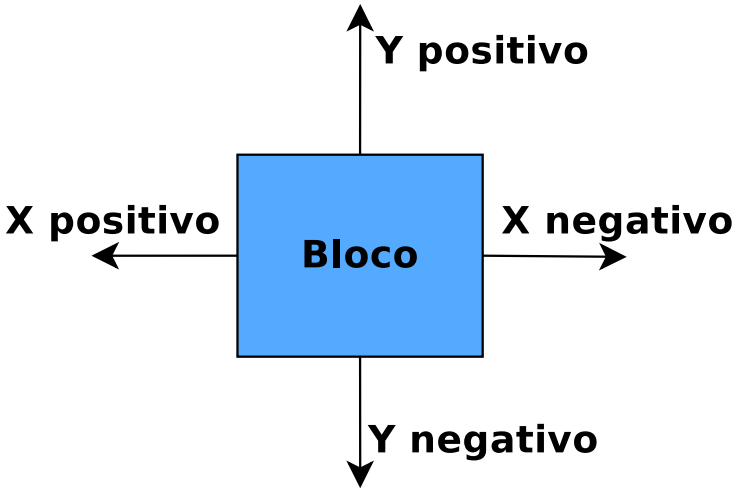

Figura 19: Como os vetores de movimento de um bloco representam a sua movimentação no vídeo.

Todas as informações relativas a estimativa de movimento do objeto de interesse são guardadas no *TrackedBoundingBox*, essa classe representa a caixa delimitadora do objeto de interesse do qual está sendo feito o *tracking*, e é criada quando se identifica um objeto de interesse pela primeira vez. Nele é guardado um somatório de todos os vetores (em QPel) de movimento alimentados no extrator. Ao chamar o método *extract_object_bounding_box* no estado de *tracking*, serão utilizadas as informações contidas no *TrackedBoundingBox* para realizar a estimativa de movimento do objeto.

Nesse método é realizada a média aritmética simples de todos os vetores, ou seja o somatório de todos os vetores é dividido pela quantidade de blocos que foram utilizados para realizar a estimativa de movimento. O somatório de todos os vetores é dividido por 4 antes de ser dividido pela quantidade de blocos, pois o somatório dos vetores está em unidades de QPel. Realizar essa divisão por 4 nos fornece a estimativa de movimento em pixels.

Tendo a estimativa de movimento em pixels basta subtrair a estimativa das coordenadas da *TrackedBoundingBox*. Essas novas coordenadas serão utilizadas para criar o *ExtractedObjectBoundingBox* que é retornado. As coordenadas (x,y) são guardadas como double no *TrackedBoundingBox* para que não ocorra perda de precisão no acúmulo de pequenos movimentos,

como pode ser visto na figura 20.

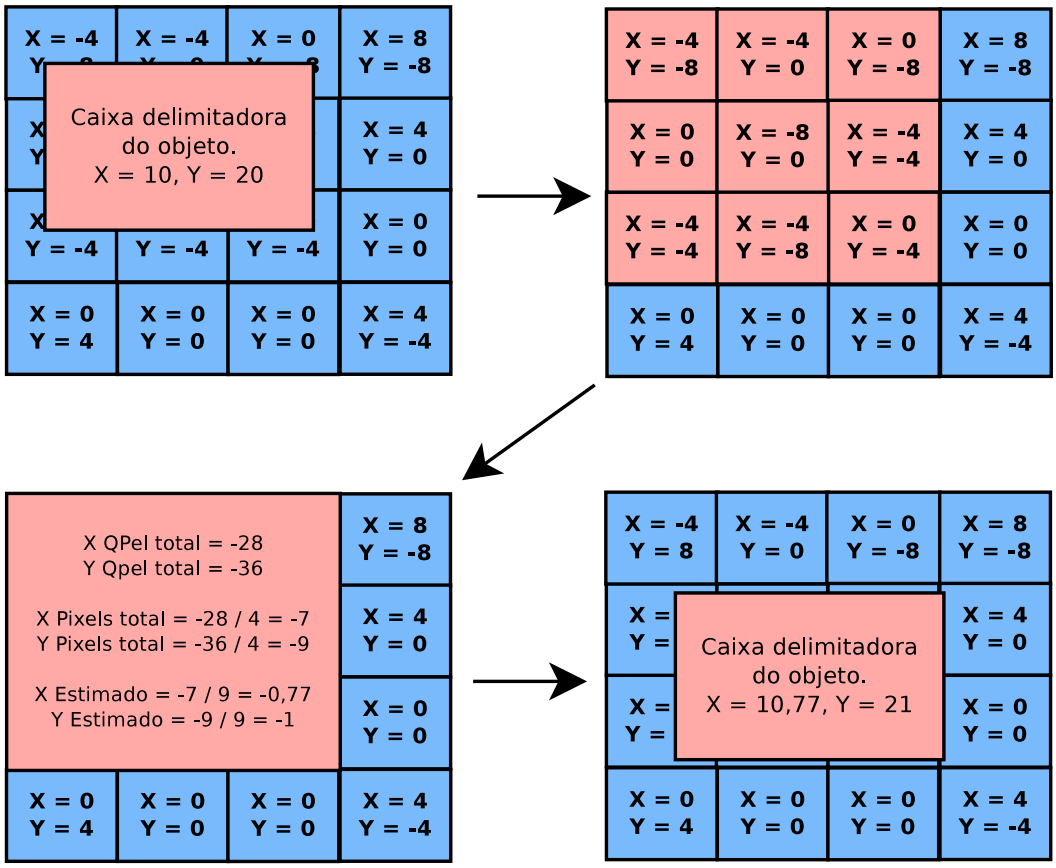

Figura 20: Estimativa de movimento de um objeto.

Esse processo pode ser visto em detalhes no método *estimate_motion* da classe *TrackedBoundingBox* que se encontra implementado em */lencod/src/metadata_extractor.c*.

3.4 ALTERAÇÕES REALIZADAS NO CODIFICADOR

O objetivo da criação dos módulos *metadata_extractor* e *extracted_metadata* foi reduzir ao máximo as alterações que teriam de ser realizadas no codificador e no decodificador e desacoplando o processo de extração de metadados do software de referência. Isso facilita a possibilidade de portar o sistema de detecção e *tracking* de objetos para uma outra implementação do H264.

O software de referência utilizado no presente projeto é desenvolvido pelo JVT (Joint Video Team) e hospedado pelo Instituto Fraunhofer para telecomunicações, Instituto Heinrich Hertz. Esta implementação será utilizada por ela ser referência para pesquisas e verificação de conformidade com a norma.

A versão utilizada é a 17.2, o download da mesma pode ser realizado gratuitamente em http://iphome.hhi.de/suehring/tml/download, todos os módulos são bem divididos e possuem boa documentação. O software consiste basicamente de um codificador (*lencod*) que codifica um arquivo de vídeo raw para o formato H.264 e de um decodificador (*ldecod*) que decodifica um arquivo H.264 em um arquivo de vídeo raw. Algumas funções de uso comum se encontram no módulo *lcommon*.

Nesta seção serão apresentadas as alterações realizadas no codificador do software de referência, para integrá-lo com os módulos construídos. As alterações realizadas no codificador se encontram no anexo A.

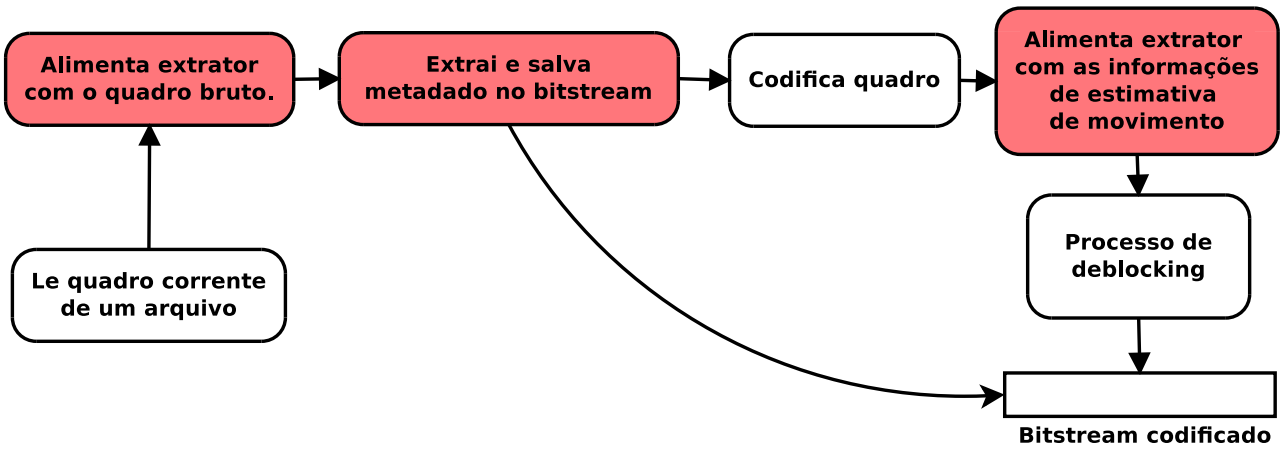

Figura 21: Visão geral do codificador H.264 modificado. Os blocos vermelhos representam os processos adicionados ao codificador.

3.4.1 *PROCESSANDO O QUADRO BRUTO*

Para extrair metadados do quadro bruto antes de ocorrer o processo de codificação, foi necessário primeiro descobrir qual o melhor local dentro do codificador para se obter o quadro que vai ser codificado. No caso do software de referência utilizado neste trabalho a entrada de dados é feita por arquivo. Foi necessário estudar quando o codificador lê o quadro e como ele organiza este quadro na memória.

A partir da função *main* em *lencod/src/lencod.c*, analisando como o codificador procedia para codificar os quadros, foi encontrada uma chamada para a função *encode_one_frame*, esta função é implementada no módulo *lencod/src/image.c*, é nesta função que o quadro é carregado em memória e pré-processado.

Dentro da função *encode_one_frame*, o processamento do quadro bruto é realizado logo após a chamada de função *process_image*, que realiza o processamento final em cima do quadro bruto, antes de se iniciar a codificação dele. É neste ponto que o método *extract_object_boun-*

ding_box é chamado e o metadado extraído é salvo no bitstream na forma de mensagem SEI *Unregistered Userdata*, utilizando o módulo *udata_gen*.

```
1
2  process_image(p_Vid, p_Inp);
3
4  if (p_Inp->object_detection_enable) {
5
6    ExtractedMetadata * metadata =
         metadata_extractor_extract_object_bounding_box(p_Vid->
         metadata_extractor, p_Vid->frame_no, (unsigned char **) p_Vid->imgData
         .frm_data[0], p_Vid->imgData.format.width[0], p_Vid->imgData.format.
         height[0]);
7
8    if (metadata) {
9      int size                = extracted_metadata_get_serialized_size(
           metadata);
10     char * data             = malloc(size);
11     NALU_t * nalu           = NULL;
12
13     /* Serialize the metadata */
14     extracted_metadata_serialize(metadata, data);
15
16     /* Insert the serialized metadata on the bitstream as SEI NALU. */
17     nalu = user_data_generate_unregistered_sei_nalu(data, size);
18     p_Vid->WriteNALU (p_Vid, nalu);
19
20     FreeNALU (nalu);
21     free(data);
22     extracted_metadata_free(metadata);
23   }
24
25 }
26
27 pad_borders (p_Inp->output, p_Vid->width, p_Vid->height, p_Vid->width_cr,
       p_Vid->height_cr, p_Vid->imgData.frm_data);
```

Tendo encontrado um bom lugar para extrair as informações foi necessário entender como o quadro fica organizado na memória. A estrutura *ImageData* armazena as informações de um quadro, e o quadro original é carregado no campo *imgdata* da estrutura *VideoParameters*. A definição da estrutura *ImageData* se encontra no módulo *lcommon/inc/io_image.h* e a estrutura *VideoParameters* é definida em *lencod/inc/global.h*.

O quadro original carregado se encontra no campo *frm_data* da estrutura *ImageData*:

```
1
2  typedef struct image_data
3  {
4    FrameFormat format;                //!< image format
5    // Standard data
6    imgpel **frm_data[MAX_PLANE];      //!< Frame Data
7    imgpel **top_data[MAX_PLANE];      //!< pointers to top field data
8    imgpel **bot_data[MAX_PLANE];      //!< pointers to bottom field data
9
10   //! Optional data (could also add uint8 data in case imgpel is of type
         uint16)
11   //! These can be useful for enabling input/conversion of content of
         different types
12   //! while keeping optimal processing size.
13   uint16 **frm_uint16[MAX_PLANE];    //!< optional frame Data for uint16
14   uint16 **top_uint16[MAX_PLANE];    //!< optional pointers to top field
         data
15   uint16 **bot_uint16[MAX_PLANE];    //!< optional pointers to bottom field
         data
16
17   int frm_stride[MAX_PLANE];
18   int top_stride[MAX_PLANE];
19   int bot_stride[MAX_PLANE];
20 } ImageData;
```

Este campo é um array tri-dimensional de *imgpel*, o tipo *imgpel* representa o tamanho de cada pixel e é definido no módulo *lcommon/inc/typedefs.h*:

```
1
2  #if (IMGTYPE == 0)
3  typedef byte   imgpel;             //!< pixel type
4  typedef uint16 distpel;            //!< distortion type (for pixels)
5  typedef int32  distblk;            //!< distortion type (for Macroblock)
6  typedef int32  transpel;           //!< transformed coefficient type
7  #elif (IMGTYPE == 2)
8  typedef float imgpel;
9  typedef float distpel;
10 typedef float distblk;
11 typedef int32 transpel;
12 #else
13 typedef uint16 imgpel;
14 typedef uint32 distpel;
```

```
15 typedef int64   distblk;
16 typedef int32   transpel;
17 #endif
```

IMGTYPE é o que define o tamanho do pixel, e pode ser definido tanto para o codificador como para o decodificador (cada um deles pode trabalhar com um tamanho de pixel diferente). No encoder a definição do *IMGTYPE* se encontra em *lencod/inc/defines.h* e do decodificador em *ldecod/inc/defines.h*. Por padrão o software de referência vem com esse valor definido como 1, ou seja utiliza 16 bits para representar cada pixel.

Tanto o encoder como o decodificador foram alterados para trabalhar com 8 bits para representar cada pixel, já que utilizar 16 bits não trazia vantagem alguma na extração dos metadados, gerando um overhead desnecessário.

MAX_PLANE é o que define a quantidade de planos existentes no quadro, e pode ser definido tanto para o encoder como para o decodificador (cada um deles pode possuir uma quantidade de planos diferente). No encoder a definição do *MAX_PLANE* se encontra em *lencod/inc/defines.h* e no decodificador em *ldecod/inc/defines.h*. Por padrão o software de referência vem com esse valor definido como 3, ou seja existem 3 planos em cada quadro.

Normalmente existem duas maneiras de se representar uma imagem: intercalada e planar. No caso dos quadros carregados no software de referência a representação é planar, no array tridimensional a primeira dimensão é o plano, *frm_data[0]* é o plano Y, *frm_data[1]* é o plano U e *frm_data[2]* é o plano V.

A segunda dimensão é o y/vertical do plano. A terceira dimensão é o x/horizontal do plano. O comprimento e a altura de cada plano se encontra no campo *format* da estrutura *ImageData*, que é do tipo *FrameFormat*. O índice 0 acessa o comprimento e altura do plano Y, o índice 1 acessa o comprimento e altura dos planos de cor (UV), o tipo *FrameFormat* é definido em *lcommon/inc/frame.h*:

```
1
2 typedef struct frame_format
3 {
4   ColorFormat yuv_format;                    //!< YUV format (0=4:0:0,
      1=4:2:0, 2=4:2:2, 3=4:4:4)
5   ColorModel  color_model;                   //!< 4:4:4 format (0: YUV, 1:
      RGB, 2: XYZ)
6   double      frame_rate;                    //!< frame rate
7   int         width[3];                      //!< component frame width
8   int         height[3];                     //!< component frame height
```

```
9   int          auto_crop_right;          //!< luma component auto crop
        right
10  int          auto_crop_bottom;         //!< luma component auto crop
        bottom
11  int          auto_crop_right_cr;       //!< chroma component auto
        crop right
12  int          auto_crop_bottom_cr;      //!< chroma component auto
        crop bottom
13  int          width_crop;               //!< width after cropping
        consideration
14  int          height_crop;              //!< height after cropping
        consideration
15  int          mb_width;                 //!< luma component frame
        width
16  int          mb_height;                //!< luma component frame
        height
17  int          size_cmp[3];              //!< component sizes (width *
        height)
18  int          size;                     //!< total image size (sum of
        size_cmp)
19  int          bit_depth[3];             //!< component bit depth
20  int          max_value[3];             //!< component max value
21  int          max_value_sq[3];          //!< component max value
        squared
22  int          pic_unit_size_on_disk;    //!< picture sample unit size
        on storage medium
23  int          pic_unit_size_shift3;     //!< pic_unit_size_on_disk >>
        3
24  } FrameFormat;
```

Para observar em mais detalhes como trabalhar com a estrutura *ImageData* as funções *MuxImages* e *FilterImageSep* no módulo *lcommon/src/img_process.c* são bons exemplos.

3.4.2 *OBTENDO ESTIMATIVA DE MOVIMENTO*

Além dos quadros brutos, outra informação importante para o extrator de metadados são os vetores de movimento dos blocos.

Na função *code_a_plane* antes da chamada de função *DeblockFrame* todo o processo de estimativa de movimento já está completo, e todos os vetores de movimento para todos os blocos da imagem estão disponíveis. Neste ponto é que os vetores de estimativa de movimento do

quadro são repassados ao extrator de metadados para que ele realize a estimativa de movimento do objeto detectado (se o extrator estiver realizando *tracking*).

Os vetores se encontram na estrutura *VideoParameters* (definida em *lencode/inc/global.h*) no campo *enc_picture* que é do tipo *struct storable_picture*. Sua definição se encontra em *lencode/inc/mbuffer.h*, essa estrutura possui o campo *mv_info* que é um array bidimensional de *PicMotionParams*.

A primeira dimensão é y do bloco, que vai de 0 até (altura da imagem / tamanho do bloco). A segunda dimensão é o x do bloco, que vai de 0 até (comprimento da imagem / tamanho do bloco). Ao acessar as duas dimensões temos o *PicMotionParams* que possui os vetores de movimento para o bloco[y][x]. A estrutura *PicMotionParams* é definida em *lencode/inc/mbuffer.h* da seguinte maneira:

```
1 //! definition of pic motion parameters
2 typedef struct pic_motion_params
3 {
4   struct storable_picture *ref_pic[2];  //!< referrence picture pointer
5   char                    ref_idx[2];  //!< reference picture   [list][
      subblock_y][subblock_x]
6   MotionVector            mv[2];       //!< motion vector
7   byte                    field_frame; //!< indicates if co_located is
      field or frame. Will be removed at some point
8 } PicMotionParams;
```

O campo *mv* possui dois vetores de movimento, a posição 0 tem os vetores de movimento calculados utilizando a lista 0, a posição 1 tem os vetores calculados utilizando a lista 1. Na implementação atual foram utilizados apenas os vetores de movimento da lista 0. Exemplo do código necessário para varrer todos os vetores de movimento da lista 0:

```
1
2 int blk_y, blk_x;
3
4 for (blk_y=0; blk_y  < ceil(p_Vid->height / BLOCK_SIZE); blk_y++)
5 {
6
7   for(blk_x = 0 ; blk_x < ceil(p_Vid->width / BLOCK_SIZE); blk_x++) {
8
9     PicMotionParams *mv_info_p = &p_Vid->enc_picture->mv_info[blk_y][blk_x
        ];
10
11    printf("Vetor de movimento X da lista 0 para o bloco[%d][%d] = %d\n",
        blk_y, blk_x, mv_info_p->mv[LIST_0].mv_x);
```

```
12      printf("Vetor de movimento Y da lista 0 para o bloco[%d][%d] = %d\n",
            blk_y, blk_x, mv_info_p->mv[LIST_0].mv_y);
13
14    }
15
16  }
```

Exemplos mais completos para entender como trabalhar com os vetores de movimento no software de referência se encontram nas funções da família *GetStrength* que se encontram nas diferentes implementações de *loop_filter* (ex. *loop_filter_normal.c*).

3.4.3 *CONFIGURAÇÕES ADICIONADAS AO CODIFICADOR*

Foram adicionadas novas configurações no codificador do software de referência, para controlar melhor o processo de detecção de objetos. As configurações primeiro têm de ser adicionadas à estrutura *struct inp_par_enc*, que se encontra em *lcommon/inc/params.h*. Os campos adicionados foram:

- object_detection_enable - Habilita ou desabilita a detecção de objetos.
- object_detection_min_width - Comprimento mínimo do objeto para que ele seja detectado.
- object_detection_min_height - Altura mínima do objeto para que ele seja detectado.
- object_detection_search_hysteresis - Histerese (em quadros) da busca de um novo objeto.
- object_detection_tracking_hysteresis - Histerese (em quadros) para confirmar a existência/posição de um objeto que está sendo feito *tracking*.
- object_detection_training_file - Arquivo de treinamento do classificador Haar, define o objeto que será detectado.

Essas configurações também foram adicionadas na estrutura *Map* que se encontra em *lencod/inc/configfile.h*. Tendo feito isso o software de referência automaticamente carregará essas configurações com os valores default definidos na estrutura *Map* e tentará obter esses campos a partir do arquivo de configuração que é informado ao executar o codificador através da opção -f. Os valores definidos no arquivo de configuração são automaticamente carregados na estrutura *InputParameters*.

Um exemplo de como definir essas configurações no arquivo de configuração:

```
1
2 #
    ##############################################################################
3 # Object detection/tracking configuration
4 #
    ##############################################################################
5 object_detection_enable              = 1  # 1 = Enable, 0 = Disable
6 object_detection_min_width           = 30 # Min width of the object that
    will be detected
7 object_detection_min_height          = 30 # Min height of the object that
    will be detected
8 object_detection_search_hysteresis   = 10 # Search for new object
    hysteresys (in frames).
9 object_detection_tracking_hysteresis = 30 # Confirm tracked object
    existence hysteresis (in frames).
10 object_detection_training_file      = "haarcascade_frontalface_alt.xml" #
    File containing the training info used on the object detection.
```

3.5 ALTERAÇÕES REALIZADAS NO DECODIFICADOR

Nesta seção serão apresentadas as alterações realizadas no decodificador do software de referência, para aplicar os metadados recebidos no vídeo decodificado. As alterações visam desenhar nos quadros decodificados os metadados do tipo *ExtractedObjectBoundingBox* presentes no bitstream, facilitando a constatação visual da eficácia do sistema de detecção e *tracking* de objetos implementados no codificador. Para simplificar a implementação do decodificador ele só detecta e aplica um metadado por quadro, mas não existe algo que impossibilite a aplicação de vários metadados em cada quadro.

As alterações se concentram no momento em que um NALU do tipo SEI *Unregistered UserData* é detectado no bitstream, e um pouco antes do quadro decodificado ser gravado no arquivo de saída configurado. As alterações realizadas no decodificador se encontram no anexo B.

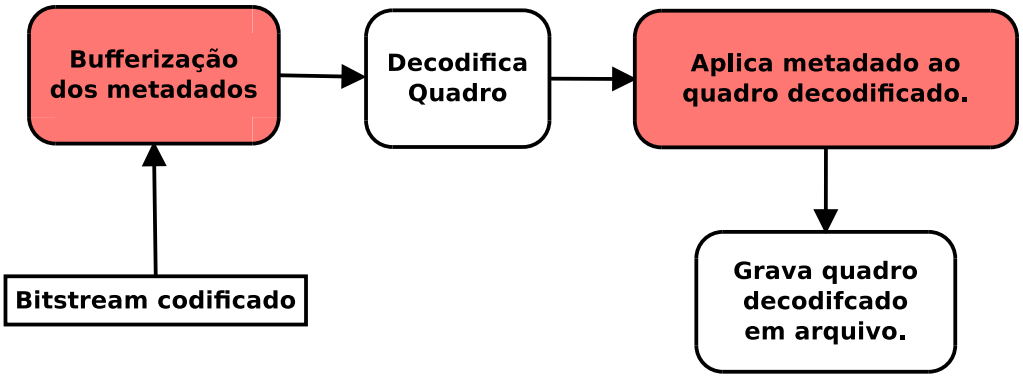

Figura 22: Visão geral do decodificador H.264 modificado. Os blocos vermelhos representam os processos adicionados ao decodificador.

3.5.1 *RECUPERANDO METADADOS A PARTIR DO BITSTREAM*

Para realizar a recuperação de um metadado previamente inserido no bitstream durante o processo de codificação foi adicionado na estrutura *VideoParameters* (definida em *ldecod/inc/global.h*) do decodificador um novo campo chamado *metadata_buffer* do tipo *ExtractedMetadataBuffer*. Esse buffer é inicializado na função *main* do decodificador, que se encontra em *ldecod/src/decoder_test.c*, logo após o decodificador ter sido configurado, mas antes de iniciar o processo de decodificação.

Ao longo do processo de decodificação, sempre que um NALU do tipo SEI é encontrado no bitstream a função *InterpretSEIMessage* é chamada, essa função se encontra em *ldecod/src/sei.c*. Nela existe um *switch case* para os diversos tipos de mensagens SEI possíveis, no *case SEI_USER_DATA_UNREGISTERED* foi adicionada detecção se essa mensagem é um *ExtractedMetadata*. Em caso afirmativo, esse metadado será adicionado ao *ExtractedMetadataBuffer*, caso contrário a mensagem é ignorada.

3.5.2 *APLICANDO O METADADO AO QUADRO*

Depois de receber e bufferizar os metadados foi necessário encontrar um bom lugar para aplicar os metadados ao quadro, de preferência logo antes do quadro ser gravado em arquivo, onde todo o processo de decodificação já teria ocorrido. Foi escolhido o inicio da função *write_out_picture*, que se encontra em *ldecod/src/output.c* como melhor local para realizar a aplicação do metadado.

Para auxiliar o processo de aplicação de metadados do tipo *ExtractedObjectBoundingBox* foi adicionado em *ldecod/src/output.c* a função *decoder_draw_bounding_box*, que verifica se o metadado é do tipo *ExtractedObjectBoundingBox*, em caso afirmativo a caixa delimitadora é

desenhada no quadro, caso contrário o metadado é ignorado.

No inicio da função *write_out_picture* um contador é utilizado para definir o número do quadro (em ordem de apresentação), o número do quadro é utilizado na chamada do método *get* da classe *ExtractedMetadataBuffer*, para verificar a existência de um metadado para aquele quadro. Se um metadado existir para este quadro, será chamada a função *decoder_draw_bounding_box*, que desenhará a caixa delimitadora no quadro.

4 TESTES

Neste capítulo são apresentados os testes realizados no sistema proposto, integrando o classificador Haar ao codificador do H.264. Testes detalhados de desempenho comparando o classificador Haar do OpenCV com outros 2 algoritmos de detecção de objetos, podem ser vistos em [9]. Todos os testes utilizaram o mesmo arquivo de treinamento *haarcascade_frontalface_alt.xml*, que realiza a busca de faces frontais, este arquivo vem junto com a distribuição do OpenCV (que pode ser encontrada em http://sourceforge.net/projects/opencvlibrary/files) no diretório *data/haarcascades*. O tamanho mínimo do objeto de interesse configurado em todos os testes foi de 30 x 30 pixels.

A configuração da máquina e o sistema operacional utilizados nos testes foi:

- Processador Pentium(R) Dual-Core E5200 á 2.50GHz.
- 4 gigabytes de memória RAM.
- Sistema operacional Ubuntu 11.04 32 bits.

Os parâmetros medidos foram os seguintes:

- Desempenho do sistema com *tracking* de objetos X sem *tracking* de objetos.
- Desempenho do sistema com o uso de histerese X sem o uso de histerese.
- Desempenho do sistema com diferentes configurações de histerese.
- Qualidade do *tracking* de objetos em suas diferentes configurações, constatando visualmente a caixa delimitadora desenhada no vídeo.
- Tamanho do bitstream com *tracking* de objetos X sem *tracking* de objetos.

Foram realizados 5 testes em cada vídeo:

- Sem *tracking* de objetos.
- Com *tracking* ativado, com histerese de busca = 1 e histerese de *tracking* = 1. Dessa maneira as informações de estimativa de movimento do codificador não são utilizadas, o *tracking* é realizado utilizando apenas o classificador Haar.
- Com *tracking* ativado, com histerese de busca = 5 e histerese de *tracking* = 10. Essa configuração faz um menor uso das informações de estimativa de movimento para realizar *tracking* de um objeto.
- Com *tracking* ativado, com histerese de busca = 10 e histerese de *tracking* = 30. Essa configuração depende mais das informações de estimativa de movimento para realizar *tracking* de um objeto.
- Com *tracking* ativado, com histerese de busca = 10 e histerese de *tracking* = 60. Essa configuração é a que mais depende mais das informações de estimativa de movimento para realizar *tracking* de um objeto.

Dessa maneira é possível avaliar a diferença entre realizar o *tracking* de um objeto utilizando apenas o classificador Haar, ou utilizando histereses com auxílio das informações de estimativa de movimento. Em todos os testes foi utilizada a mesma configuração de codificação, a única diferença entre as configurações são o arquivo de origem, arquivo de destino, resolução e taxa de apresentação (pois esses parâmetros são diferentes para cada vídeo). No apêndice E encontram-se as configurações que foram alteradas a partir do arquivo de configuração original que vem junto com o software de referência (o arquivo de configuração original se encontra em *bin/encoder.cfg*).

4.1 VÍDEO AKIYO - QCIF - 300 QUADROS

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/akiyo_qcif.y4m e consiste basicamente de 300 quadros positivos (existe o objeto de interesse ao longo de todo o vídeo, nesse caso uma face frontal).

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 300.

- comprimento = 176 pixels.
- altura = 144 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Tempo Total de Codificação (segundos)	35,053	38,889	35,500	35,260	35,190
Tamanho bitstream (bytes)	359220	372974	372790	372560	372561
Atraso no tempo total codificação	0%	10,94%	1,27%	0,59%	0,39%
Aumento bitstream	0%	3,83%	3,77%	3,71%	3,71%

Tabela 1: Desempenho do sistema com o vídeo Akiyo.

Figura 23: Face detectada no vídeo Akiyo.

Como este vídeo é composto de apenas uma pessoa, realizando pequenos movimentos suaves, os resultados com histerese de *tracking* alta foram bons. A ativação de detecção de objetos em todos os quadros, utilizando apenas o classificador Haar para realizar o *tracking*, tornou o processo de codificação aproximadamente 10,94% mais lento.

A utilização do classificador Haar em conjunto com a estimativa de movimento fez com que a caixa delimitadora realizasse movimentos mais suaves, com um atraso variando de 1,27% á 0,39% em relação ao processo de codificação original, dependendo da configuração da histerese. O aumento no bitstream no pior caso foi de 3,83%.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
1
2  Y { PSNR (dB), cSNR (dB), MSE }    : {  41.348,  39.978,   6.53497 }
3  U { PSNR (dB), cSNR (dB), MSE }    : {  49.603,  49.132,   0.79417 }
4  V { PSNR (dB), cSNR (dB), MSE }    : {  51.103,  51.031,   0.51284 }
5
6  Total bits                         : 2873760 (I 1944040, P 929560, NVB 160)
7  Bit rate (kbit/s) @ 30.00 Hz       : 287.38
8  Bits to avoid Startcode Emulation  : 0
9  Bits for parameter sets            : 160
10 Bits for filler data               : 0
```

4.2 VÍDEO *COAST GUARD* - QCIF - 300 QUADROS

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/coastguard_qcif.y4m e consiste basicamente de 300 quadros negativos (não existe o objeto de interesse ao longo de todo o vídeo). Como a histerese de *tracking* não é utilizada neste teste, a configuração com histerese de busca de 10 quadros e histerese de *tracking* de 60 quadros não será apresentada.

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 300.
- comprimento = 176 pixels.
- altura = 144 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**
Tempo Total de Codificação (segundos)	78,200	84,800	79,482	78,896
Tamanho bitstream (bytes)	1240200	1240200	1240200	1240200
Atraso no tempo total codificação	0%	8,43%	1,63%	0,89%
Aumento bitstream	0%	0%	0%	0%

Tabela 2: Desempenho do sistema com o vídeo Coast Guard.

A ativação de detecção de objetos em todos os quadros tornou o processo de codificação aproximadamente 8,43% mais lento. A utilização do classificador Haar com histerese de busca de 10 quadros gerou bons resultados, um atraso de apenas 0,89% em relação ao processo de codificação original. Com uma histerese de busca de 5 quadros o atraso dobrou para 1,63%, ainda sendo um valor bem inferior ao atraso gerado pelo processamento de todos os quadros. Como esse vídeo não possui nenhum objeto de interesse o bitstream teve o mesmo tamanho em todas as configurações, e a histerese de *tracking* não foi nem sequer utilizada.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
1
2  Y { PSNR (dB), cSNR (dB), MSE }    : {  36.174,  34.758,  21.73983 }
3  U { PSNR (dB), cSNR (dB), MSE }    : {  47.488,  46.834,   1.34807 }
4  V { PSNR (dB), cSNR (dB), MSE }    : {  49.023,  48.567,   0.90448 }
5
6  Total bits                         : 9921600 (I 3033832, P 6887608, NVB 160)
7  Bit rate (kbit/s) @ 29.97 Hz       : 991.17
8  Bits to avoid Startcode Emulation  : 0
9  Bits for parameter sets            : 160
10 Bits for filler data               : 0
```

4.3 VÍDEO *CREW* - CIF - 300 QUADROS

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/crew_cif.y4m, possui várias pessoas andando juntas, possuindo diversas faces frontais em uma boa parte do vídeo, andando na direção da câmera (efeito de *zoom*), até que as faces ficam de perfil.

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 300.
- comprimento = 352 pixels.
- altura = 288 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Tempo Total de Codificação (segundos)	341,362	371,870	344,997	342,596	342,376
Tamanho bitstream (bytes)	1245195	1255323	1256008	1256661	1257621
Atraso no tempo total codificação	0%	8,93%	1,06%	0,36%	0,29%
Aumento bitstream	0%	0,81%	0,86%	0,92%	0,99%

Tabela 3: Desempenho do sistema com o vídeo Crew.

Figura 24: Face detectada no vídeo *Crew*.

Como este vídeo é composto de várias pessoas andando juntas, possui diversas faces frontais simultaneamente. Dessa maneira ele exibe duas limitações do sistema, a capacidade de detectar e realizar o tracking de apenas um objeto por vez (é possível realizar o *tracking* de vários objetos diferentes, utilizando extratores com treinamentos diferentes, mas não o *tracking* de vários objetos com mesma forma), e algumas falhas quando um objeto se move rápido demais (a caixa delimitadora se move mais lentamente que o objeto).

Quanto ao desempenho a ativação de detecção de objetos em todos os quadros tornou o processo de codificação 8,93% mais lento. A utilização do classificador Haar em conjunto com a estimativa de movimento diminui consideravelmente o custo computacional, com um atraso no processo de codificação variando entre 1,06% e 0,29% em relação ao processo de codificação original. O bitstream com os metadados inseridos no pior caso ficou 0,99% maior que o bitstream original.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
1 Y { PSNR (dB), cSNR (dB), MSE }    : {  35.790,  35.087,  20.15370 }
2 U { PSNR (dB), cSNR (dB), MSE }    : {  39.443,  38.836,   8.50048 }
3 V { PSNR (dB), cSNR (dB), MSE }    : {  38.490,  37.766,  10.87730 }
4
5 Total bits                         : 10000072 (I 2404808, P 7595096, NVB
    168)
6 Bit rate (kbit/s) @ 30.00 Hz       : 1000.01
7 Bits to avoid Startcode Emulation : 0
8 Bits for parameter sets            : 168
9 Bits for filler data               : 0
```

4.4 VÍDEO FOREMAN - CIF - 300 QUADROS

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/foreman_cif.y4m e possui 3 situações diferentes, uma face frontal, a face fica de perfil em alguns momentos, e depois a face sai do vídeo.

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 300.
- comprimento = 352 pixels.
- altura = 288 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Tempo Total de Codificação (segundos)	260,432	286,162	265,003	262,654	262,126
Tamanho bitstream (bytes)	1244893	1249485	1250345	1251793	1253161
Atraso no tempo total codificação	0%	9,87%	1,75%	0,85%	0,65%
Aumento bitstream	0%	0,36%	0,43%	0,55%	0,66%

Tabela 4: Desempenho do sistema com o vídeo Foreman.

Figura 25: Face detectada no vídeo Foreman.

Com a detecção de objetos ocorrendo em todos os quadros percebe-se uma alta taxa de falsos negativos, já que o rosto no vídeo fica de perfil em alguns momentos e o classificador Haar não consegue identificar o rosto em perfil, perdendo assim o *tracking* do rosto. Com a detecção de objetos em todos os quadros também ocorreu um falso positivo quando a câmera se move e deixa de gravar o rosto. A medida que a histerese de *tracking* é aumentada, além de não ocorrer o falso positivo, os movimentos que o rosto realiza são acompanhados com maior suavidade e com uma menor taxa de falsos negativo (com a histerese de *tracking* configurada para 60 quadros não ocorreu nenhum falso negativo).

Quanto ao desempenho a ativação de detecção de objetos em todos os quadros tornou o processo de codificação 9,87% mais lento. A utilização do classificador Haar em conjunto com a estimativa de movimento diminui consideravelmente o custo computacional, com um atraso no processo de codificação variando entre 1,75% e 0,65% em relação ao processo de codificação original. O bitstream com os metadados inseridos no pior caso ficou 0,66% maior que o bitstream original.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
Y { PSNR (dB), cSNR (dB), MSE }    : {  36.017,  34.568,  22.71311 }
U { PSNR (dB), cSNR (dB), MSE }    : {  41.277,  40.985,   5.18274 }
V { PSNR (dB), cSNR (dB), MSE }    : {  43.175,  42.592,   3.57977 }

Total bits                         : 9959144 (I 2813168, P 7145808, NVB 168)
Bit rate (kbit/s) @ 30.00 Hz       : 995.91
Bits to avoid Startcode Emulation  : 0
Bits for parameter sets            : 168
Bits for filler data               : 0
```

4.5 VÍDEO *PEDESTRIAN AREA* - 375 QUADROS

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/pedestrian_area_-1080p25.y4m. Ele consiste basicamente de vários pedestres caminhando em uma rua, possui faces frontais e faces em perfil se movendo continuamente.

4.5.1 *TESTES COM RESOLUÇÃO 1080P (1920 X 1080)*

Especificações do vídeo:

- quadros por segundo = 25.
- total de quadros = 375.
- comprimento = 1920 pixels.
- altura = 1080 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Tempo Total de Codificação (segundos)	6915.861	7280.818	6921.871	6918.653	6916.272
Tamanho bitstream (bytes)	7854045	7870345	7870468	7870642	7870601
Atraso no tempo total codificação	0%	5,27%	0,08%	0,04%	0,005%
Aumento bitstream	0%	0,207%	0,209%	0,211%	0,210%

Tabela 5: Desempenho do sistema com o vídeo *Pedestrian area*.

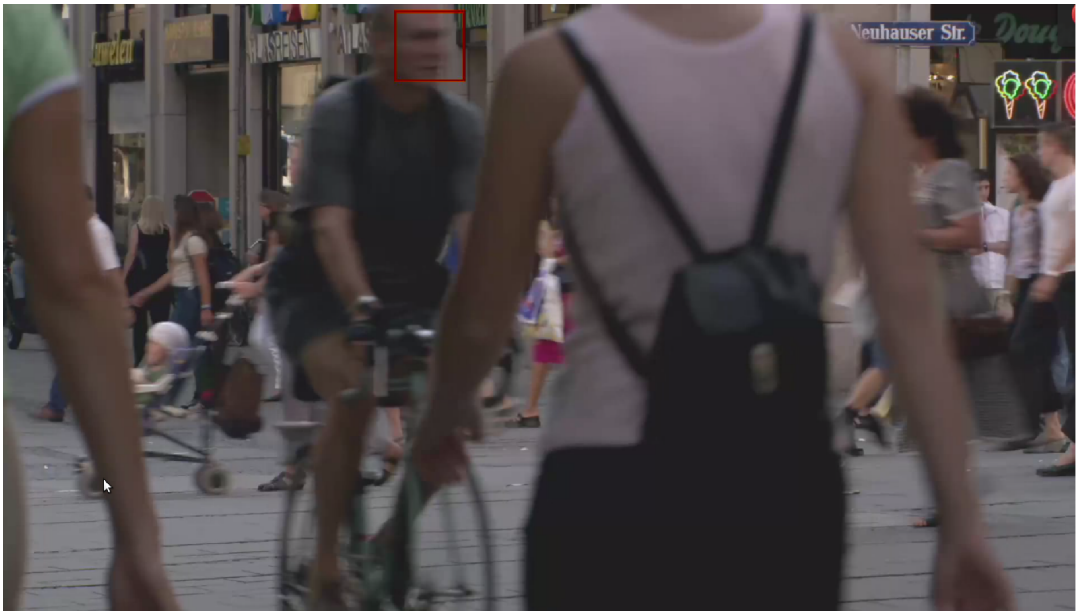

Figura 26: Face detectada no vídeo *Pedestrian area*.

Com a detecção de objetos ocorrendo em todos os quadros ocorre uma grande quantidade de falsos positivos, e a limitação de realizar o *tracking* de apenas um objeto por vez ficou bem evidente já que neste vídeo existem diversas pessoas andando ao mesmo tempo o sistema consegue realizar o *tracking* de algumas faces por um curto período de tempo, mas logo depois detecta outra face e passa a realizar o *tracking* dessa face ao invés da que foi previamente detectada.

A utilização das histereses alterou um pouco o comportamento do sistema (em geral ele manteve uma alta taxa de falsos positivos), a primeira detecção que ocorre com sucesso realiza o *tracking* da face corretamente, mas após essa primeira detecção ocorre uma série de falsos positivos, e em algumas detecções positivas o *tracking* não consegue acompanhar a face por ela estar se movendo rápido demais.

Quanto ao desempenho a ativação de detecção de objetos em todos os quadros tornou o processo de codificação 5,27% mais lento. A utilização do classificador Haar em conjunto com a estimativa de movimento diminui consideravelmente o custo computacional, com um atraso no processo de codificação inferior a 0,1% em todas as configurações de histerese testadas. O bitstream com os metadados inseridos no pior caso ficou 0,211% maior que o bitstream original.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
1 Y { PSNR (dB), cSNR (dB), MSE }    : {  37.196,  37.136,  12.57412 }
2 U { PSNR (dB), cSNR (dB), MSE }    : {  41.390,  41.356,   4.75815 }
3 V { PSNR (dB), cSNR (dB), MSE }    : {  42.574,  42.530,   3.63104 }
4
```

```
5 Total bits                          : 62832360 (I 11049512, P 51782664, NVB
    184)
6 Bit rate (kbit/s) @ 25.00 Hz        : 4188.82
7 Bits to avoid Startcode Emulation : 0
8 Bits for parameter sets             : 184
9 Bits for filler data                : 0
```

4.5.2 *TESTES COM RESOLUÇÃO 720P (1280 X 720)*

Neste teste a resolução do vídeo que foi reduzida de *1080p* (1920 x 1080) para *720p* (1280 x 720), visando a constatação da diferença do impacto do sistema de *tracking* ao processar o mesmo vídeo com resoluções diferentes. O escalonamento do vídeo foi realizado utilizando o Gstreamer.

Especificações do vídeo:

- quadros por segundo = 25.
- total de quadros = 375.
- comprimento = 1280 pixels.
- altura = 720 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Tempo Total de Codificação (segundos)	2986.361	3317.612	3011.626	2999.675	2993.329
Tamanho bitstream (bytes)	4269076	4283747	4284989	4283815	4284129
Atraso no tempo total codificação	0%	11,09%	0,84%	0,44%	0,23%
Aumento bitstream	0%	0,34%	0,37%	0,34%	0,35%

Tabela 6: Desempenho do sistema com o vídeo *Pedestrian area - 720p*.

A diminuição da resolução do vídeo não gerou nenhuma diferença na qualidade do *tracking*, porém foi possível perceber um aumento no atraso no tempo total de codificação em todas as configurações. Com *tracking* habilitado e sem histerese, o atraso no vídeo original (*1080p*) foi de 5,27% enquanto que o atraso gerado no mesmo vídeo, reescalonado para *720p*, foi de 11,09%, mostrando que a medida que o tamanho do vídeo aumenta o atraso gerado pelo classificador Haar tende a diminuir.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
1 Y { PSNR (dB), cSNR (dB), MSE }    : {  36.269,  36.196,  15.61449 }
2 U { PSNR (dB), cSNR (dB), MSE }    : {  40.982,  40.944,   5.23199 }
3 V { PSNR (dB), cSNR (dB), MSE }    : {  41.777,  41.730,   4.36622 }
4
5 Total bits                         : 34152608 (I 6300464, P 27851968, NVB
    176)
6 Bit rate (kbit/s) @ 25.00 Hz       : 2276.84
7 Bits to avoid Startcode Emulation : 0
8 Bits for parameter sets            : 176
9 Bits for filler data               : 0
```

4.6 VÍDEO *SPEED BAG* - 570 QUADROS

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/speed_bag_1080p.y4m. Uma parte do vídeo possui uma face frontal realizando pequenos movimentos. O vídeo original se encontra com espaço de cor YUV 4:2:2, porém para facilitar o procedimento de teste e a comparação com os resultados de outros vídeos ele foi convertido para o espaço de cor YUV 4:2:0, utilizando o Gstreamer. O vídeo consiste de um face de perfil se movendo (pessoa caminhando), nenhuma face, uma face frontal realizando movimentos suaves e depois uma face de perfil realizando movimentos rápidos (pugilista socando um saco de pancada).

4.6.1 *TESTES COM RESOLUÇÃO 1080P (1920 X 1080)*

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 570.
- comprimento = 1920 pixels.
- altura = 1080 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Tempo Total de Codificação (segundos)	12719.172	13689.180	12867.480	12821.436	12765.063
Tamanho bitstream (bytes)	8409840	8425882	8428471	8428672	8430918
Atraso no tempo total codificação	0%	7,62%	1,16%	0,8%	0,36%
Aumento bitstream	0%	0,19%	0,22%	0,22%	0,25%

Tabela 7: Desempenho do sistema com o vídeo *Speed bag - 1080p*.

Figura 27: Face detectada no vídeo *Speed bag*.

Utilizando a detecção de objetos em todos os quadros ocorrem diversos falsos positivos, a maior parte dos momentos em que existe a face de perfil ocorrem falsos positivos ou falsos negativos. O momento em que ocorre a face frontal no vídeo possui uma maior quantidade de

positivos, porém em alguns momentos ocorrem falsos positivos (um objeto no fundo é detectado como a face), mostrando claramente a limitação que existe ao se configurar o classificador Haar para realizar a detecção de apenas um objeto na imagem. O *tracking* pode se confundir ao detectar outro objeto que não é uma face, já que apenas o primeiro objeto a ser encontrado na imagem é retornado pelo classificador Haar.

A utilização de histereses ofereceu pouca melhora na qualidade do *tracking*, já que neste vídeo a quantidade de falsos positivos foi muito grande. Apenas no momento em que ocorria a face frontal a histerese mostrou uma melhora significativa, já que a face frontal era detectada e durante a histerese de *tracking* apenas os vetores de movimento calculados pela estimativa de movimento era utilizados para realizar o *tracking*, reduzindo os falsos positivos gerados pelo classificador Haar.

Quanto ao desempenho a ativação de detecção de objetos em todos os quadros tornou o processo de codificação 7,62% mais lento. A utilização do classificador Haar em conjunto com a estimativa de movimento diminui consideravelmente o custo computacional, com um atraso no processo de codificação variando entre 1,16% e 0,36% em relação ao processo de codificação original. O bitstream com os metadados inseridos no pior caso ficou 0,25% maior que o bitstream original.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
1 Y { PSNR (dB), cSNR (dB), MSE }    : {  39.554,  39.283,   7.67013 }
2 U { PSNR (dB), cSNR (dB), MSE }    : {  43.495,  43.394,   2.97618 }
3 V { PSNR (dB), cSNR (dB), MSE }    : {  44.830,  44.669,   2.21926 }
4
5 Total bits                         : 67278720 (I 9809312, P 57469224, NVB
    184)
6 Bit rate (kbit/s) @ 30.00 Hz       : 3540.99
7 Bits to avoid Startcode Emulation : 0
8 Bits for parameter sets            : 184
9 Bits for filler data               : 0
```

4.6.2 *TESTES COM RESOLUÇÃO 720P (1280 X 720)*

Neste teste a resolução do vídeo que foi reduzida de *1080p* (1920 x 1080) para *720p* (1280 x 720), visando a constatação da diferença do impacto do sistema de *tracking* ao processar o mesmo vídeo a resoluções diferentes. O escalonamento do vídeo foi realizado utilizando o Gstreamer.

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 570.
- comprimento = 1280 pixels.
- altura = 720 pixels.
- espaço de cor = YUV, 4:2:0.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Tempo Total de Codificação (segundos)	5346.245	5843.674	5394.242	5365.767	5352.772
Tamanho bitstream (bytes)	4595164	4607265	4609359	4612805	4616437
Atraso no tempo total codificação	0%	9,3%	0,89%	0,36%	0,12%
Aumento bitstream	0%	0,26%	0,30%	0,38%	0,46%

Tabela 8: Desempenho do sistema com o vídeo *Speed bag - 720p*.

A diminuição da resolução do vídeo não gerou nenhuma diferença na qualidade do *tracking*, porém foi possível perceber um aumento no atraso no tempo total de codificação em todas as configurações. Com *tracking* habilitado e sem histerese, o atraso no vídeo original (*1080p*) foi de 7,62% enquanto que o atraso gerado no mesmo vídeo, reescalonado para *720p*, foi de 9,3%, mostrando que a medida que o tamanho do vídeo aumenta o atraso gerado pelo classificador Haar tende a diminuir.

Em todos as configurações a qualidade do vídeo codificado mostrou ser a mesma:

```
Y { PSNR (dB), cSNR (dB), MSE }    : {  39.131,  38.752,   8.66746 }
U { PSNR (dB), cSNR (dB), MSE }    : {  43.023,  42.893,   3.34007 }
V { PSNR (dB), cSNR (dB), MSE }    : {  44.422,  44.220,   2.46079 }

Total bits                         : 36761312 (I 5554416, P 31206720, NVB
    176)
Bit rate (kbit/s) @ 30.00 Hz       : 1934.81
Bits to avoid Startcode Emulation : 0
Bits for parameter sets            : 176
Bits for filler data               : 0
```

4.7 CONCLUSÃO DA AVALIAÇÃO DO SISTEMA

De acordo com os testes realizados a utilização de uma histerese na execução do classificador Haar tornou o sistema mais eficiente. Se somente fosse utilizada a histerese, isso geraria saltos na caixa delimitadora, o uso da estimativa de movimento suavizou esse efeito sem aumentar a complexidade do codificador, já que a estimativa de movimento faz parte do processo normal de codificação. Dependendo da natureza do movimento realizado pelo objeto, a utilização da estimativa de movimento pode gerar resultados inferiores ou superiores á execução do classificador Haar em todos os quadros.

Por exemplo, nos testes apresentados o objeto de interesse era face frontal. Sem utilizar a estimativa de movimento quando uma face se vira de lado o classificador Haar não consegue mais detectar a face que está de perfil, porém utilizando estimativa de movimento o tracking da face de perfil funciona já que uma vez identificado o objeto só é necessário acompanhar seus movimentos. Ao mesmo tempo, dependendo do quanto o objeto de interesse se mover a caixa delimitadora não acompanha perfeitamente o movimento do objeto utilizando as informações de estimativa de movimento.

Em vídeos com longos trechos sem nenhum objeto de interesse o impacto do classificador Haar tende a ser um pouco menor que o normal, mas em conjunto com a histerese de busca o custo computacional foi reduzido. Inserir um metadado por quadro, mesmo onde o vídeo possui o objeto de interesse em grande parte dos seus quadros, não gerou um bitstream muito maior que o normal, o pior caso não passou de um aumento de 4%, considerando que quanto maior for a resolução do vídeo e a qualidade do processo de codificação, menor será esse aumento em relação ao bitstream original (isso pode ser observado nos testes com diferentes resoluções).

Com relação ao impacto da utilização do classificador Haar integrado ao codificador em

diferentes resoluções, foi possível observar que a medida que a resolução do vídeo aumenta, o custo computacional do processo de codificação tende a aumentar mais que o custo computacional do classificador Haar, já que nos testes em vídeos com alta resolução o impacto da utilização do sistema de *tracking* foi menor que nos vídeos com resoluções menores. Esse resultado foi obtido com o perfil de codificação utilizado nos testes, seria interessante realizar testes com outros perfis de codificação, pois poderiam mostrar resultados diferentes.

Percebe-se também a necessidade de realizar a detecção e *tracking* de múltiplos objetos (que sejam do mesmo padrão, ou seja, utilizem o mesmo arquivo de treinamento para realizar a busca) simultaneamente, como pode ser constatado em [5] página 507, o classificador Haar tende a ter uma baixa taxa de falsos negativos (ou seja, não detectar a presença de um objeto de interesse), em troca de ter uma alta taxa de falsos positivos (detectar um objeto que não é o de interesse), por causa disso utilizar o classificador Haar para retornar o primeiro objeto encontrado simplifica o sistema mas deixa o *tracking* mais suscetível a erros.

5 CONCLUSÕES

Para desenvolver o projeto foi escolhido o software de referência para o padrão de compressão de vídeo MPEG 4 parte 10. Dois tipos de metadados foram desenvolvidos ao longo do trabalho, um representa a caixa delimitadora de um objeto de interesse, o outro é o plano luma bruto de um objeto de interesse.

Testes realizados tanto com o decodificador de referência como com o Gstreamer mostraram que o uso de mensagens *Supplemental Enhancement Information* do tipo *Unregistered Userdata* para transportar os metadados diretamente no *bitstream* do vídeo não alteraram a conformidade do mesmo com o padrão, sendo possível exibir um vídeo com metadados embutidos em qualquer decodificador MPEG 4 parte 10.

Apesar do padrão MPEG 4 parte 10 não definir um tamanho máximo para mensagens *Supplemental Enhancement Information* do tipo *Unregistered Userdata*, pode existir um limite implícito no tamanho máximo de um NALU (de acordo com o nível/perfil implementado), o que refletiria em um tamanho máximo para as mensagens.

Para facilitar a detecção de objetos foi utilizado um algoritmo de detecção de padrões em imagens estáticas, o classificador Haar, implementado na biblioteca de visão computacional OpenCV. Executar o classificador Haar em todos os quadros do vídeo mostrou um aumento considerável no custo computacional do codificador.

Utilizou-se informações de estimativa de movimento calculadas pelo codificador para estimar o movimento do objeto, evitando a necessidade de executar o classificador Haar em todos os quadros do vídeo, dessa maneira constatou-se uma significativa redução do custo computacional reutilizando informações geradas pelo processo de codificação. Com as alterações realizadas no decodificador de referência foi possível recuperar os metadados. Os metadados representando a caixa delimitadora de um objeto detectado, são desenhados no quadro, facilitando a constatação visual do funcionamento do sistema.

Como toda a extração do metadado e inserção dele dentro do *bitstream* é feita internamente no codificador isso facilita a construção de um chip codificador H.264 que realize *tracking* de

objetos, esse chip codificador poderia ter grande parte do classificador Haar acelerado também em hardware, dentro do chip. Como pode ser visto em [8], obtêm-se um grande aumento no desempenho da detecção de objetos ao se implementar o classificador Haar em FPGA.

Este trabalhou mostrou a viabilidade de se utilizar a estimativa de movimento gerada pelo codificador para auxiliar o *tracking* de objetos e do envio dessas informações através do *bitstream* do vídeo. Além das informações de *tracking* foi possível enviar o objeto detectado não compactado como metadado, o que pode ser útil em algoritmos de identificação biométrica. No momento da análise dos vídeos é necessário analisar apenas os metadados que estão inseridos no *bitstream*, grande parte do processamento já foi realizado durante o processo de codificação.

5.1 TRABALHOS FUTUROS

Como possíveis trabalhos futuros, cita-se:

- Fazer um melhor uso das informações geradas pelo processo de codificação para gerar heurísticas mais inteligentes. Um exemplo seria utilizar histereses dinâmicas, quando existe muito movimento no vídeo as histereses diminuem, mas quando não existe movimento as histereses aumentam.
- Estender o sistema para realizar a detecção e *tracking* de múltiplos objetos (com mesma forma) simultaneamente.
- Buscar no classificador Haar cálculos que já possam ter sido feitos pelo codificador, melhorando a integração dos dois.
- Utilizar outro algoritmo de detecção de padrões que tenha uma maior interseção com os algoritmos presentes no codificador.
- Utilizar apenas as informações de estimativa de movimento para a construção de cercas virtuais ou alarmes que não se importem com a forma do objeto mas com padrões de movimento suspeitos.
- Desenvolver um chip codificador H.264 integrado ao classificador Haar integrado no chip, realizando a detecção e *tracking* de objetos em tempo real. Cita-se [8] como exemplo de um classificador Haar acelerado em hardware.

REFERÊNCIAS

[1] LAI, S. S. Z. e S. H. *FACE DETECTION DIRECTLY FROM H.264 COMPRESSED VIDEO WITH CONVOLUTIONAL NEURAL NETWORK*. [S.l.]: IEEE, 2009.

[2] RICHARDSON, I. *The H.264 Advanced Video Compression Standard*. [S.l.]: John Wiley and Sons, 2010.

[3] RECOMMENDATION H.264 - Advanced Video Coding (03/2010). [S.l.]: ITU-T and ISO/IEC, 2010.

[4] IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 13, NO.7, JULY 2003. [S.l.]: IEEE, 2003.

[5] KAEHLER, G. B. e A. *Learning OpenCV*. [S.l.]: O'Really, 2008.

[6] JONES, P. V. e M. *Rapid Object Detection Using a Boosted Cascade of Simple Features*. [S.l.]: IEEE CVPR, 2001.

[7] MAYDT, R. L. e J. *An Extended Set of Haar-like Features for Rapid Object Detection*. [S.l.]: IEEE ICIP, 2002.

[8] LU, C. G. e S.-L. *Novel FPGA Based Haar Classifier Face Detection Algorithm Acceleration*. [S.l.]: IEEE, 2008.

[9] BOLME, M. S. e. J. R. B. D. S. *FacePerf: Benchmarks for Face Recognition Algorithms*. [S.l.]: IEEE, 2007.

6 APÊNDICE A – CODIGO FONTE DO PROCEDIMENTO DE TESTE DA INSERÇÃO DE NALUS SEI

No caso dos módulos novos, todo o seu fonte é documentado aqui.

No caso dos módulos que já existiam no software de referência e sofreram alterações, apenas as funções que sofreram alteração serão documentadas aqui.

6.1 MÓDULO ADICIONADO AO CODIFICADOR

6.1.1 *ARQUIVO UDATA_GEN.H*

```
1  /*!
2   ************************************************************************
3   *  \file
4   *     udata_gen.h
5   *  \brief
6   *     definitions for Supplemental Enhanced Information Userdata
        Generation
7   *  \author(s)
8   *      - Tiago Katcipis                          <tiagokatcipis@gmail.
        com>
9   *
10  *
        ************************************************************************
11  */
12
13 #ifndef UDATA_GEN_H
14 #define UDATA_GEN_H
15
16 #include "typedefs.h"
17 #include "nal.h"
18
```

```
19 /*!
20  ************************************************************************

21  * Generates a SEI NALU that with ONE unregistered userdata SEI message.
22  *
23  * @param data The data to be sent on the SEI unregistered userdata message
        .
24  * @param size The size of the data.
25  * @return A SEI NALU containing the SEI message.
26  *
27  ************************************************************************

28  */
29 NALU_t *user_data_generate_unregistered_sei_nalu(char * data, unsigned int
      size);
30
31 /*!
32  ************************************************************************

33  * Generates a SEI NALU that with ONE unregistered userdata SEI message.
34  *
35  * @param msg The message to be sent on the SEI unregistered userdata
        message,
36  *            must be a 0 terminated string.
37  * @return A SEI NALU containing the SEI message.
38  *
39  ************************************************************************

40  */
41 NALU_t *user_data_generate_unregistered_sei_nalu_from_msg(char * msg);
42
43 /*!
44  ************************************************************************

45  * Generates a 0 terminated string to be sent on a SEI NALU. The size of
        the message is
46  * random, but it will not exceed MAXNALUSIZE.
47  *
48  * @return a 0 terminated string.
49  *
50  ************************************************************************

51  */
```

```
52 char* user_data_generate_create_random_message();
53
54 /*!
55  ********************************************************************************

56  * Frees the resources used by a msg.
57  *
58  * @param msg A previously created message.
59  *
60  ********************************************************************************

61  */
62 void user_data_generate_destroy_random_message(char * msg);
63
64 #endif
```

6.1.2 *ARQUIVO UDATA_GEN.C*

```
1
2  #include <string.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include "udata_gen.h"
6  #include "vlc.h"
7  #include "sei.h"
8  #include "nalu.h"
9
10
11 static const char* random_message_start_template = "\nRandom message[%d]
       start\n";
12 static const char* random_message_end_template   = "\nRandom message[%d]
       end!\n";
13 static int msg_counter                           = 1;
14
15 /* Lets guarantee that max_msg_size + headers +
16    emulation prevention bytes dont exceed the MAXNALUSIZE */
17 static const int max_msg_size                    = MAXNALUSIZE - 1024;
18
19 static int growth_rate                           = 1024;
20 static int actual_size                           = 1024;
21
22 #define MAX_TEMPLATE_MSG_SIZE 512
```

```
23 static char random_message_start_buffer[MAX_TEMPLATE_MSG_SIZE];
24 static char random_message_end_buffer[MAX_TEMPLATE_MSG_SIZE];
25
26 /*!
27  ************************************************************************************

28  * \brief
29  *    int GenerateUserDataSEImessage_rbsp(int, byte*, char*, unsigned int)
30  *
31  * \return
32  *    size of the RBSP in bytes, negative in case of an error
33  *
34  ************************************************************************************

35  */
36 static int GenerateUserDataSEImessage_rbsp (int id, byte *rbsp, char*
      sei_message, unsigned int sei_message_size)
37 {
38   Bitstream *bitstream;
39   unsigned int message_size = sei_message_size;
40
41   int LenInBytes;
42   assert (rbsp != NULL);
43
44   if ((bitstream=calloc(1, sizeof(Bitstream)))==NULL)
45     no_mem_exit("SeqParameterSet:bitstream");
46
47   // .. and use the rbsp provided (or allocated above) for the data
48   bitstream->streamBuffer = rbsp;
49   bitstream->bits_to_go = 8;
50
51   {
52     char uuid_message[9] = "Random"; // This is supposed to be Random
53     unsigned int i;
54
55     TIME_T start_time;
56     gettime(&start_time);     // start time
57
58     u_v (8,"SEI: last_payload_type_byte", 5, bitstream);
59     message_size += 17;
60     while (message_size > 254)
61     {
62       u_v (8,"SEI: ff_byte",255, bitstream);
```

```
63          message_size -= 255;
64        }
65        u_v (8,"SEI: last_payload_size_byte",message_size, bitstream);
66
67        // Lets randomize uuid based on time
68        u_v (32,"SEI: uuid_iso_iec_11578",(int) start_time.tv_sec, bitstream);
69        u_v (32,"SEI: uuid_iso_iec_11578",(int) start_time.tv_usec, bitstream);
70
71        u_v (32,"SEI: uuid_iso_iec_11578",(int) (uuid_message[0] << 24) + (
              uuid_message[1] << 16)  + (uuid_message[2] << 8) + (uuid_message[3]
             << 0), bitstream);
72        u_v (32,"SEI: uuid_iso_iec_11578",(int) (uuid_message[4] << 24) + (
              uuid_message[5] << 16)  + (uuid_message[6] << 8) + (uuid_message[7]
             << 0), bitstream);
73
74        for (i = 0; i < sei_message_size; i++) {
75            u_v (8,"SEI: user_data_payload_byte",sei_message[i], bitstream);
76        }
77
78        /* FIXME we MUST have this zero or the original coded was suposed to
              zero terminate the msg ? */
79        u_v (8,"SEI: user_data_payload_byte", 0, bitstream);
80      }
81
82      SODBtoRBSP(bitstream);      // copies the last couple of bits into the
           byte buffer
83      LenInBytes=bitstream->byte_pos;
84
85      free(bitstream);
86      return LenInBytes;
87    }
88
89
90    /*!
91     ************************************************************************

92     * \brief
93     *    Function body for Unregistered userdata SEI message NALU generation
94     *
95     * \return
96     *    A NALU containing the SEI message.
97     *
```

```
98   ************************************************************************
99   */
100  NALU_t * user_data_generate_unregistered_sei_nalu(char * data, unsigned int
         size)
101  {
102    NALU_t *n = AllocNALU(MAXNALUSIZE);
103    int RBSPlen = 0;
104    byte rbsp[MAXRBSPSIZE];
105
106    RBSPlen = GenerateUserDataSEImessage_rbsp (NORMAL_SEI, rbsp, data, size);
107    RBSPtoNALU (rbsp, n, RBSPlen, NALU_TYPE_SEI, NALU_PRIORITY_DISPOSABLE, 1)
           ;
108
109    n->startcodeprefix_len = 4;
110    return n;
111  }
112
113
114  /*!
115   ************************************************************************
116   * \brief
117   *    Function body for Unregistered userdata SEI message NALU generation
118   *
119   * \return
120   *    A NALU containing the SEI message.
121   *
122   ************************************************************************
123   */
124  NALU_t *user_data_generate_unregistered_sei_nalu_from_msg(char * msg)
125  {
126      return user_data_generate_unregistered_sei_nalu(msg, strlen(msg));
127  }
128
129
130  /*!
131   ************************************************************************
132   * \brief
133   *    Function body for random string message generation.
134   *
```

```
135  * \return
136  *    A 0 terminted string.
137  *
138  ********************************************************************************

139  */
140 char* user_data_generate_create_random_message()
141 {
142     char * msg     = 0;
143     int msg_size   = 0;
144     int body_size  = 0;
145     int start_len  = 0;
146     int end_len    = 0;
147     int i          = 0;
148
149     /* This will help debug of the SEI messages at the decoder */
150     snprintf (random_message_start_buffer, MAX_TEMPLATE_MSG_SIZE,
            random_message_start_template, msg_counter);
151     snprintf (random_message_end_buffer, MAX_TEMPLATE_MSG_SIZE,
            random_message_end_template, msg_counter);
152     msg_counter++;
153
154     start_len = strlen(random_message_start_buffer);
155     end_len  = strlen(random_message_end_buffer);
156
157     body_size = actual_size;
158     actual_size += growth_rate;
159
160     if (actual_size > max_msg_size) {
161         actual_size = growth_rate;
162     }
163
164     msg_size = body_size + start_len +  end_len + 1;
165     msg      = malloc(msg_size);
166
167     /* writing the start */
168     memcpy(msg, random_message_start_buffer, start_len);
169
170     /* writing the body */
171     for (i = start_len; i < msg_size - end_len; i++) {
172         msg[i] = 'h'; /* whatever */
173     }
174
```

```
175     /* writing the end */
176     memcpy(msg + start_len + body_size, random_message_end_buffer, end_len)
            ;
177
178     /* 0 terminate it */
179     msg[msg_size - 1] = '\0';
180     return msg;
181 }
182
183 /*!
184  ************************************************************************

185  * \brief
186  *     Function body for random message destruction.
187  ************************************************************************

188  */
189 void user_data_generate_destroy_random_message(char * msg)
190 {
191     free(msg);
192 }
```

6.2 ALTERAÇÕES NO CODIFICADOR

6.2.1 *ARQUIVO FILEHANDLE.C*

```
1
2  /*!
3   ************************************************************************
4   * \brief
5   *     This function opens the output files and generates the
6   *     appropriate sequence header
7   ************************************************************************
8   */
9  int start_sequence(VideoParameters *p_Vid, InputParameters *p_Inp)
10 {
11   int i,len=0, total_pps = (p_Inp->GenerateMultiplePPS) ? 3 : 1;
12   NALU_t *nalu;
13
14   switch(p_Inp->of_mode)
15   {
16     case PAR_OF_ANNEXB:
```

```
17          OpenAnnexbFile (p_Vid, p_Inp->outfile);
18          p_Vid->WriteNALU = WriteAnnexbNALU;
19          break;
20        case PAR_OF_RTP:
21          OpenRTPFile (p_Vid, p_Inp->outfile);
22          p_Vid->WriteNALU = WriteRTPNALU;
23          break;
24        default:
25          snprintf(errortext, ET_SIZE, "Output File Mode %d not supported",
                p_Inp->of_mode);
26          error(errortext,1);
27      }
28
29      // Access Unit Delimiter NALU
30      if ( p_Inp->SendAUD )
31      {
32        len += Write_AUD_NALU(p_Vid);
33      }
34
35      //! As a sequence header, here we write both sequence and picture
36      //! parameter sets.  As soon as IDR is implemented, this should go to the
37      //! IDR part, as both parsets have to be transmitted as part of an IDR.
38      //! An alternative may be to consider this function the IDR start
            function.
39
40      nalu = NULL;
41      nalu = GenerateSeq_parameter_set_NALU (p_Vid);
42      len += p_Vid->WriteNALU (p_Vid, nalu);
43      FreeNALU (nalu);
44
45    #if (MVC_EXTENSION_ENABLE)
46      if(p_Inp->num_of_views==2)
47      {
48        int bits;
49        nalu = NULL;
50        nalu = GenerateSubsetSeq_parameter_set_NALU (p_Vid);
51        bits = p_Vid->WriteNALU (p_Vid, nalu);
52        len += bits;
53        p_Vid->p_Stats->bit_ctr_parametersets_n_v[1] = bits;
54        FreeNALU (nalu);
55      }
56      else
57      {
```

```
58      p_Vid->p_Stats->bit_ctr_parametersets_n_v[1] = 0;
59    }
60  #endif
61
62    //! Lets write now the Picture Parameter sets. Output will be equal to
          the total number of bits spend here.
63    for (i=0;i<total_pps;i++)
64    {
65      len = write_PPS(p_Vid, len, i);
66    }
67
68    if (p_Inp->GenerateSEIMessage)
69    {
70      nalu = NULL;
71      nalu = GenerateSEImessage_NALU(p_Inp);
72      len += p_Vid->WriteNALU (p_Vid, nalu);
73      FreeNALU (nalu);
74    }
75
76    /* Lets send 1000 SEI NALUs containing a SEI Userdata Unregistered
          message */
77    for (i = 0; i < 1000; i++) {
78        nalu = NULL;
79        char * msg = user_data_generate_create_random_message();
80        nalu = user_data_generate_unregistered_sei_nalu_from_msg(msg);
81        len += p_Vid->WriteNALU (p_Vid, nalu);
82        FreeNALU (nalu);
83        user_data_generate_destroy_random_message(msg);
84    }
85
86
87    p_Vid->p_Stats->bit_ctr_parametersets_n = len;
88  #if (MVC_EXTENSION_ENABLE)
89    if(p_Inp->num_of_views==2)
90    {
91      p_Vid->p_Stats->bit_ctr_parametersets_n_v[0] = len - p_Vid->p_Stats->
            bit_ctr_parametersets_n_v[1];
92    }
93  #endif
94    return 0;
95  }
```

6.2.2 *ARQUIVO LENCOD.C*

```
1  /*!
2   ************************************************************************
3   * \brief
4   *    Encode a sequence
5   ************************************************************************
6   */
7  static void encode_sequence(VideoParameters *p_Vid, InputParameters *p_Inp)
8  {
9    int curr_frame_to_code;
10   int frames_to_code;
11   int frame_num_bak = 0, frame_coded;
12   int frm_struct_buffer;
13   SeqStructure *p_seq_struct = p_Vid->p_pred;
14   FrameUnitStruct *p_frm;
15
16 #if (MVC_EXTENSION_ENABLE)
17   int tmp_rate_control_enable = p_Inp->RCEnable;
18
19   if ( p_Inp->num_of_views == 2 )
20   {
21     frames_to_code = p_Inp->no_frames << 1;
22     p_frm = p_seq_struct->p_frm_mvc;
23     frm_struct_buffer = p_seq_struct->num_frames_mvc;
24   }
25   else
26 #endif
27   {
28     frames_to_code = p_Inp->no_frames;
29     p_frm = p_seq_struct->p_frm;
30     frm_struct_buffer = p_Vid->frm_struct_buffer;
31   }
32
33   for (curr_frame_to_code = 0; curr_frame_to_code < frames_to_code;
       curr_frame_to_code++)
34   {
35 #if (MVC_EXTENSION_ENABLE)
36     if ( p_Inp->num_of_views == 2 )
37     {
38       if ( (curr_frame_to_code & 1) == 0 ) // call only for view_id 0
39       {
```

```
40          // determine whether to populate additional frames in the
                prediction structure
41          if ( (curr_frame_to_code >> 1) >= p_Vid->p_pred->pop_start_frame )
42          {
43            int start = p_seq_struct->pop_start_frame, end;
44
45            populate_frm_struct( p_Vid, p_Inp, p_seq_struct, p_Inp->
                  FrmStructBufferLength, frames_to_code >> 1 );
46            end = p_seq_struct->pop_start_frame;
47            populate_frm_struct_mvc( p_Vid, p_Inp, p_seq_struct, start, end )
                  ;
48          }
49        }
50      }
51      else
52  #endif
53      {
54        // determine whether to populate additional frames in the prediction
              structure
55        if ( curr_frame_to_code >= p_Vid->p_pred->pop_start_frame )
56        {
57          populate_frm_struct( p_Vid, p_Inp, p_seq_struct, p_Inp->
                FrmStructBufferLength, frames_to_code );
58        }
59      }
60
61      p_Vid->curr_frm_idx = curr_frame_to_code;
62      p_Vid->p_curr_frm_struct = p_frm + ( p_Vid->curr_frm_idx %
            frm_struct_buffer ); // pointer to current frame structure
63      p_Vid->number = curr_frame_to_code;
64
65  #if (MVC_EXTENSION_ENABLE)
66      if(p_Inp->num_of_views==2)
67      {
68        p_Vid->view_id = p_Vid->p_curr_frm_struct->view_id;
69        if ( p_Vid->view_id == 1 )
70        {
71          p_Vid->curr_frm_idx = p_Vid->number = (curr_frame_to_code - 1) >>
                1;
72          p_Vid->p_curr_frm_struct->qp = p_Vid->qp = iClip3( -p_Vid->
                bitdepth_luma_qp_scale, MAX_QP, p_Vid->AverageFrameQP + p_Inp->
                View1QPOffset );
73        }
```

```
74          else
75          {
76            p_Vid->curr_frm_idx = p_Vid->number = curr_frame_to_code >> 1;
77          }
78          if ( p_Vid->view_id == 1 && tmp_rate_control_enable )
79          {
80            p_Inp->RCEnable = 0;
81          }
82          else
83          {
84            p_Inp->RCEnable = tmp_rate_control_enable;
85          }
86        }
87    #endif
88
89        if ( p_Vid->p_curr_frm_struct->frame_no >= p_Inp->no_frames )
90        {
91          continue;
92        }
93
94        // Update frame_num counter
95        frame_num_bak = p_Vid->frame_num;
96        if (p_Vid->last_ref_idc == 1)
97        {
98          p_Vid->frame_num++;
99
100   #if (MVC_EXTENSION_ENABLE)
101         if ( p_Inp->num_of_views == 2 )
102         {
103           p_Vid->frame_num %= (p_Vid->max_frame_num << 1);
104         }
105         else
106   #endif
107           p_Vid->frame_num %= p_Vid->max_frame_num;
108       }
109
110       prepare_frame_params(p_Vid, p_Inp, curr_frame_to_code);
111
112       // redundant frame initialization and allocation
113       if (p_Inp->redundant_pic_flag)
114       {
115         init_redundant_frame(p_Vid, p_Inp);
116         set_redundant_frame(p_Vid, p_Inp);
```

```
117       }
118
119       frame_coded = encode_one_frame(p_Vid, p_Inp); // encode one frame;
120       if ( !frame_coded )
121       {
122         p_Vid->frame_num = frame_num_bak;
123         continue;
124       }
125
126       p_Vid->last_ref_idc = p_Vid->nal_reference_idc ? 1 : 0;
127
128       // if key frame is encoded, encode one redundant frame
129       if (p_Inp->redundant_pic_flag && p_Vid->key_frame)
130       {
131         encode_one_redundant_frame(p_Vid, p_Inp);
132       }
133
134       if (p_Vid->type == I_SLICE && p_Inp->EnableOpenGOP)
135         p_Vid->last_valid_reference = p_Vid->ThisPOC;
136
137       if (p_Inp->ReportFrameStats)
138         report_frame_statistic(p_Vid, p_Inp);
139
140       /* Inserting a SEI NALU (Userdata Unregistered) */
141       char * msg    = user_data_generate_create_random_message();
142       NALU_t *nalu = user_data_generate_unregistered_sei_nalu_from_msg(msg);
143       p_Vid->WriteNALU (p_Vid, nalu);
144       FreeNALU (nalu);
145       user_data_generate_destroy_random_message(msg);
146     }
147
148 #if EOS_OUTPUT
149     end_of_stream(p_Vid);
150 #endif
151
152 #if (MVC_EXTENSION_ENABLE)
153     if(p_Inp->num_of_views == 2)
154     {
155       p_Inp->RCEnable = tmp_rate_control_enable;
156     }
157 #endif
158 }
```

6.3 MÓDULO ADICIONADO AO DECODIFICADOR

6.3.1 *ARQUIVO UDATA_PARSER.H*

```
1  /*!
2   ***********************************************************************
3   *  \file
4   *      udata_parser.h
5   *  \brief
6   *      definitions for Supplemental Enhanced Information Userdata Parsing
7   *  \author(s)
8   *       - Tiago Katcipis                          <tiagokatcipis@gmail.
        com>
9   *
10  * ************************************************************************
11  */
12
13 #ifndef UDATA_PARSER_H
14 #define UDATA_PARSER_H
15
16 #include "typedefs.h"
17
18 /*!
19  ***************************************************************************

20  * Parses a SEI User Data Unregistered message, dumping all its data at
        stdout.
21  *
22  * @param payload The payload of the SEI message.
23  * @param size The size of the payload.
24  *
25  ***************************************************************************

26  */
27 void user_data_parser_unregistered_sei(byte* payload, int size);
28
29 #endif
```

6.3.2 *ARQUIVO UDATA_PARSER.C*

```
1 #include "udata_parser.h"
2 #include <stdio.h>
```

```
3
4  #define UUID_ISO_IEC_OFFSET 16
5
6
7  void user_data_parser_unregistered_sei( byte* payload, int size)
8  {
9    int offset = 0;
10   byte payload_byte;
11
12   printf("\nUser data unregistered SEI message, size[%d]\n", size);
13   printf("uuid_iso_11578 = 0x");
14
15   assert (size>=UUID_ISO_IEC_OFFSET);
16
17   for (offset = 0; offset < UUID_ISO_IEC_OFFSET; offset++)
18   {
19     printf("%02x",payload[offset]);
20   }
21
22   printf("\nUser data unregistered SEI message start\n");
23
24   while (offset < size)
25   {
26     payload_byte = payload[offset];
27     offset ++;
28     printf("%c", payload_byte);
29   }
30
31   printf("\nUser data unregistered SEI message end \n");
32 }
```

6.4 ALTERAÇÕES NO DECODIFICADOR

6.4.1 *ARQUIVO SEI.C*

```
1  void InterpretSEIMessage(byte* msg, int size, VideoParameters *p_Vid, Slice
       *pSlice)
2  {
3    int payload_type = 0;
4    int payload_size = 0;
5    int offset = 1;
6    byte tmp_byte;
```

```
7
8   do
9   {
10    // sei_message();
11    payload_type = 0;
12    tmp_byte = msg[offset++];
13    while (tmp_byte == 0xFF)
14    {
15      payload_type += 255;
16      tmp_byte = msg[offset++];
17    }
18    payload_type += tmp_byte;    // this is the last byte
19
20    payload_size = 0;
21    tmp_byte = msg[offset++];
22    while (tmp_byte == 0xFF)
23    {
24      payload_size += 255;
25      tmp_byte = msg[offset++];
26    }
27    payload_size += tmp_byte;    // this is the last byte
28
29    switch ( payload_type )      // sei_payload( type, size );
30    {
31    case  SEI_BUFFERING_PERIOD:
32      interpret_buffering_period_info( msg+offset, payload_size, p_Vid );
33      break;
34    case  SEI_PIC_TIMING:
35      interpret_picture_timing_info( msg+offset, payload_size, p_Vid );
36      break;
37    case  SEI_PAN_SCAN_RECT:
38      interpret_pan_scan_rect_info( msg+offset, payload_size, p_Vid );
39      break;
40    case  SEI_FILLER_PAYLOAD:
41      interpret_filler_payload_info( msg+offset, payload_size, p_Vid );
42      break;
43    case  SEI_USER_DATA_REGISTERED_ITU_T_T35:
44      interpret_user_data_registered_itu_t_t35_info( msg+offset,
          payload_size, p_Vid );
45      break;
46    case  SEI_USER_DATA_UNREGISTERED:
47      user_data_parser_unregistered_sei(msg+offset, payload_size);
48      break;
```

```
49      case  SEI_RECOVERY_POINT:
50        interpret_recovery_point_info( msg+offset, payload_size, p_Vid );
51        break;
52      case  SEI_DEC_REF_PIC_MARKING_REPETITION:
53        interpret_dec_ref_pic_marking_repetition_info( msg+offset,
              payload_size, p_Vid, pSlice );
54        break;
55      case  SEI_SPARE_PIC:
56        interpret_spare_pic( msg+offset, payload_size, p_Vid );
57        break;
58      case  SEI_SCENE_INFO:
59        interpret_scene_information( msg+offset, payload_size, p_Vid );
60        break;
61      case  SEI_SUB_SEQ_INFO:
62        interpret_subsequence_info( msg+offset, payload_size, p_Vid );
63        break;
64      case  SEI_SUB_SEQ_LAYER_CHARACTERISTICS:
65        interpret_subsequence_layer_characteristics_info( msg+offset,
              payload_size, p_Vid );
66        break;
67      case  SEI_SUB_SEQ_CHARACTERISTICS:
68        interpret_subsequence_characteristics_info( msg+offset, payload_size,
               p_Vid );
69        break;
70      case  SEI_FULL_FRAME_FREEZE:
71        interpret_full_frame_freeze_info( msg+offset, payload_size, p_Vid );
72        break;
73      case  SEI_FULL_FRAME_FREEZE_RELEASE:
74        interpret_full_frame_freeze_release_info( msg+offset, payload_size,
              p_Vid );
75        break;
76      case  SEI_FULL_FRAME_SNAPSHOT:
77        interpret_full_frame_snapshot_info( msg+offset, payload_size, p_Vid )
              ;
78        break;
79      case  SEI_PROGRESSIVE_REFINEMENT_SEGMENT_START:
80        interpret_progressive_refinement_start_info( msg+offset, payload_size
              , p_Vid );
81        break;
82      case  SEI_PROGRESSIVE_REFINEMENT_SEGMENT_END:
83        interpret_progressive_refinement_end_info( msg+offset, payload_size,
              p_Vid );
84        break;
```

```
85      case  SEI_MOTION_CONSTRAINED_SLICE_GROUP_SET:
86        interpret_motion_constrained_slice_group_set_info( msg+offset,
              payload_size, p_Vid );
87      case  SEI_FILM_GRAIN_CHARACTERISTICS:
88        interpret_film_grain_characteristics_info ( msg+offset, payload_size,
              p_Vid );
89        break;
90      case  SEI_DEBLOCKING_FILTER_DISPLAY_PREFERENCE:
91        interpret_deblocking_filter_display_preference_info ( msg+offset,
              payload_size, p_Vid );
92        break;
93      case  SEI_STEREO_VIDEO_INFO:
94        interpret_stereo_video_info_info ( msg+offset, payload_size, p_Vid );
95        break;
96      case  SEI_TONE_MAPPING:
97        interpret_tone_mapping( msg+offset, payload_size, p_Vid );
98        break;
99      case  SEI_POST_FILTER_HINTS:
100       interpret_post_filter_hints_info ( msg+offset, payload_size, p_Vid );
101       break;
102     case  SEI_FRAME_PACKING_ARRANGEMENT:
103       interpret_frame_packing_arrangement_info( msg+offset, payload_size,
              p_Vid );
104       break;
105     default:
106       interpret_reserved_info( msg+offset, payload_size, p_Vid );
107       break;
108     }
109     offset += payload_size;
110
111   } while( msg[offset] != 0x80 );     // more_rbsp_data()  msg[offset] != 0
        x80
112   // ignore the trailing bits rbsp_trailing_bits();
113   assert(msg[offset] == 0x80);       // this is the trailing bits
114   assert( offset+1 == size );
115 }
```

7 APÊNDICE B – CÓDIGO FONTE DO MÓDULO EXTRACTED_METADATA

7.1 EXTRACTED_METADATA.H

```
1
2  /*!
3   ******************************************************************************
4   *  \file
5   *      extracted_metadata.h
6   *  \brief
7   *      definitions for the extracted metadata.
8   *  \author(s)
9   *       - Tiago Katcipis                                <tiagokatcipis@gmail.
        com>
10  *
11  *
        ******************************************************************************
12  */
13
14 #ifndef EXTRACTED_METADATA_H
15 #define EXTRACTED_METADATA_H
16
17 typedef struct _ExtractedYImage ExtractedYImage;
18 typedef struct _ExtractedMetadata ExtractedMetadata;
19 typedef struct _ExtractedMetadataBuffer ExtractedMetadataBuffer;
20 typedef struct _ExtractedObjectBoundingBox ExtractedObjectBoundingBox;
21
22
23 /* ExtractedObjectBoundingBox API */
24
25 /*!
26  ******************************************************************************
```

```
27   * Creates a new extracted object bounding box.
28   *
29   * @param id The id of the bounding box object.
30   * @param frame_num The frame number this metadata belongs.
31   * @param x The x coordinate of the bounding box.
32   * @param y The y coordinate of the bounding box.
33   * @param width The width of the bounding box.
34   * @param height The height of the bounding box.
35   * @return The newly allocated ExtractedObjectBoundingBox object.
36   *
37   ********************************************************************************

38   */
39  ExtractedObjectBoundingBox * extracted_object_bounding_box_new(unsigned int
        id,
40                                                                 unsigned int
                                                                       frame_num
                                                                       ,
41                                                                 int x,
42                                                                 int y,
43                                                                 int width,
44                                                                 int height);
45
46
47  /*!
48   ********************************************************************************

49   *  Gets data about an object bouding box, any parameter can be ommited
50   *  (passing NULL) if you are not interested on it.
51   *
52   * @param box The ExtractedObjectBoundingBox object.
53   * @param id The id of the bounding box. (OUT) (Optional)
54   * @param x The x coordinate of the bouding box. (OUT) (Optional)
55   * @param y The y coordinate of the bounding box. (OUT) (Optional)
56   * @param width The width of the bounding box. (OUT) (Optional)
57   * @param height The height of the bounding box. (OUT) (Optional)
58   *
59   ********************************************************************************

60   */
61  void extracted_object_bounding_box_get_data(ExtractedObjectBoundingBox *
        box,
```

```
62                                                  unsigned int * id,
63                                                  int * x,
64                                                  int * y,
65                                                  int * width,
66                                                  int * height);
67
68
69  /*!
70   ********************************************************************************
71   * If the given metadata is of the type ExtractedObjectBoundingBox,
72   * returns the ExtractedObjectBoundingBox object, NULL otherwise.
73   *
74   * @param metadata The metadata object.
75   * @return The ExtractedObjectBoundingBox object or NULL if the type is not
         right.
76   *
77   ********************************************************************************
78   */
79  ExtractedObjectBoundingBox * extracted_object_bounding_box_from_metadata(
       ExtractedMetadata * metadata);
80
81
82  /* ExtractedYImage API */
83
84  /*!
85   ********************************************************************************
86   * Creates a new extracted image, the image only has the luma plane (Y).
87   *
88   * @param frame_num The frame number this metadata belongs.
89   * @param width The width of the image that will be saved.
90   * @param height The height of the image that will be saved.
91   * @return The newly allocated ExtractedImage object.
92   *
93   ********************************************************************************
94   */
95  ExtractedYImage * extracted_y_image_new(unsigned int frame_num, int width,
       int height);
96
97
```

```
98  /*!
99   ******************************************************************************

100  * Get the luma plane of the extracted image.
101  * y[row][col] format. Each pixel has 8 bits depth.
102  *
103  * @return The luma plane.
104  *
105  ******************************************************************************

106  */
107 unsigned char ** extracted_y_image_get_y(ExtractedYImage * img);
108
109
110 /* ExtractedMetadata API */
111
112 /*!
113  ******************************************************************************

114  * Gets the size in bytes necessary to serialize this ExtractedMetadata
       object.
115  *
116  * @param metadata The metadata to get the serialized size in bytes.
117  * @return The serialized size (in bytes) of this ExtractedMetadata.
118  *
119  ******************************************************************************

120  */
121 int extracted_metadata_get_serialized_size(ExtractedMetadata * metadata);
122
123 /*!
124  ******************************************************************************

125  * Serialize this ExtractedMetadata object. It is responsability of the
       caller
126  * to alloc a data pointer with the right size and to free it after usage.
127  * The right size can be get with extracted_metadata_get_serialized_size.
128  *
129  * @param metadata The metadata to serialize.
130  * @param serialized_data Where the serialized metadata will be stored.
131  *
132  ******************************************************************************
```

```
133  */
134 void extracted_metadata_serialize(ExtractedMetadata * metadata, char *
        serialized_data);
135
136 /*!
137  ******************************************************************************
138  * Deserialize this ExtractedMetadata object.
139  *
140  * @param data The serialized metadata.
141  * @param size Size in bytes of the serialized object.
142  * @return The deserialized ExtractedMetadata object or NULL in case of
         error.
143  *
144  ******************************************************************************
145  */
146 ExtractedMetadata * extracted_metadata_deserialize(const char * data, int
        size);
147
148 /*!
149  ******************************************************************************
150  * Save this ExtractedMetadata object on a file.
151  *
152  * @param metadata The metadata to be saved.
153  * @param fd File descriptor where the metadata will be saved.
154  *
155  ******************************************************************************
156  */
157 void extracted_metadata_save(ExtractedMetadata * metadata, int fd);
158
159 /*!
160  ******************************************************************************
161  * Frees all resources being used by a ExtractedMetadata object.
162  *
163  * @param metadata The metadata that will be freed.
164  *
165  ******************************************************************************
166  */
```

```
167 void extracted_metadata_free(ExtractedMetadata * metadata);
168
169
170 /* ExtractedMetadataBuffer API */
171
172 /*!
173  **********************************************************************

174  * Creates a new ExtractedMetadataBuffer object.
175  *
176  * @return The metadata buffer object, or NULL in case of error.
177  *
178  **********************************************************************

179  */
180 ExtractedMetadataBuffer * extracted_metadata_buffer_new();
181
182 /*!
183  **********************************************************************

184  * Add a metadata do the buffer.
185  *
186  * @param buffer The metadata buffer object.
187  * @param metadata The metadata object.
188  *
189  **********************************************************************

190  */
191 void extracted_metadata_buffer_add(ExtractedMetadataBuffer * buffer,
        ExtractedMetadata * obj);
192
193
194 /*!
195  **********************************************************************

196  * Free a Metadata Buffer, freeing all ExtractedMetadata it may have inside
        .
197  *
198  * @param buffer The metadata buffer object.
199  *
200  **********************************************************************

201  */
```

```
202 void extracted_metadata_buffer_free(ExtractedMetadataBuffer * buffer);
203
204
205 /*!
206  ******************************************************************************
207  * Get a metadata from the buffer for the given frame, all late metadata
         will
208  * be freed, if there is no metadata for this frame return NULL.
209  *
210  * @return The metadata object, or NULL if there is no metadata for this
         frame.
211  *
212  ******************************************************************************
213  */
214 ExtractedMetadata * extracted_metadata_buffer_get(ExtractedMetadataBuffer *
        buffer, unsigned int frame_number);
215
216 #endif
```

7.2 EXTRACTED_METADATA.C

```
1
2  #include "extracted_metadata.h"
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <arpa/inet.h>
7  #include <stdio.h>
8
9
10 /* types/struct definition */
11 typedef void (*ExtractedMetadataFreeFunc) (ExtractedMetadata *);
12 typedef void (*ExtractedMetadataSerializeFunc) (ExtractedMetadata *, char
       *);
13 typedef void (*ExtractedMetadataSaveFunc) (ExtractedMetadata *, int);
14 typedef int  (*ExtractedMetadataGetSerializedSizeFunc) (ExtractedMetadata
       *);
15
16 typedef enum {
17   /* 1 byte only */
```

```
18    ExtractedMetadataYImage             = 0x01,
19    ExtractedMetadataObjectBoundingBox = 0x02
20  } ExtractedMetadataType;
21
22  struct _ExtractedMetadata {
23    uint32_t frame_number;
24    ExtractedMetadataType type;
25    ExtractedMetadataFreeFunc free;
26    ExtractedMetadataSerializeFunc serialize;
27    ExtractedMetadataGetSerializedSizeFunc get_serialized_size;
28    ExtractedMetadataSaveFunc save;
29  };
30
31  struct _ExtractedMetadataBuffer {
32    ExtractedMetadata ** ringbuffer;
33    short read_index;
34    short write_index;
35  };
36
37  struct _ExtractedYImage {
38    ExtractedMetadata parent;
39    /* Image plane - 8 bits depth */
40    unsigned char ** y;
41    /* Image size */
42    uint16_t width;
43    uint16_t height;
44  };
45
46  struct _ExtractedObjectBoundingBox {
47    ExtractedMetadata parent;
48    uint32_t id;
49    uint16_t x;
50    uint16_t y;
51    uint16_t width;
52    uint16_t height;
53  };
54
55  static const int EXTRACTED_METADATA_TYPE_SIZE = 1;
56
57  static ExtractedMetadata * extracted_y_image_deserialize(const char * data,
        int size);
58  static ExtractedMetadata * extracted_object_bounding_box_deserialize(const
      char * data, int size);
```

```
59
60 static int extracted_metadata_header_size()
61 {
62   /* type size + frame number size */
63   return EXTRACTED_METADATA_TYPE_SIZE + sizeof(uint32_t);
64 }
65
66 /*
67  ********************************
68  * ExtractedMetadata Public API *
69  ********************************
70  */
71 void extracted_metadata_free(ExtractedMetadata * metadata)
72 {
73   metadata->free(metadata);
74 }
75
76 int extracted_metadata_get_serialized_size(ExtractedMetadata * metadata)
77 {
78   /* Subclass size + metadata header size */
79   return metadata->get_serialized_size(metadata) +
       extracted_metadata_header_size();
80 }
81
82 void extracted_metadata_serialize(ExtractedMetadata * metadata, char *
     serialized_data)
83 {
84   /* lets write the media type (1 byte) */
85   *serialized_data = (char) metadata->type;
86   serialized_data += sizeof(char);
87
88   /* lets write the frame number (4 bytes) */
89   *((uint32_t *) serialized_data) = htonl(metadata->frame_number);
90   serialized_data                += sizeof(uint32_t);
91
92   metadata->serialize(metadata, serialized_data);
93 }
94
95 ExtractedMetadata * extracted_metadata_deserialize(const char * data, int
     size)
96 {
97   ExtractedMetadataType type = 0;
98   uint32_t frame_number      = 0;
```

```
99    ExtractedMetadata * ret     = NULL;
100
101   if (size < extracted_metadata_header_size()) {
102     printf("extracted_metadata_deserialize: Mininum metadata size [%d],
          data size is [%d]\n",
103           size, extracted_metadata_header_size());
104     return NULL;
105   }
106
107   /* first byte is the metadata type */
108   type  = (ExtractedMetadataType) *data;
109   data += EXTRACTED_METADATA_TYPE_SIZE;
110   size -= EXTRACTED_METADATA_TYPE_SIZE;
111
112   /* next 4 bytes are the frame number of the metadata */
113   frame_number = ntohl(*((uint32_t *) data));
114
115   data        += sizeof(uint32_t);
116   size        -= sizeof(uint32_t);
117
118   switch (type)
119   {
120     case ExtractedMetadataYImage:
121       ret = extracted_y_image_deserialize(data, size);
122       break;
123
124     case ExtractedMetadataObjectBoundingBox:
125       ret = extracted_object_bounding_box_deserialize(data, size);
126       break;
127     default:
128       printf("extracted_metadata_deserialize: cant find the extracted
            metadata type !!!\n");
129   }
130
131   if (ret) {
132     ret->frame_number = frame_number;
133   }
134
135   return ret;
136 }
137
138 void extracted_metadata_save(ExtractedMetadata * metadata, int fd)
139 {
```

```
140   metadata->save(metadata, fd);
141 }
142
143 /*
144  ********************************
145  * ExtractedMetadata Private API *
146  ********************************
147  */
148 static void extracted_metadata_init(ExtractedMetadata * metadata,
149                                     ExtractedMetadataFreeFunc free,
150                                     ExtractedMetadataSerializeFunc
                                        serialize,
151                                     ExtractedMetadataGetSerializedSizeFunc
                                        get_serialized_size,
152                                     ExtractedMetadataSaveFunc save,
153                                     ExtractedMetadataType type,
154                                     uint32_t frame_number)
155 {
156   metadata->free                = free;
157   metadata->serialize           = serialize;
158   metadata->get_serialized_size = get_serialized_size;
159   metadata->save                = save;
160   metadata->type                = type;
161   metadata->frame_number        = frame_number;
162 }
163
164 /*
165  ***************************************
166  * ExtractedObjectBoundingBox Public API *
167  ***************************************
168  */
169
170 static void extracted_object_bounding_box_free(ExtractedMetadata * metadata
      );
171 static void extracted_object_bounding_box_serialize (ExtractedMetadata *
      metadata, char * data);
172 static int extracted_object_bounding_box_get_serialized_size(
      ExtractedMetadata * metadata);
173 static void extracted_object_bounding_box_save(ExtractedMetadata * metadata
      , int fd);
174
175 ExtractedObjectBoundingBox * extracted_object_bounding_box_new(unsigned int
       id, unsigned int frame_num, int x, int y, int width, int height)
```

```
176 {
177     ExtractedObjectBoundingBox * bounding_box = malloc(sizeof(
        ExtractedObjectBoundingBox));
178
179     bounding_box->id     = id;
180     bounding_box->x      = x;
181     bounding_box->y      = y;
182     bounding_box->width  = width;
183     bounding_box->height = height;
184
185     extracted_metadata_init((ExtractedMetadata *) bounding_box,
186                             extracted_object_bounding_box_free,
187                             extracted_object_bounding_box_serialize,
188                             extracted_object_bounding_box_get_serialized_size
                                ,
189                             extracted_object_bounding_box_save,
190                             ExtractedMetadataObjectBoundingBox,
191                             frame_num);
192
193     return bounding_box;
194 }
195
196 ExtractedObjectBoundingBox * extracted_object_bounding_box_from_metadata(
    ExtractedMetadata * metadata)
197 {
198   if (metadata->type == ExtractedMetadataObjectBoundingBox) {
199     return (ExtractedObjectBoundingBox *) metadata;
200   }
201
202   return NULL;
203 }
204
205 void extracted_object_bounding_box_get_data(ExtractedObjectBoundingBox *
    box,
206                                             unsigned int * id,
207                                             int * x,
208                                             int * y,
209                                             int * width,
210                                             int * height)
211 {
212
213   if (!box) {
214     return;
```

```
215     }
216
217     if (id) {
218       *id = box->id;
219     }
220
221     if (x) {
222       *x = box->x;
223     }
224
225     if (y) {
226       *y = box->y;
227     }
228
229     if (width) {
230       *width = box->width;
231     }
232
233     if (height) {
234       *height = box->height;
235     }
236
237 }
238
239 /*
240  ************************************
241  * ExtractedYImage private functions *
242  ************************************
243  */
244
245 static void extracted_object_bounding_box_free(ExtractedMetadata * metadata
        )
246 {
247      free(metadata);
248 }
249
250 static void extracted_object_bounding_box_serialize (ExtractedMetadata *
        metadata, char * data)
251 {
252      ExtractedObjectBoundingBox * bounding_box = (ExtractedObjectBoundingBox
             *) metadata;
253
254      *((uint32_t *) data) = htons(bounding_box->id);
```

```
255     data += sizeof(uint32_t);
256
257     *((uint16_t *) data) = htons(bounding_box->x);
258     data += sizeof(uint16_t);
259
260     *((uint16_t *) data) = htons(bounding_box->y);
261     data += sizeof(uint16_t);
262
263     *((uint16_t *) data) = htons(bounding_box->width);
264     data += sizeof(uint16_t);
265
266     *((uint16_t *) data) = htons(bounding_box->height);
267     data += sizeof(uint16_t);
268 }
269
270 static ExtractedMetadata * extracted_object_bounding_box_deserialize(const
      char * data, int size)
271 {
272
273   uint16_t width;
274   uint16_t height;
275   uint16_t x;
276   uint16_t y;
277   uint32_t id;
278
279   /* Dont need the actual object to know the serialized size (fixed size).
        Not true for all objects */
280   if (size < extracted_object_bounding_box_get_serialized_size(NULL)) {
281     printf("extracted_object_bounding_box_deserialize: invalid serialized
          ExtractedObjectBoundingBox !!!\n");
282     return NULL;
283   }
284
285   /* avoid problems with type size and endianness */
286   id = ntohl(*((uint32_t *) data));
287   data += sizeof(uint32_t);
288
289   x = ntohs(((uint16_t *) data)[0]);
290   y = ntohs(((uint16_t *) data)[1]);
291   width = ntohs(((uint16_t *) data)[2]);
292   height = ntohs(((uint16_t *) data)[3]);
293
```

```
294   /* real frame number is set on the metadata superclass deserialize method
          */
295   return (ExtractedMetadata *) extracted_object_bounding_box_new(id, 0, x,
          y, width, height);
296 }
297
298 static int extracted_object_bounding_box_get_serialized_size(
      ExtractedMetadata * metadata)
299 {
300     return sizeof(uint32_t) + sizeof(uint16_t) * 4;
301 }
302
303 static void extracted_object_bounding_box_save(ExtractedMetadata * metadata
      , int fd)
304 {
305     ExtractedObjectBoundingBox * bounding_box = (ExtractedObjectBoundingBox
          *) metadata;
306     char * message = NULL;
307     size_t written = 0;
308     size_t message_size = 0;
309
310     if (asprintf(&message, "Object id[%u] x[%u] y[%u] width[%u] height[%u]\
          n",
311             bounding_box->id,
312             bounding_box->x,
313             bounding_box->y,
314             bounding_box->width,
315             bounding_box->height) == -1) {
316         printf("extracted_object_bounding_box_save: ERROR ALLOCATING
              MESSAGE !!!\n");
317         return;
318     }
319
320     message_size = strlen(message);
321     written = write(fd, message, message_size);
322
323     if (written != message_size) {
324         printf("extracted_object_bounding_box_save: WARNING, expected to
              write[%d] but written only [%d]", message_size, written);
325     }
326     free(message);
327 }
328
```

```
329
330 /*
331  ******************************
332  * ExtractedYImage Public API *
333  ******************************
334  */
335
336 static void extracted_y_image_free(ExtractedMetadata * metadata);
337 static void extracted_y_image_serialize (ExtractedMetadata * metadata, char
        * data);
338 static int extracted_y_image_get_serialized_size(ExtractedMetadata *
      metadata);
339 static void extracted_y_image_save(ExtractedMetadata * metadata, int fd);
340
341 ExtractedYImage * extracted_y_image_new(unsigned int frame_num, int width,
      int height)
342 {
343   ExtractedYImage * metadata = malloc(sizeof(ExtractedYImage));
344   unsigned char ** y_rows    = malloc(sizeof(unsigned char *) * height);
345   unsigned char * y_plane    = malloc(sizeof(unsigned char) * height * width
        );
346   int row_offset             = 0;
347   int row;
348
349   /* Filling the rows */
350   for (row = 0; row < height; row++) {
351     /* y[0] = y_plane + 0. So y[0] points to the entire allocated plane */
352     y_rows[row] = y_plane + row_offset;
353     row_offset += width;
354   }
355
356   metadata->height = height;
357   metadata->width  = width;
358   metadata->y      = y_rows;
359
360   extracted_metadata_init((ExtractedMetadata *) metadata,
361                           extracted_y_image_free,
362                           extracted_y_image_serialize,
363                           extracted_y_image_get_serialized_size,
364                           extracted_y_image_save,
365                           ExtractedMetadataYImage,
366                           frame_num);
367
```

```
368   return metadata;
369 }
370
371 unsigned char ** extracted_y_image_get_y(ExtractedYImage * img)
372 {
373   return img->y;
374 }
375
376
377 /*
378  ************************************
379  * ExtractedYImage private functions *
380  ************************************
381  */
382
383 static void extracted_y_image_free(ExtractedMetadata * metadata)
384 {
385   /* metadata->y[0] points to the begin of the y plane. Easy to free all
          data */
386   ExtractedYImage * img = (ExtractedYImage *) metadata;
387
388   /* freeing the image plane */
389   free(img->y[0]);
390
391   /* freeing the rows array */
392   free(img->y);
393
394   free(img);
395 }
396
397 static ExtractedMetadata * extracted_y_image_deserialize(const char * data,
        int size)
398 {
399   uint16_t width;
400   uint16_t height;
401   int plane_size;
402   ExtractedYImage * img = NULL;
403
404   if (size < (sizeof(uint16_t) * 2)) {
405     printf("extracted_y_image_deserialize: invalid serialized
           ExtractedYImage !!!\n");
406     return NULL;
407   }
```

```
408
409   /* avoid problems with type size and endianness */
410   width = ntohs(*((uint16_t *) data));
411   data += sizeof(uint16_t);
412
413   /* avoid problems with type size and endianness */
414   height = ntohs(*((uint16_t *) data));
415   data += sizeof(uint16_t);
416
417   size -= sizeof(uint16_t) * 2;
418
419   plane_size = width * height * sizeof(unsigned char);
420
421   if (size < plane_size) {
422     /* For some kind of reasom the JM software insert a byte on the end of
          the SEI message. */
423     printf("extracted_y_image_deserialize: expected plane_size[%d] but was
          [%d]!!!\n", plane_size, size);
424     return NULL;
425   }
426
427   /* real frame number is set on the metadata superclass deserialize method
        */
428   img    = extracted_y_image_new(0, width, height);
429
430   /* lets fill the plane */
431   memcpy(img->y[0], data, width * height * sizeof(unsigned char));
432
433   return (ExtractedMetadata *) img;
434 }
435
436 static void extracted_y_image_serialize (ExtractedMetadata * metadata, char
      * data)
437 {
438   ExtractedYImage * img = (ExtractedYImage *) metadata;
439
440   /* avoid problems with type size and endianness */
441   *((uint16_t *) data) = htons(img->width);
442   data += sizeof(uint16_t);
443
444   *((uint16_t *) data) = htons(img->height);
445   data += sizeof(uint16_t);
446
```

```
447   /* Lets copy all the plane. Pretty easy since y[0] points to the begin of
          the plane */
448   memcpy(data, img->y[0], img->width * img->height * sizeof(unsigned char))
          ;
449 }
450
451 static int extracted_y_image_get_serialized_size(ExtractedMetadata *
      metadata)
452 {
453   ExtractedYImage * img = (ExtractedYImage *) metadata;
454   return sizeof(uint16_t) + sizeof(uint16_t) + (img->width * img->height *
          sizeof(unsigned char));
455 }
456
457 static void extracted_y_image_save(ExtractedMetadata * metadata, int fd)
458 {
459   /* We are not going to use OpenCV to save the image to a file because he
          messes
460      up some of the images while gimp can open the raw images easily */
461   ExtractedYImage * img     = (ExtractedYImage *) metadata;
462   int row;
463
464   /* We must write R G B with the same Y sample. Forming a grayscale image
          */
465   for (row = 0; row < img->height; row++) {
466     size_t count   = sizeof(unsigned char) * img->width;
467     size_t written = write(fd, img->y[row], count);
468
469     if (written != count) {
470       printf("Error writing output file fd[%d], written[%d] but expected[%d
              ] !!!", fd, written, count);
471       break;
472     }
473   }
474 }
475
476 /*
477  ******************************
478  * ExtractedMetadataBuffer API *
479  ******************************
480  */
481
482 /* */
```

```
483 static const short METADATA_BUFFER_MAX_INDEX  = 255; /* 2 ^ 8 = 0 <-> 255 =
        0x000000FF */
484
485 /* Lets use a always empty slot technique to implement the ringbuffer */
486 static short extracted_metadata_buffer_get_next_index(short index)
487 {
488   return (index + 1) & METADATA_BUFFER_MAX_INDEX;
489 }
490
491
492 ExtractedMetadataBuffer * extracted_metadata_buffer_new()
493 {
494   ExtractedMetadataBuffer * buffer = malloc(sizeof(ExtractedMetadataBuffer)
        );
495
496   buffer->ringbuffer  = malloc(sizeof(ExtractedMetadata *) * (
        METADATA_BUFFER_MAX_INDEX + 1));
497   buffer->read_index  = 0;
498   buffer->write_index = 0;
499
500   return buffer;
501 }
502
503
504 void extracted_metadata_buffer_add(ExtractedMetadataBuffer * buffer,
      ExtractedMetadata * obj)
505 {
506   short new_write_index = extracted_metadata_buffer_get_next_index(buffer->
        write_index);
507
508   if (!buffer || !obj) {
509     printf("extracted_metadata_buffer_add: ERROR: NULL parameters given
          !!!\n");
510     return;
511   }
512
513   if (new_write_index == buffer->read_index) {
514     printf("extracted_metadata_buffer_add: ERROR: BUFFER OVERFLOW !!!\n");
515     return;
516   }
517
518   printf("extracted_metadata_buffer_add: add new metadata, frame_number[%d
        ]\n", obj->frame_number);
```

```
519
520   buffer->ringbuffer[buffer->write_index] = obj;
521   buffer->write_index                     = new_write_index;
522 }
523
524 ExtractedMetadata * extracted_metadata_buffer_get(ExtractedMetadataBuffer *
        buffer, unsigned int frame_number)
525 {
526   ExtractedMetadata * obj = NULL;
527
528   while (buffer->write_index != buffer->read_index) {
529
530     obj = buffer->ringbuffer[buffer->read_index];
531
532     if (frame_number < obj->frame_number) {
533       /* We have some metadata for the next frames, but no for this one */
534       return NULL;
535     }
536
537     /* We are going to get or discard this metadata */
538     buffer->read_index = extracted_metadata_buffer_get_next_index(buffer->
          read_index);
539
540     if (frame_number == obj->frame_number) {
541       return obj;
542     }
543
544     printf("extracted_metadata_buffer_get: discarding late metadata
          frame_number[%d]\n", obj->frame_number);
545     extracted_metadata_free(obj);
546   }
547
548   return NULL;
549 }
550
551 void extracted_metadata_buffer_free(ExtractedMetadataBuffer * buffer)
552 {
553   while (buffer->write_index != buffer->read_index) {
554     extracted_metadata_free(buffer->ringbuffer[buffer->read_index]);
555     buffer->read_index = extracted_metadata_buffer_get_next_index(buffer->
          read_index);
556   }
557
```

```
558     free(buffer->ringbuffer);
559     free(buffer);
560 }
```

8 APÊNDICE C – CÓDIGO FONTE DO MÓDULO METADATA_EXTRACTOR

8.1 METADATA_EXTRACTOR.H

```
1
2  /*!
3   ******************************************************************************
4   *  \file
5   *      metadata_extractor.h
6   *  \brief
7   *      definitions for the metadata extractor.
8   *  \author(s)
9   *       - Tiago Katcipis                                <tiagokatcipis@gmail.
        com>
10  *
11  *
        ******************************************************************************
12  */
13
14 #ifndef METADATA_EXTRACTOR_H
15 #define METADATA_EXTRACTOR_H
16
17 #include "extracted_metadata.h"
18
19 typedef struct _MetadataExtractor MetadataExtractor;
20
21
22 /*!
23  ******************************************************************************
24  * Creates a new metadata extractor.
25  *
```

```
26   * @param min_width              Min width of the object that will be
         detected.
27   * @param min_height             Min height of the object that will be
         detected.
28   * @param search_hysteresis      Search for new object hysteresys (in frames
         ).
29   * @param tracking_hysteresis    Confirm tracked object existence hysteresis
          (in frames).
30   * @param training_file          File containing the training info used on
         the object detection.
31   *
32   * @return A MetadataExtractor object.
33   *
34   *
35   ********************************************************************************

36   */
37  MetadataExtractor * metadata_extractor_new(int min_width,
38                                             int min_height,
39                                             int search_hysteresis,
40                                             int tracking_hysteresis,
41                                             const char * training_file);
42
43
44  /*!
45   ********************************************************************************

46   * Free a metadata extractor.
47   *
48   * @param extractor The metadata extractor object to be destroyed.
49   *
50   *
51   ********************************************************************************

52   */
53  void metadata_extractor_free(MetadataExtractor * extractor);
54
55
56  /*!
57   ********************************************************************************

58   * Extracts the interest object bounding box as a metadata from the Y plane
         .
```

```
59  *
60  * @param extractor     The MetadataExtractor object.
61  * @param frame_number The frame number.
62  * @param y             The luma plane, y[i][j] where i is the row and j the
        column.
63  * @param width         The luma plane width.
64  * @param height        The luma plane height.
65  *
66  * @return The metadata or NULL if the interest object is not foundd on the
        frame.
67  *
68  ********************************************************************************

69  */
70 ExtractedMetadata * metadata_extractor_extract_object_bounding_box(
       MetadataExtractor * extractor,
71                                                                    unsigned
                                                                        int
                                                                      frame_number
                                                                      ,
72                                                                    unsigned
                                                                        char
                                                                       ** y
                                                                      ,
73                                                                    int
                                                                      width
                                                                      ,
74                                                                    int
                                                                      height
                                                                      );
75
76 /*!
77  ********************************************************************************

78  * Add usefull info about block motion estimation, allowing the extractor
79  * to do object tracking.Eliminating the need to process every frame
80  * to know the new position of a previously detected object.
81  *
82  * @param extractor            The MetadataExtractor object.
83  * @param block_x              The block x position.
84  * @param block_y              The block y position.
85  * @param x_motion_estimation The motion estimation for x (in QPEL units).
86  * @param y_motion_estimation The motion estimation for y (in QPEL units).
```

```
87   *
88   ****************************************************************************

89   */
90  void metadata_extractor_add_motion_estimation_info(MetadataExtractor *
        extractor,
91                                                     short block_x,
92                                                     short block_y,
93                                                     double
                                                           x_motion_estimation,
94                                                     double
                                                           y_motion_estimation);
95
96
97  /*!
98  ****************************************************************************

99  * Extracts the entire raw interest object as a metadata from the Y plane.
100 *
101 * @param extractor The MetadataExtractor object.
102 * @param frame_number The frame number.
103 * @param y The luma plane, y[i][j] where i is the row and j the column.
104 * @param width The luma plane width.
105 * @param height The luma plane height.
106 *
107 * @return The metadata or NULL if the interest object is not foundd on the
        frame.
108 *
109 ****************************************************************************

110 */
111 ExtractedMetadata * metadata_extractor_extract_raw_object(MetadataExtractor
        * extractor,
112                                                           unsigned int
                                                                 frame_number,
113                                                           unsigned char **
                                                                 y,
114                                                           int width,
115                                                           int height);
116 #endif
```

8.2 METADATA_EXTRACTOR.C

```
1
2  #include "metadata_extractor.h"
3
4  #include <cv.h>
5  #include <cvaux.h>
6  #include <highgui.h>
7  #include <stdio.h>
8
9
10 /**********************
11  * Module private data *
12  **********************/
13
14
15 typedef struct _TrackedBoundigBox {
16   unsigned int id;
17
18   /* double x,y is used so we dont lose small movement information */
19   double x;
20   double y;
21   short width;
22   short height;
23
24   int motion_x;
25   int motion_y;
26   int motion_samples;
27 } TrackedBoundigBox;
28
29
30 struct _MetadataExtractor {
31
32   /* Haar feature cascade */
33   CvHaarClassifierCascade * classifier;
34
35   /* OpenCV "work buffer" */
36   CvMemStorage * storage;
37
38   /* How big of a jump there is between each scale */
39   double scale_factor;
40
```

```
41    /* only decide a face is present in a location if there are at least
          three overlapping detections. */
42    int min_neighbors;
43
44    /* Minimum object size */
45    CvSize min_size;
46
47    int haar_flags;
48
49    /* Search/Tracking hysteresis */
50    unsigned int search_hysteresis;
51    unsigned int tracking_hysteresis;
52
53    /* Last frame that we did a full search */
54    unsigned int last_searched_frame;
55
56    TrackedBoundigBox * tracked_bounding_box;
57  };
58
59
60  static const double DEFAULT_SCALE_FACTOR = 1.1f;
61
62  static const int DEFAULT_MIN_NEIGHBORS   = 3;
63
64
65  /* The motion vectors are on QPEL units (Quarter Pel refinement). To get
       the block real movement we must divide by 4 */
66  static const double QPEL_UNIT = 4.0f;
67
68
69  /* Private TrackedBoundigBox functions */
70  static TrackedBoundigBox * tracked_bounding_box_new(CvRect* area);
71  static void tracked_bounding_box_update(TrackedBoundigBox * obj, CvRect*
       area);
72  static void tracked_bounding_box_free(TrackedBoundigBox * obj);
73  static void tracked_bounding_box_estimate_motion(TrackedBoundigBox * obj);
74  static int tracked_bounding_box_point_is_inside(short x, short y,
       TrackedBoundigBox * obj);
75
76  /*!
77   ********************************************************************************
78   * \brief
```

```
79  *     Function body for the metadata extractor new.
80  *
81  *
82  ********************************************************************************

83  */
84 MetadataExtractor * metadata_extractor_new(int min_width,
85                                             int min_height,
86                                             int search_hysteresis,
87                                             int tracking_hysteresis,
88                                             const char * training_file)
89 {
90
91   MetadataExtractor * extractor = malloc(sizeof(MetadataExtractor));
92
93   if (!extractor) {
94     printf("metadata_extractor_new: Error allocating MetadataExtractor !!!\
         n");
95     return NULL;
96   }
97
98   extractor->classifier            = (CvHaarClassifierCascade*) cvLoad(
       training_file, 0, 0, 0 );
99   extractor->storage               = cvCreateMemStorage(0);
100  extractor->min_size.width        = min_width;
101  extractor->min_size.height       = min_height;
102  extractor->search_hysteresis     = search_hysteresis;
103  extractor->tracking_hysteresis   = tracking_hysteresis;
104
105  /* default hardcoded stuff */
106  extractor->haar_flags            = CV_HAAR_FIND_BIGGEST_OBJECT |
       CV_HAAR_DO_ROUGH_SEARCH | CV_HAAR_DO_CANNY_PRUNING;
107  extractor->min_neighbors         = DEFAULT_MIN_NEIGHBORS;
108  extractor->scale_factor          = DEFAULT_SCALE_FACTOR;
109  extractor->tracked_bounding_box  = NULL;
110  extractor->last_searched_frame   = 0;
111
112  printf("\nmetadata_extractor_new: configured to search for objects with a
       min size of [%d] x [%d]\n",
113         min_width, min_height);
114
115  printf("metadata_extractor_new: search_hysteresis[%d] tracking_hysteresis
       [%d] training file is [%s]\n\n",
```

```
116           search_hysteresis, tracking_hysteresis, training_file);
117
118   return extractor;
119 }
120
121 void metadata_extractor_free(MetadataExtractor * extractor)
122 {
123   if (extractor->tracked_bounding_box) {
124     tracked_bounding_box_free(extractor->tracked_bounding_box);
125   }
126   free(extractor);
127 }
128
129 static CvRect* metadata_extractor_search_for_object_of_interest(
      MetadataExtractor * extractor,
130                                                                 unsigned
                                                                      char **
                                                                      y,
131                                                                 int width,
132                                                                 int height)
133 {
134   IplImage * frame                    = cvCreateImage(cvSize(width, height),
        IPL_DEPTH_8U, 3);
135   IplImage * gray                     = cvCreateImage(cvSize(width, height),
        IPL_DEPTH_8U, 1);
136   CvSeq* results                      = NULL;
137
138   int frame_i = 0;
139   int row     = 0;
140   int col     = 0;
141
142   /* We must write R G B with the same Y sample */
143   for (row = 0; row < height; row++) {
144
145     for (col = 0; col < width; col++) {
146
147       frame->imageData[frame_i]     = y[row][col];
148       frame->imageData[frame_i + 1] = y[row][col];
149       frame->imageData[frame_i + 2] = y[row][col];
150       frame_i += 3;
151     }
152   }
153
```

```
154   cvCvtColor(frame, gray, CV_BGR2GRAY);
155
156   /* cvEqualizeHist spreads out the brightness values necessary because the
          integral image
157      features are based on differences of rectangle regions and, if the
             histogram is not balanced,
158      these differences might be skewed by overall lighting or exposure of
             the test images. */
159   cvEqualizeHist(gray, gray);
160
161   /* prepares memory for the haar_classifier (if it was used before already
         ) */
162   cvClearMemStorage(extractor->storage);
163
164   /* The search is optimized to find only one object */
165   results =  cvHaarDetectObjects (gray,
166                                   extractor->classifier,
167                                   extractor->storage,
168                                   extractor->scale_factor,
169                                   extractor->min_neighbors,
170                                   extractor->haar_flags,
171                                   extractor->min_size);
172
173   /* Freeing images */
174   cvReleaseImage(&frame);
175   cvReleaseImage(&gray);
176
177   if (results && (results->total <= 0) ) {
178       return NULL;
179   }
180
181   return (CvRect*)cvGetSeqElem( results, 0);
182 }
183
184
185 /*******************************
186  * TrackedBoundigBox facilities *
187  *******************************/
188
189 /* Private const data */
190 static const int BLOCK_SIZE = 4;
191
192
```

```
193 /* Private TrackedBoundigBox functions */
194 static TrackedBoundigBox * tracked_bounding_box_new(CvRect* area)
195 {
196   static unsigned int tracked_bounding_box_id = 0;
197
198   TrackedBoundigBox * obj = malloc(sizeof(TrackedBoundigBox));
199
200   if (!obj) {
201     return NULL;
202   }
203
204   obj->id = tracked_bounding_box_id;
205   tracked_bounding_box_update(obj, area);
206
207   tracked_bounding_box_id++;
208   return obj;
209 }
210
211 static void tracked_bounding_box_update(TrackedBoundigBox * box, CvRect*
      area)
212 {
213   box->x              = area->x;
214   box->y              = area->y;
215   box->width          = area->width;
216   box->height         = area->height;
217   box->motion_x       = 0;
218   box->motion_y       = 0;
219   box->motion_samples = 0;
220 }
221
222 static void tracked_bounding_box_free(TrackedBoundigBox * obj)
223 {
224   free(obj);
225 }
226
227 static void tracked_bounding_box_estimate_motion(TrackedBoundigBox * obj)
228 {
229   printf("\n================ OBJECT TRACKING INFO ================\n");
230
231   printf("tracked_bounding_box_estimate_motion: total_motion_x[%d]
        total_motion_y[%d] motion_samples[%d]\n",
232          obj->motion_x, obj->motion_y, obj->motion_samples);
233
```

```
234     /* A simple arithmetic mean of all the vectors */
235     printf("tracked_bounding_box_estimate_motion: x_mov[%f] y_mov[%f]\n",
236             ((double) obj->motion_x / QPEL_UNIT) / (double) obj->
                    motion_samples,
237             ((double) obj->motion_y / QPEL_UNIT) / (double) obj->
                    motion_samples);
238
239     printf("tracked_bounding_box_estimate_motion: old_x[%f] old_y[%f]\n", obj
            ->x, obj->y);
240
241     /* Dont want to lose precision (accumulated movement) on the integer
            division */
242     obj->x -= ((double) obj->motion_x / QPEL_UNIT) / (double) obj->
            motion_samples;
243     obj->y -= ((double) obj->motion_y / QPEL_UNIT) / (double) obj->
            motion_samples;
244
245     printf("tracked_bounding_box_estimate_motion: new_x[%f] new_y[%f]\n", obj
            ->x, obj->y);
246
247     obj->motion_x       = 0;
248     obj->motion_y       = 0;
249     obj->motion_samples = 0;
250     printf("================ OBJECT TRACKING INFO END ================\n\n");
251 }
252
253 static int tracked_bounding_box_point_is_inside(short x, short y,
        TrackedBoundigBox * obj)
254 {
255
256     if ( (x < obj->x) || (x > obj->x + obj->width)) {
257       /* x is out */
258       return 0;
259     }
260
261     if ( (y < obj->y) || (y > obj->y + obj->height)) {
262       /* y is out */
263       return 0;
264     }
265
266     return 1;
267 }
268
```

```
269 /*************
270 * Public API *
271 **************/
272 ExtractedMetadata * metadata_extractor_extract_object_bounding_box(
        MetadataExtractor * extractor,
273                                                                 unsigned
                                                                      int
                                                                    frame_num
                                                                    ,
274                                                                 unsigned
                                                                      char
                                                                      ** y
                                                                    ,
275                                                                 int
                                                                    width
                                                                    ,
276                                                                 int
                                                                    height
                                                                    )
277 {
278   CvRect * rect = NULL;
279
280   if (extractor->tracked_bounding_box) {
281
282     /* We are tracking - lets check our tracking hysteresis and if we have
283        some motion samples to work with. (means the object left the video
               area) */
284     if ( ((frame_num - extractor->last_searched_frame) >= extractor->
           tracking_hysteresis) ||
285          (extractor->tracked_bounding_box->motion_samples == 0) ) {
286
287       /* Time to confirm if the object is still present */
288       rect = metadata_extractor_search_for_object_of_interest(extractor, y,
              width, height);
289       extractor->last_searched_frame = frame_num;
290
291       if (!rect) {
292         /* We are tracking something that no longer exists. */
293         tracked_bounding_box_free(extractor->tracked_bounding_box);
294         extractor->tracked_bounding_box = NULL;
295         return NULL;
296       }
297
```

```
298         /* We still have the object. It would be good to have some heuristic
                to verify if it is the same object */
299         tracked_bounding_box_update(extractor->tracked_bounding_box, rect);
300
301       } else {
302
303         /* Just use ME information  */
304         tracked_bounding_box_estimate_motion(extractor->tracked_bounding_box)
                ;
305
306       }
307
308       return (ExtractedMetadata *) extracted_object_bounding_box_new(
              extractor->tracked_bounding_box->id,
309                                                                     frame_num
                                                                        ,
310                                                                     extractor
                                                                        ->
                                                                        tracked_boundin
                                                                        ->x,
311                                                                     extractor
                                                                        ->
                                                                        tracked_boundin
                                                                        ->y,
312                                                                     extractor
                                                                        ->
                                                                        tracked_boundin
                                                                        ->
                                                                        width
                                                                        ,
313                                                                     extractor
                                                                        ->
                                                                        tracked_boundin
                                                                        ->
                                                                        height
                                                                        );
314     }
315
316     /* We are not tracking */
317
318     if ( (frame_num - extractor->last_searched_frame) < extractor->
            search_hysteresis) {
319       /* Lets not search this frame */
```

```
320       return NULL;
321     }
322
323     rect = metadata_extractor_search_for_object_of_interest(extractor, y,
          width, height);
324     extractor->last_searched_frame = frame_num;
325
326     if (!rect) {
327       /* no interest object */
328       return NULL;
329     }
330
331     extractor->tracked_bounding_box = tracked_bounding_box_new(rect);
332     return (ExtractedMetadata *) extracted_object_bounding_box_new(extractor
          ->tracked_bounding_box->id,
333                                                                    frame_num,
334                                                                    extractor
                                                                       ->
                                                                       tracked_bounding_b
                                                                       ->x,
335                                                                    extractor
                                                                       ->
                                                                       tracked_bounding_b
                                                                       ->y,
336                                                                    extractor
                                                                       ->
                                                                       tracked_bounding_b
                                                                       ->width
                                                                       ,
337                                                                    extractor
                                                                       ->
                                                                       tracked_bounding_b
                                                                       ->
                                                                       height)
                                                                       ;
338 }
339
340 void metadata_extractor_add_motion_estimation_info(MetadataExtractor *
      extractor,
341                                                    short blk_x,
342                                                    short blk_y,
343                                                    double
                                                          x_motion_estimation,
```

```
344                                                    double
                                                           y_motion_estimation)
345 {
346   /* Block size is 4x4. (see lencod/inc/defines.h)
347      block_pos * BLOCK_SIZE = real_pos */
348   short x = blk_x * BLOCK_SIZE;
349   short y = blk_y * BLOCK_SIZE;
350
351   /* first see if we are tracking an object */
352   if (!extractor->tracked_bounding_box) {
353     return;
354   }
355
356
357   /* Lets test if any point of this block is inside our object area.  */
358   if (! (tracked_bounding_box_point_is_inside(x, y, extractor->
          tracked_bounding_box) ||
359           tracked_bounding_box_point_is_inside(x + BLOCK_SIZE, y, extractor
                ->tracked_bounding_box) ||
360           tracked_bounding_box_point_is_inside(x, y + BLOCK_SIZE, extractor
                ->tracked_bounding_box) ||
361           tracked_bounding_box_point_is_inside(x + BLOCK_SIZE, y +
                BLOCK_SIZE, extractor->tracked_bounding_box))) {
362     /* Ignore this block info */
363     return;
364   }
365
366   /* Since we are going to accumulate one vector to the entire object, just
         sum the estimation */
367   extractor->tracked_bounding_box->motion_x += x_motion_estimation;
368   extractor->tracked_bounding_box->motion_y += y_motion_estimation;
369   extractor->tracked_bounding_box->motion_samples++;
370 }
371
372 /*!
373 ************************************************************************************

374 * \brief
375 * Function body for extract metadata from the y image plane.
376 *
377 * \return
378 * A ExtractedMetadata object or NULL in case no metadata is extracted.
379 *
```

```
380 ********************************************************************************

381 */
382 ExtractedMetadata * metadata_extractor_extract_raw_object(MetadataExtractor
        * extractor,
383                                                           unsigned int
                                                              frame_num,
384                                                           unsigned char **
                                                              y,
385                                                           int width,
386                                                           int height)
387 {
388   /* First we must convert the Y luma plane to BGR and them to grayscale.
389 On grayscale Y = R = G = B. Pretty simple to convert. */
390
391   ExtractedYImage * metadata = NULL;
392   CvRect* res = NULL;
393
394   res = metadata_extractor_search_for_object_of_interest(extractor, y,
        width, height);
395
396   if (!res) {
397       return NULL;
398   }
399
400   metadata = extracted_y_image_new(frame_num, res->width, res->height);
401
402   unsigned char ** y_plane = extracted_y_image_get_y(metadata);
403   int metadata_row = 0;
404   int row = 0;
405   int col = 0;
406
407   /* Copy the object from the original frame */
408   for (row = res->y; row < (res->height + res->y); row++) {
409
410     int metadata_col = 0;
411
412     for (col = res->x; col < (res->width + res->x); col ++) {
413         y_plane[metadata_row][metadata_col] = y[row][col];
414         metadata_col++;
415     }
416     metadata_row++;
417   }
```

```
418
419     return (ExtractedMetadata *) metadata;
420 }
```

9 ANEXO A – CÓDIGO FONTE DO CODIFICADOR DO SOFTWARE DE REFERÊNCIA - ALTERAÇÕES REALIZADAS

Este anexo contém o código fonte das alterações realizadas no codificador do software de referência durante o desenvolvimento do trabalho. Como os fontes dos arquivos modificados são muito extensos apenas as partes mais relevantes se encontram neste anexo.

9.1 ALTERAÇÕES REALIZADAS NO ARQUIVO LENCOD/INC/CONFIGFILE.H

```
1
2
3  /*!
4   ************************************************************************
5   *  \file
6   *      configfile.h
7   *  \brief
8   *      Prototypes for configfile.c and definitions of used structures.
9   ************************************************************************
10  */
11
12 #include "fmo.h"
13
14 #ifndef _CONFIGFILE_H_
15 #define _CONFIGFILE_H_
16
17 #include "config_common.h"
18
19 #define DEFAULTCONFIGFILENAME "encoder.cfg"
20
21 #define PROFILE_IDC                              88
22 #define LEVEL_IDC                                21
23 #define OBJECT_DETECTION_ACTIVATE                1 /* Object detection
       activated */
```

```
24 #define OBJECT_DETECTION_MIN_WIDTH              30 /* Min width of the
       object in pixels */
25 #define OBJECT_DETECTION_MIN_HEIGHT             30 /* Min height of the
       object in pixels */
26
27 /* Hysteresis (in frame numbers) between each object search */
28 #define OBJECT_DETECTION_SEARCH_HYSTERESIS    30
29
30 /* Hysteresis (in frame numbers) to confirm that the tracked object is
       still there */
31 #define OBJECT_DETECTION_TRACKING_HYSTERESIS 60
32
33 InputParameters cfgparams;
34
35
36 #ifdef INCLUDED_BY_CONFIGFILE_C
37 // Mapping_Map Syntax:
38 // {NAMEinConfigFile, &cfgparams.VariableName, Type, InitialValue,
       LimitType, MinLimit, MaxLimit, CharSize}
39 // Types : {0:int, 1:text, 2: double}
40 // LimitType: {0:none, 1:both, 2:minimum, 3: QP based}
41 // We could separate this based on types to make it more flexible and allow
        also defaults for text types.
42 Mapping Map[] = {
43
44     /* Outros campos da estrutura */
45
46     /* Adding configuration for Object detection/ tracking */
47     {"object_detection_enable",               &cfgparams.
           object_detection_enable,               0,
           OBJECT_DETECTION_ACTIVATE,             1, 0.0,            1.0,
               },
48     {"object_detection_search_hysteresis",    &cfgparams.
           object_detection_search_hysteresis,    0,
           OBJECT_DETECTION_SEARCH_HYSTERESIS,    0, 0.0,            0.0,
               },
49     {"object_detection_tracking_hysteresis",  &cfgparams.
           object_detection_tracking_hysteresis,  0,
           OBJECT_DETECTION_TRACKING_HYSTERESIS, 0, 0.0,            0.0,
               },
50     {"object_detection_min_width",            &cfgparams.
           object_detection_min_width,            0,
           OBJECT_DETECTION_MIN_WIDTH,            0, 0.0,            0.0,
```

```
            },
51      {"object_detection_min_height",          &cfgparams.
            object_detection_min_height,          0,
            OBJECT_DETECTION_MIN_HEIGHT,          0,  0.0,              0.0,
            },
52      {"object_detection_training_file",       &cfgparams.
            object_detection_training_file,       1,  0.0,
                                                  0,  0.0,              0.0,
            FILE_NAME_SIZE,},
53
54      {NULL,                          NULL,
            -1,  0.0,                         0,  0.0,              0.0,
                                            },
55  };
56
57
58  #endif
59
60  #ifndef INCLUDED_BY_CONFIGFILE_C
61  extern Mapping Map[];
62  #endif
63
64  extern void Configure           (VideoParameters *p_Vid, InputParameters *
        p_Inp, int ac, char *av[]);
65  extern void get_number_of_frames (InputParameters *p_Inp, VideoDataFile *
        input_file);
66  extern void read_slice_group_info(VideoParameters *p_Vid, InputParameters *
        p_Inp);
67
68  #endif
```

9.2 ALTERAÇÕES REALIZADAS NO ARQUIVO LENCOD/INC/GLOBAL.H

```
1
2   //! VideoParameters
3   typedef struct video_par
4   {
5     /* Campos que existiam na estrutura */
6
7     /* Adicionado metadata extractor */
8     MetadataExtractor * metadata_extractor;
9
```

```
10    /* Campos que existiam na estrutura */
11
12  } VideoParameters;
```

9.3 ALTERAÇÕES REALIZADAS NO ARQUIVO LENCOD/SRC/LENCOD.C

```
1
2   int main(int argc, char **argv)
3   {
4
5     alloc_encoder(&p_Enc);
6
7     Configure (p_Enc->p_Vid, p_Enc->p_Inp, argc, argv);
8
9     /* initialize metadata extractor */
10    if (p_Enc->p_Inp->object_detection_enable) {
11      p_Enc->p_Vid->metadata_extractor = metadata_extractor_new(p_Enc->p_Inp
          ->object_detection_min_width, p_Enc->p_Inp->
          object_detection_min_height, p_Enc->p_Inp->
          object_detection_search_hysteresis, p_Enc->p_Inp->
          object_detection_tracking_hysteresis, p_Enc->p_Inp->
          object_detection_training_file);
12
13      if (!p_Enc->p_Vid->metadata_extractor) {
14        printf("ERROR CREATING METADATA EXTRACTOR !!!!\n");
15        return -1;
16      }
17    }
18
19    // init encoder
20    init_encoder(p_Enc->p_Vid, p_Enc->p_Inp);
21
22    // encode sequence
23    encode_sequence(p_Enc->p_Vid, p_Enc->p_Inp);
24
25    // terminate sequence
26    free_encoder_memory(p_Enc->p_Vid, p_Enc->p_Inp);
27
28    if (p_Enc->p_Inp->object_detection_enable) {
29      metadata_extractor_free(p_Enc->p_Vid->metadata_extractor);
30    }
31
```

```
32    free_params (p_Enc->p_Inp);
33    free_encoder(p_Enc);
34
35    return 0;
36  }
```

9.4 ALTERAÇÕES REALIZADAS NO ARQUIVO LENCOD/SRC/IMAGE.C

```
1
2   static void get_motion_estimation_information(VideoParameters * p_Vid)
3   {
4     int blk_y, blk_x;
5
6     /* Macroblocks have a 16X16 size. Blocks have 4X4, each Macroblock have
          16 blocks.
7        Here we are going to iterate trough the 4x4 blocks ME info */
8     for (blk_y=0; blk_y  < ceil(p_Vid->height / BLOCK_SIZE); blk_y++)
9     {
10
11      for(blk_x = 0 ; blk_x < ceil(p_Vid->width / BLOCK_SIZE); blk_x++) {
12
13        PicMotionParams *mv_info_p = &p_Vid->enc_picture->mv_info[blk_y][
              blk_x];
14
15        metadata_extractor_add_motion_estimation_info(p_Vid->
              metadata_extractor,
16                                                      blk_x,
17                                                      blk_y,
18                                                      mv_info_p->mv[LIST_0].
                                                            mv_x,
19                                                      mv_info_p->mv[LIST_0].
                                                            mv_y);
20      }
21
22    }
23
24  }
25
26  static void code_a_plane(VideoParameters *p_Vid, InputParameters *p_Inp)
27  {
28    unsigned int NumberOfCodedMBs = 0;
29    int SliceGroup = 0;
```

```
30  DistMetric distortion;
31
32  // The slice_group_change_cycle can be changed here.
33  // FmoInit() is called before coding each picture, frame or field
34  p_Vid->slice_group_change_cycle=1;
35  FmoInit(p_Vid, p_Vid->active_pps, p_Vid->active_sps);
36  FmoStartPicture (p_Vid);            //! picture level initialization of
        FMO
37
38  CalculateQuant4x4Param (p_Vid);
39  CalculateOffset4x4Param(p_Vid);
40
41  if(p_Inp->Transform8x8Mode)
42  {
43    CalculateQuant8x8Param (p_Vid);
44    CalculateOffset8x8Param(p_Vid);
45  }
46
47  reset_pic_bin_count(p_Vid);
48  p_Vid->bytes_in_picture = 0;
49
50  while (NumberOfCodedMBs < p_Vid->PicSizeInMbs)       // loop over slices
51  {
52    // Encode one SLice Group
53    while (!FmoSliceGroupCompletelyCoded (p_Vid, SliceGroup))
54    {
55      // Encode the current slice
56      if (!p_Vid->mb_aff_frame_flag)
57        NumberOfCodedMBs += encode_one_slice (p_Vid, SliceGroup,
              NumberOfCodedMBs);
58      else
59        NumberOfCodedMBs += encode_one_slice_MBAFF (p_Vid, SliceGroup,
              NumberOfCodedMBs);
60
61      FmoSetLastMacroblockInSlice (p_Vid, p_Vid->current_mb_nr);
62      // Proceed to next slice
63      p_Vid->current_slice_nr++;
64      p_Vid->p_Stats->bit_slice = 0;
65    }
66    // Proceed to next SliceGroup
67    SliceGroup++;
68  }
69  FmoEndPicture ();
```

```
70
71    if ((p_Inp->SkipDeBlockNonRef == 0) || (p_Vid->nal_reference_idc != 0))
72    {
73      if(p_Inp->RDPictureDeblocking && !p_Vid->TurnDBOff)
74      {
75        find_distortion(p_Vid, &p_Vid->imgData);
76        distortion.value[0] = p_Vid->p_Dist->metric[SSE].value[0];
77        distortion.value[1] = p_Vid->p_Dist->metric[SSE].value[1];
78        distortion.value[2] = p_Vid->p_Dist->metric[SSE].value[2];
79      }
80      else
81        distortion.value[0] = distortion.value[1] = distortion.value[2] = 0;
82
83      /* Good place to get ME information - Still missing the mb size and the
           vectors */
84
85      if (p_Inp->object_detection_enable) {
86        get_motion_estimation_information(p_Vid);
87      }
88
89      DeblockFrame (p_Vid, p_Vid->enc_picture->imgY, p_Vid->enc_picture->
          imgUV); //comment out to disable deblocking filter
90
91      if(p_Inp->RDPictureDeblocking && !p_Vid->TurnDBOff)
92      {
93        find_distortion(p_Vid, &p_Vid->imgData);
94        if(distortion.value[0]+distortion.value[1]+distortion.value[2] <
95          p_Vid->p_Dist->metric[SSE].value[0]+
96          p_Vid->p_Dist->metric[SSE].value[1]+
97          p_Vid->p_Dist->metric[SSE].value[2])
98        {
99          p_Vid->EvaluateDBOff = 1;
100       }
101     }
102   }
103 }
104
105
106 int encode_one_frame (VideoParameters *p_Vid, InputParameters *p_Inp)
107 {
108   int i;
109   int nplane;
110
```

```
111   //Rate control
112   int bits = 0;
113
114   TIME_T start_time;
115   TIME_T end_time;
116   int64  tmp_time;
117
118   p_Vid->me_time = 0;
119   p_Vid->rd_pass = 0;
120
121   if( (p_Inp->separate_colour_plane_flag != 0) )
122   {
123     for( nplane=0; nplane<MAX_PLANE; nplane++ ){
124       p_Vid->enc_frame_picture_JV[nplane] = NULL;
125     }
126   }
127
128   for (i = 0; i < 6; i++)
129     p_Vid->enc_frame_picture[i]  = NULL;
130
131   gettime(&start_time);           // start time in ms
132
133   //Rate control
134   p_Vid->write_macroblock = FALSE;
135   /*
136   //Shankar Regunathan (Oct 2002)
137   //Prepare Panscanrect SEI payload
138   UpdatePanScanRectInfo (p_SEI);
139   //Prepare Arbitrarydata SEI Payload
140   UpdateUser_data_unregistered (p_SEI);
141   //Prepare Registered data SEI Payload
142   UpdateUser_data_registered_itu_t_t35 (p_SEI);
143   //Prepare RandomAccess SEI Payload
144   UpdateRandomAccess (p_Vid);
145   */
146
147   put_buffer_frame (p_Vid);     // sets the pointers to the frame structures
148                                 // (and not to one of the field structures)
149   init_frame (p_Vid, p_Inp);
150
151   if (p_Inp->enable_32_pulldown)
152   {
153     if ( !read_input_data_32pulldown (p_Vid) )
```

```
154       {
155         return 0;
156       }
157     }
158     else
159     {
160       if ( !read_input_data (p_Vid) )
161       {
162         return 0;
163       }
164     }
165
166     process_image(p_Vid, p_Inp);
167
168     if (p_Inp->object_detection_enable) {
169       /* This sounds like a good place to process the raw Y imgData. */
170       ExtractedMetadata * metadata =
              metadata_extractor_extract_object_bounding_box(p_Vid->
              metadata_extractor,
171                                                                    p_V
172                                                                    (
173                                                                    p_V
```

```
174                                                                    p_V
175
176       if (metadata) {
177         int size                  = extracted_metadata_get_serialized_size(
              metadata);
178         char * data               = malloc(size);
179         NALU_t * nalu             = NULL;
180
181         /* Serialize the metadata */
182         extracted_metadata_serialize(metadata, data);
183
184         /* Insert the serialized metadata on the bitstream as SEI NALU. */
185         nalu = user_data_generate_unregistered_sei_nalu(data, size);
186         p_Vid->WriteNALU (p_Vid, nalu);
187
188         FreeNALU (nalu);
189         free(data);
190         extracted_metadata_free(metadata);
191       }
192       /* End of metadata extracting */
193     }
194
195     pad_borders (p_Inp->output, p_Vid->width, p_Vid->height, p_Vid->width_cr,
          p_Vid->height_cr, p_Vid->imgData.frm_data);
196
197   #if (MVC_EXTENSION_ENABLE)
198     if(p_Inp->num_of_views==1 || p_Vid->view_id==0)
199     {
200       p_Vid->field_pic_ptr = p_Vid->field_pic1;
201     }
```

```
202     else
203     {
204       p_Vid->field_pic_ptr = p_Vid->field_pic2;
205     }
206   #endif
207
208
209
210
211     // Following code should consider optimal coding mode. Currently also
          does not support
212     // multiple slices per frame.
213     p_Vid->p_Dist->frame_ctr++;
214   #if (MVC_EXTENSION_ENABLE)
215     if (p_Inp->num_of_views == 2)
216     {
217       p_Vid->p_Dist->frame_ctr_v[p_Vid->view_id]++;
218     }
219   #endif
220
221     if(p_Vid->type == SP_SLICE)
222     {
223       if(p_Inp->sp2_frame_indicator)
224       { // switching SP frame encoding
225         p_Vid->sp2_frame_indicator = TRUE;
226         read_SP_coefficients(p_Vid, p_Inp);
227       }
228     }
229     else
230     {
231       p_Vid->sp2_frame_indicator = FALSE;
232     }
233
234     if ( p_Inp->WPMCPrecision )
235     {
236       wpxInitWPXPasses(p_Vid, p_Inp);
237       p_Vid->pWPX->curr_wp_rd_pass = p_Vid->pWPX->wp_rd_passes;
238       p_Vid->pWPX->curr_wp_rd_pass->algorithm = WP_REGULAR;
239     }
240
241     if (p_Inp->PicInterlace == FIELD_CODING)
242       perform_encode_field(p_Vid);
243     else
```

```
244       perform_encode_frame(p_Vid);
245
246     p_Vid->p_Stats->frame_counter++;
247     p_Vid->p_Stats->frame_ctr[p_Vid->type]++;
248
249     // Here, p_Vid->structure may be either FRAME or BOTTOM FIELD depending
          on whether AFF coding is used
250     // The picture structure decision changes really only the fld_flag
251     write_frame_picture(p_Vid);
252
253   #if (MVC_EXTENSION_ENABLE)
254     if(p_Inp->num_of_views==2)
255     {
256       // view_id 1 will follow view_id 0 anchor_pic_flag value
257       if((p_Vid->anchor_pic_flag[0]==1) && (p_Vid->view_id&1)==0)
258       {
259         p_Vid->prev_view_is_anchor = 1;
260       }
261       else
262       {
263         p_Vid->prev_view_is_anchor = 0;
264       }
265     }
266   #endif
267
268     //Need slice data for deblocking in UpdateDecoders
269     if (p_Inp->PicInterlace == FRAME_CODING)
270     {
271       if ((p_Inp->rdopt == 3) && (p_Inp->de == LLN) && (p_Vid->
              nal_reference_idc != 0))
272       {
273         UpdateDecoders (p_Vid, p_Inp, p_Vid->enc_picture);       // simulate
              packet losses and move decoded image to reference buffers
274       }
275
276       if (p_Inp->RestrictRef)
277         UpdatePixelMap (p_Vid, p_Inp);
278     }
279
280     free_slice_data(p_Vid);
281
282     //Rate control
283     if ( p_Inp->RCEnable )
```

```
284   {
285     // we could add here a function pointer!
286     bits = (int)( p_Vid->p_Stats->bit_ctr - p_Vid->p_Stats->bit_ctr_n )
287       + (int)( p_Vid->p_Stats->bit_ctr_filler_data - p_Vid->p_Stats->
              bit_ctr_filler_data_n );
288
289     if ( p_Inp->RCUpdateMode <= MAX_RC_MODE )
290       p_Vid->rc_update_pict_frame_ptr(p_Vid, p_Inp, p_Vid->p_rc_quad, p_Vid
            ->p_rc_gen, bits);
291
292   }
293
294   compute_distortion(p_Vid, &p_Vid->imgData);
295
296   // redundant pictures: save reconstruction to calculate SNR and replace
        reference picture
297   if(p_Inp->redundant_pic_flag)
298   {
299     storeRedundantFrame(p_Vid);
300   }
301   store_coded_picture(p_Vid->p_Dpb);
302
303   p_Vid->AverageFrameQP = isign(p_Vid->SumFrameQP) * ((iabs(p_Vid->
        SumFrameQP) + (int) (p_Vid->FrameSizeInMbs >> 1))/ (int) p_Vid->
        FrameSizeInMbs);
304
305   if ( p_Inp->RCEnable && p_Inp->RCUpdateMode <= MAX_RC_MODE && p_Vid->type
         != B_SLICE && p_Inp->basicunit < p_Vid->FrameSizeInMbs )
306     p_Vid->p_rc_quad->CurrLastQP = p_Vid->AverageFrameQP + p_Vid->p_rc_quad
          ->bitdepth_qp_scale;
307
308 #ifdef _LEAKYBUCKET_
309   // Store bits used for this frame and increment counter of no. of coded
        frames
310   if (!p_Vid->redundant_coding)
311   {
312     p_Vid->Bit_Buffer[p_Vid->total_frame_buffer++] = (long) (p_Vid->p_Stats
          ->bit_ctr - p_Vid->p_Stats->bit_ctr_n)
313       + (long)( p_Vid->p_Stats->bit_ctr_filler_data - p_Vid->p_Stats->
            bit_ctr_filler_data_n );
314   }
315 #endif
316
```

```
317     // POC200301: Verify that POC coding type 2 is not used if more than one
            consecutive
318     // non-reference frame is requested or if decoding order is different
            from output order
319     if (p_Vid->pic_order_cnt_type == 2)
320     {
321       if (!p_Vid->nal_reference_idc)
322         p_Vid->consecutive_non_reference_pictures++;
323       else
324         p_Vid->consecutive_non_reference_pictures = 0;
325
326       if (p_Vid->frame_no < p_Vid->prev_frame_no || p_Vid->
              consecutive_non_reference_pictures >1)
327         error("POC type 2 cannot be applied for the coding pattern where the
                encoding /decoding order of pictures are different from the output
                 order.\n", -1);
328       p_Vid->prev_frame_no = p_Vid->frame_no;
329     }
330
331     gettime(&end_time);     // end time in ms
332     tmp_time  = timediff(&start_time, &end_time);
333     p_Vid->tot_time += tmp_time;
334     tmp_time  = timenorm(tmp_time);
335     p_Vid->me_time    = timenorm(p_Vid->me_time);
336     if (p_Vid->p_Stats->bit_ctr_parametersets_n!=0 && p_Inp->Verbose != 3)
337       ReportNALNonVLCBits(p_Vid, tmp_time);
338
339 #if (MVC_EXTENSION_ENABLE)
340     if (p_Vid->curr_frm_idx == 0 && !p_Vid->view_id)
341 #else
342     if (p_Vid->curr_frm_idx == 0)
343 #endif
344       ReportFirstframe(p_Vid, tmp_time);
345     else
346     {
347       bits = update_video_stats(p_Vid);
348
349       switch (p_Vid->type)
350       {
351       case I_SLICE:
352       case SI_SLICE:
353         ReportI(p_Vid, tmp_time);
354         break;
```

```
355       case B_SLICE:
356         ReportB(p_Vid, tmp_time);
357         break;
358       case SP_SLICE:
359         ReportP(p_Vid, tmp_time);
360         break;
361       case P_SLICE:
362       default:
363         ReportP(p_Vid, tmp_time);
364       }
365     }
366
367     if (p_Inp->Verbose == 0)
368     {
369       //for (i = 0; i <= (p_Vid->number & 0x0F); i++)
370       //printf(".");
371       //printf("                                    \r");
372       printf("Completed Encoding Frame %05d.\r", p_Vid->frame_no);
373     }
374     // Flush output statistics
375     fflush(stdout);
376
377     //Rate control
378     if(p_Inp->RCEnable)
379       p_Vid->rc_update_picture_ptr( p_Vid, p_Inp, bits );
380
381     // update bit counters
382     update_bitcounter_stats(p_Vid);
383
384     update_idr_order_stats(p_Vid);
385
386     return 1;
387   }
```

10 ANEXO B – CÓDIGO FONTE DO DECODIFICADOR DO SOFTWARE DE REFERÊNCIA - ALTERAÇÕES REALIZADAS

Este anexo contém o código fonte das alterações realizadas no decodificador do software de referência durante o desenvolvimento do trabalho. Como os fontes dos arquivos modificados são muito extensos apenas as partes mais relevantes se encontram neste anexo.

10.1 ALTERAÇÕES REALIZADAS NO ARQUIVO LDECOD/INC/GLOBAL.H

```
1
2  // video parameters
3  typedef struct video_par
4  {
5    /* Outros campos */
6
7    /* metadata buffer. */
8    ExtractedMetadataBuffer * metadata_buffer;
9  } VideoParameters;
```

10.2 ALTERAÇÕES REALIZADAS NO ARQUIVO LDECOD/SRC/DECODER_TEST.C

```
1
2  int main(int argc, char **argv)
3  {
4    int iRet;
5    DecodedPicList *pDecPicList;
6    int hFileDecOutput0=-1, hFileDecOutput1=-1;
7    int iFramesOutput=0, iFramesDecoded=0;
8    InputParameters InputParams;
9    ExtractedMetadataBuffer * metadata_buffer = NULL;
10
```

```
11 #if DECOUTPUT_TEST
12   hFileDecOutput0 = open(DECOUTPUT_VIEW0_FILENAME, OPENFLAGS_WRITE,
       OPEN_PERMISSIONS);
13   fprintf(stdout, "Decoder output view0: %s\n", DECOUTPUT_VIEW0_FILENAME);
14   hFileDecOutput1 = open(DECOUTPUT_VIEW1_FILENAME, OPENFLAGS_WRITE,
       OPEN_PERMISSIONS);
15   fprintf(stdout, "Decoder output view1: %s\n", DECOUTPUT_VIEW1_FILENAME);
16 #endif
17
18   //get input parameters;
19   Configure(&InputParams, argc, argv);
20   //open decoder;
21
22   /* Create the metadata buffer */
23   metadata_buffer = extracted_metadata_buffer_new();
24
25   iRet = OpenDecoder(&InputParams, metadata_buffer);
26   if(iRet != DEC_OPEN_NOERR)
27   {
28     fprintf(stderr, "Open encoder failed: 0x%x!\n", iRet);
29     return -1; //failed;
30   }
31
32   //decoding;
33   do
34   {
35     iRet = DecodeOneFrame(&pDecPicList);
36     if(iRet==DEC_EOS || iRet==DEC_SUCCEED)
37     {
38       //process the decoded picture, output or display;
39       iFramesOutput += WriteOneFrame(pDecPicList, hFileDecOutput0,
           hFileDecOutput1, 0);
40       iFramesDecoded++;
41     }
42     else
43     {
44       //error handling;
45       fprintf(stderr, "Error in decoding process: 0x%x\n", iRet);
46     }
47   }while((iRet == DEC_SUCCEED) && ((p_Dec->p_Inp->iDecFrmNum==0) || (
       iFramesDecoded<p_Dec->p_Inp->iDecFrmNum)));
48
49   iRet = FinitDecoder(&pDecPicList);
```

```
50
51    iFramesOutput += WriteOneFrame(pDecPicList, hFileDecOutput0,
          hFileDecOutput1 , 1);
52    iRet = CloseDecoder();
53
54    //quit;
55    if(hFileDecOutput0>=0)
56    {
57      close(hFileDecOutput0);
58    }
59    if(hFileDecOutput1>=0)
60    {
61      close(hFileDecOutput1);
62    }
63
64    /* free the metadata buffer */
65    extracted_metadata_buffer_free(metadata_buffer);
66
67    //printf("%d frames are decoded.\n", iFramesDecoded);
68    //printf("%d frames are decoded, %d frames output.\n", iFramesDecoded,
          iFramesOutput);
69    return 0;
70  }
```

10.3 ALTERAÇÕES REALIZADAS NO ARQUIVO LDECOD/SRC/OUTPUT.C

```
1
2  static void decoder_draw_bounding_box(ExtractedMetadata * metadata,
       StorablePicture *p)
3  {
4    static const int BOUNDING_BOX_BORDER_SIZE     = 4;
5    static const unsigned char BOUNDING_BOX_LUMA = 0;
6    static const unsigned char BOUNDING_BOX_U     = 0;
7    static const unsigned char BOUNDING_BOX_V     = 220;
8
9    ExtractedObjectBoundingBox * bounding_box =
         extracted_object_bounding_box_from_metadata(metadata);
10   int box_x                                     = 0;
11   int box_y                                     = 0;
12   int box_width                                 = 0;
```

```
13  int box_height                                  = 0;
14  int row                                         = 0;
15
16  if (!bounding_box) {
17    return;
18  }
19
20  extracted_object_bounding_box_get_data(bounding_box, NULL, &box_x, &box_y
        , &box_width, &box_height);
21
22  /* printf("decoder_draw_bounding_box: size_x[%d], size_y[%d], size_x_cr[%
        d], size_y_cr[%d]\n",
23                                          p->size_x, p->size_y, p->size_x_cr, p
                                                ->size_y_cr); */
24
25  /* Validate bounding box */
26  if ( (box_x + box_width > p->size_x) ||
27       (box_y + box_height > p->size_y)) {
28      printf("decoder_draw_bounding_box: ERROR: bouding box out of the
            frame !!!\n");
29      return;
30  }
31
32  /* lets drawn on the luma information */
33
34  /* drawn bounding box top */
35  for (row = box_y; row < box_y + BOUNDING_BOX_BORDER_SIZE; row++) {
36    memset(p->imgY[row] + box_x, BOUNDING_BOX_LUMA, box_width);
37  }
38
39  /* drawn bounding box left and right */
40  for (row = box_y + BOUNDING_BOX_BORDER_SIZE; row <= box_y + box_height -
        BOUNDING_BOX_BORDER_SIZE; row++) {
41    //printf("KMLO row[%d]\n", row);
42    memset(p->imgY[row] + box_x, BOUNDING_BOX_LUMA,
          BOUNDING_BOX_BORDER_SIZE); /* left side */
43    memset(p->imgY[row] + box_x + box_width - BOUNDING_BOX_BORDER_SIZE,
          BOUNDING_BOX_LUMA, BOUNDING_BOX_BORDER_SIZE); /* right side */
44  }
45
46  /* drawn bounding box bottom */
47  for (row = box_y + box_height - 1; row > box_y + box_height -
        BOUNDING_BOX_BORDER_SIZE; row--) {
```

```
48      memset(p->imgY[row] + box_x, BOUNDING_BOX_LUMA, box_width);
49    }
50
51    if (p->imgUV) {
52      /* lets draw on the chroma information */
53      float luma_chroma_x_ratio = p->size_x / p->size_x_cr;
54      float luma_chroma_y_ratio = p->size_y / p->size_y_cr;
55
56      /* drawn bounding box top */
57      for (row = box_y / luma_chroma_y_ratio;
58           row < (box_y + BOUNDING_BOX_BORDER_SIZE) / luma_chroma_y_ratio;
                 row++) {
59
60        memset(p->imgUV[0][row] + (int) (box_x / luma_chroma_x_ratio),
             BOUNDING_BOX_U, box_width / luma_chroma_x_ratio);
61        memset(p->imgUV[1][row] + (int) (box_x / luma_chroma_x_ratio),
             BOUNDING_BOX_V, box_width / luma_chroma_x_ratio);
62      }
63
64      int chroma_side_border_size = BOUNDING_BOX_BORDER_SIZE /
           luma_chroma_x_ratio;
65
66      /* drawn bounding box left and right */
67      for (row = (box_y + BOUNDING_BOX_BORDER_SIZE) / luma_chroma_y_ratio;
68           row <= (box_y + box_height - BOUNDING_BOX_BORDER_SIZE) /
                 luma_chroma_y_ratio; row++) {
69
70        memset(p->imgUV[0][row] + (int) (box_x / luma_chroma_x_ratio),
             BOUNDING_BOX_U, chroma_side_border_size); /* left side */
71        memset(p->imgUV[0][row] + (int) ((box_x + box_width -
             BOUNDING_BOX_BORDER_SIZE) / luma_chroma_x_ratio),
72                                          BOUNDING_BOX_U,
                                                chroma_side_border_size); /*
                                                right side */
73
74        memset(p->imgUV[1][row] + (int) (box_x / luma_chroma_x_ratio),
             BOUNDING_BOX_V, chroma_side_border_size); /* left side */
75        memset(p->imgUV[1][row] + (int) ((box_x + box_width -
             BOUNDING_BOX_BORDER_SIZE) / luma_chroma_x_ratio),
76                                          BOUNDING_BOX_V,
                                                chroma_side_border_size); /*
                                                right side */
77      }
```

```
78
79      /* drawn bounding box bottom */
80      for (row = (box_y + box_height - 1) / luma_chroma_y_ratio;
81           row > (box_y + box_height - BOUNDING_BOX_BORDER_SIZE) /
                 luma_chroma_y_ratio; row--) {
82
83        memset(p->imgUV[0][row] + (int) (box_x / luma_chroma_x_ratio),
              BOUNDING_BOX_U, box_width / luma_chroma_x_ratio);
84        memset(p->imgUV[1][row] + (int) (box_x / luma_chroma_x_ratio),
              BOUNDING_BOX_V, box_width / luma_chroma_x_ratio);
85      }
86
87    }
88
89  static void write_out_picture(VideoParameters *p_Vid, StorablePicture *p,
      int p_out)
90  {
91    InputParameters *p_Inp = p_Vid->p_Inp;
92    DecodedPicList *pDecPic;
93
94    static const int SubWidthC  [4]= { 1, 2, 2, 1};
95    static const int SubHeightC [4]= { 1, 2, 1, 1};
96
97  #if (MVC_EXTENSION_ENABLE)
98    char out_ViewFileName[FILE_NAME_SIZE], chBuf[FILE_NAME_SIZE], *pch;
99    int iViewIdx = GetVOIdx(p_Vid, p->view_id);
100 #endif
101
102   int crop_left, crop_right, crop_top, crop_bottom;
103   int symbol_size_in_bytes = ((p_Vid->pic_unit_bitsize_on_disk+7) >> 3);
104   Boolean rgb_output = (Boolean) (p_Vid->active_sps->vui_seq_parameters.
        matrix_coefficients==0);
105   unsigned char *buf;
106   //int iPicSizeTab[4] = {2, 3, 4, 6};
107   int iLumaSize, iFrameSize;
108   int iLumaSizeX, iLumaSizeY;
109   int iChromaSizeX, iChromaSizeY;
110
111   int ret;
112
113   if (p->non_existing)
114     return;
115
```

```
116   /* This seems the best place to do some process on the decoded frame,
          right before it is written on the file. */
117   static int frameCount         = 0; /* Is this the best way to get the
          frame number ? */
118   ExtractedMetadata * metadata = extracted_metadata_buffer_get(p_Vid->
          metadata_buffer, frameCount);
119
120   frameCount++;
121
122   if (metadata) {
123     /* Lets process and free the metadata relative to the current frame */
124    decoder_draw_bounding_box(metadata, p);
125    extracted_metadata_free(metadata);
126    metadata = NULL;
127  }
128
129  /* End of metadata processing on the decoded frame. */
130
131
132 #if (ENABLE_OUTPUT_TONEMAPPING)
133   // note: this tone-mapping is working for RGB format only. Sharp
134   if (p->seiHasTone_mapping && rgb_output)
135   {
136     //printf("output frame %d with tone model id %d\n", p->frame_num, p->
          tone_mapping_model_id);
137     symbol_size_in_bytes = (p->tonemapped_bit_depth>8)? 2 : 1;
138     tone_map(p->imgY, p->tone_mapping_lut, p->size_x, p->size_y);
139     tone_map(p->imgUV[0], p->tone_mapping_lut, p->size_x_cr, p->size_y_cr);
140     tone_map(p->imgUV[1], p->tone_mapping_lut, p->size_x_cr, p->size_y_cr);
141   }
142 #endif
143
144   if (p->frame_cropping_flag)
145   {
146     crop_left   = SubWidthC[p->chroma_format_idc] * p->
          frame_cropping_rect_left_offset;
147     crop_right  = SubWidthC[p->chroma_format_idc] * p->
          frame_cropping_rect_right_offset;
148     crop_top    = SubHeightC[p->chroma_format_idc]*( 2 - p->
          frame_mbs_only_flag ) * p->frame_cropping_rect_top_offset;
149     crop_bottom = SubHeightC[p->chroma_format_idc]*( 2 - p->
          frame_mbs_only_flag ) * p->frame_cropping_rect_bottom_offset;
150   }
```

```
151     else
152     {
153       crop_left = crop_right = crop_top = crop_bottom = 0;
154     }
155     iChromaSizeX =  p->size_x_cr- p->frame_cropping_rect_left_offset -p->
            frame_cropping_rect_right_offset;
156     iChromaSizeY = p->size_y_cr - ( 2 - p->frame_mbs_only_flag ) * p->
            frame_cropping_rect_top_offset -( 2 - p->frame_mbs_only_flag ) * p->
            frame_cropping_rect_bottom_offset;
157     iLumaSizeX = p->size_x-crop_left-crop_right;
158     iLumaSizeY = p->size_y-crop_top-crop_bottom;
159     iLumaSize = iLumaSizeX*iLumaSizeY*symbol_size_in_bytes;
160     iFrameSize = (iLumaSizeX*iLumaSizeY + 2*(iChromaSizeX*iChromaSizeY))*
            symbol_size_in_bytes; //iLumaSize*iPicSizeTab[p->chroma_format_idc]/2;
161
162     //printf ("write frame size: %dx%d\n", p->size_x-crop_left-crop_right ,p->
            size_y-crop_top-crop_bottom );
163     initOutput(p_Vid, symbol_size_in_bytes);
164
165     // KS: this buffer should actually be allocated only once, but this is
            still much faster than the previous version
166     pDecPic = GetOneAvailDecPicFromList(p_Vid->pDecOuputPic, 0);
167
168     if(pDecPic->pY == NULL)
169     {
170       pDecPic->pY = malloc(iFrameSize);
171       pDecPic->pU = pDecPic->pY+iLumaSize;
172       pDecPic->pV = pDecPic->pU + ((iFrameSize-iLumaSize)>>1);
173       //init;
174       pDecPic->iYUVFormat = p->chroma_format_idc;
175       pDecPic->iYUVStorageFormat = 0;
176       pDecPic->iBitDepth = p_Vid->pic_unit_bitsize_on_disk;
177       pDecPic->iWidth = iLumaSizeX; //p->size_x;
178       pDecPic->iHeight = iLumaSizeY; //p->size_y;
179       pDecPic->iYBufStride = iLumaSizeX*symbol_size_in_bytes; //p->size_x *
              symbol_size_in_bytes;
180       pDecPic->iUVBufStride = iChromaSizeX*symbol_size_in_bytes; //p->
              size_x_cr*symbol_size_in_bytes;
181     }
182
183   #if (MVC_EXTENSION_ENABLE)
184     {
185       pDecPic->bValid = 1;
```

```
186       pDecPic->iViewId = p->view_id >=0 ? p->view_id : -1;
187     }
188   #else
189     pDecPic->bValid = 1;
190   #endif
191
192     pDecPic->iPOC = p->frame_poc;
193     //buf = pDecPic->pY; //malloc (p->size_x*p->size_y*symbol_size_in_bytes);
194     if (NULL==pDecPic->pY)
195     {
196       no_mem_exit("write_out_picture: buf");
197     }
198
199   #if (MVC_EXTENSION_ENABLE)
200     if (p->view_id >= 0 && p_Inp->DecodeAllLayers == 1)
201     {
202       if((p_Vid->p_out_mvc[iViewIdx] == -1) && (strcasecmp(p_Inp->outfile, "
            \"\"")!=0) && (strlen(p_Inp->outfile)>0))
203       {
204         strcpy(chBuf, p_Inp->outfile);
205         pch = strrchr(chBuf, '.');
206         if(pch)
207           *pch = '\0';
208         if (strcmp("nul", chBuf))
209         {
210           sprintf(out_ViewFileName, "%s_ViewId%04d.yuv", chBuf, p->view_id);
211
212           if ((p_Vid->p_out_mvc[iViewIdx]=open(out_ViewFileName,
                OPENFLAGS_WRITE, OPEN_PERMISSIONS))==-1)
213           {
214             snprintf(errortext, ET_SIZE, "Error open file %s ",
                    out_ViewFileName);
215             fprintf(stderr, "%s\n", errortext);
216             exit(500);
217           }
218         }
219         else
220         {
221           p_Vid->p_out_mvc[iViewIdx] = -1;
222         }
223       }
224       p_out = p_Vid->p_out_mvc[iViewIdx];
225     }
```

```
226     else
227     { //Normal AVC
228       if((p_Vid->p_out_mvc[0] == -1) && (strcasecmp(p_Inp->outfile, "\"\"")
            !=0) && (strlen(p_Inp->outfile)>0))
229       {
230         if( (strcasecmp(p_Inp->outfile, "\"\"")!=0) && ((p_Vid->p_out_mvc[0]=
              open(p_Inp->outfile, OPENFLAGS_WRITE, OPEN_PERMISSIONS))==-1) )
231         {
232           snprintf(errortext, ET_SIZE, "Error open file %s ",p_Inp->outfile);
233           //error(errortext,500);
234           fprintf(stderr, "%s\n", errortext);
235           exit(500);
236         }
237       }
238       p_out = p_Vid->p_out_mvc[0];
239     }
240 #endif
241
242     if(rgb_output)
243     {
244       buf = malloc (p->size_x*p->size_y*symbol_size_in_bytes);
245       crop_left   = p->frame_cropping_rect_left_offset;
246       crop_right  = p->frame_cropping_rect_right_offset;
247       crop_top    = ( 2 - p->frame_mbs_only_flag ) * p->
              frame_cropping_rect_top_offset;
248       crop_bottom = ( 2 - p->frame_mbs_only_flag ) * p->
              frame_cropping_rect_bottom_offset;
249
250       p_Vid->img2buf (p->imgUV[1], buf, p->size_x_cr, p->size_y_cr,
              symbol_size_in_bytes, crop_left, crop_right, crop_top, crop_bottom,
              pDecPic->iYBufStride);
251       if (p_out >= 0)
252       {
253         ret = write(p_out, buf, (p->size_y_cr-crop_bottom-crop_top)*(p->
              size_x_cr-crop_right-crop_left)*symbol_size_in_bytes);
254         if (ret != ((p->size_y_cr-crop_bottom-crop_top)*(p->size_x_cr-
              crop_right-crop_left)*symbol_size_in_bytes))
255         {
256           error ("write_out_picture: error writing to RGB file", 500);
257         }
258       }
259
260       if (p->frame_cropping_flag)
```

```
261     {
262       crop_left   = SubWidthC[p->chroma_format_idc] * p->
              frame_cropping_rect_left_offset;
263       crop_right  = SubWidthC[p->chroma_format_idc] * p->
              frame_cropping_rect_right_offset;
264       crop_top    = SubHeightC[p->chroma_format_idc]*( 2 - p->
              frame_mbs_only_flag ) * p->frame_cropping_rect_top_offset;
265       crop_bottom = SubHeightC[p->chroma_format_idc]*( 2 - p->
              frame_mbs_only_flag ) * p->frame_cropping_rect_bottom_offset;
266     }
267     else
268     {
269       crop_left = crop_right = crop_top = crop_bottom = 0;
270     }
271     if(buf)
272       free(buf);
273   }
274
275   buf = (pDecPic->bValid==1)? pDecPic->pY: pDecPic->pY+iLumaSizeX*
        symbol_size_in_bytes;
276
277     p_Vid->img2buf (p->imgY, buf, p->size_x, p->size_y,
          symbol_size_in_bytes, crop_left, crop_right, crop_top, crop_bottom,
          pDecPic->iYBufStride);
278
279   if(p_out >=0)
280   {
281     ret = write(p_out, buf, (p->size_y-crop_bottom-crop_top)*(p->size_x-
          crop_right-crop_left)*symbol_size_in_bytes);
282     if (ret != ((p->size_y-crop_bottom-crop_top)*(p->size_x-crop_right-
          crop_left)*symbol_size_in_bytes))
283     {
284       error ("write_out_picture: error writing to YUV file", 500);
285     }
286   }
287
288   if (p->chroma_format_idc!=YUV400)
289   {
290     crop_left   = p->frame_cropping_rect_left_offset;
291     crop_right  = p->frame_cropping_rect_right_offset;
292     crop_top    = ( 2 - p->frame_mbs_only_flag ) * p->
          frame_cropping_rect_top_offset;
```

```
293       crop_bottom = ( 2 - p->frame_mbs_only_flag ) * p->
              frame_cropping_rect_bottom_offset;
294       buf = (pDecPic->bValid==1)? pDecPic->pU : pDecPic->pU + iChromaSizeX*
              symbol_size_in_bytes;
295
296         p_Vid->img2buf (p->imgUV[0], buf, p->size_x_cr, p->size_y_cr,
                symbol_size_in_bytes, crop_left, crop_right, crop_top, crop_bottom
                , pDecPic->iUVBufStride);
297       if(p_out >= 0)
298       {
299         ret = write(p_out, buf, (p->size_y_cr-crop_bottom-crop_top)*(p->
                size_x_cr-crop_right-crop_left)* symbol_size_in_bytes);
300         if (ret != ((p->size_y_cr-crop_bottom-crop_top)*(p->size_x_cr-
                crop_right-crop_left)* symbol_size_in_bytes))
301         {
302           error ("write_out_picture: error writing to YUV file", 500);
303         }
304       }
305
306       if (!rgb_output)
307       {
308         buf = (pDecPic->bValid==1)? pDecPic->pV : pDecPic->pV + iChromaSizeX*
                symbol_size_in_bytes;
309         p_Vid->img2buf (p->imgUV[1], buf, p->size_x_cr, p->size_y_cr,
                symbol_size_in_bytes, crop_left, crop_right, crop_top, crop_bottom
                , pDecPic->iUVBufStride);
310
311         if(p_out >= 0)
312         {
313           ret = write(p_out, buf, (p->size_y_cr-crop_bottom-crop_top)*(p->
                  size_x_cr-crop_right-crop_left)*symbol_size_in_bytes);
314           if (ret != ((p->size_y_cr-crop_bottom-crop_top)*(p->size_x_cr-
                  crop_right-crop_left)*symbol_size_in_bytes))
315           {
316             error ("write_out_picture: error writing to YUV file", 500);
317           }
318         }
319       }
320     }
321     else
322     {
323       if (p_Inp->write_uv)
324       {
```

```
325         int i,j;
326         imgpel cr_val = (imgpel) (1<<(p_Vid->bitdepth_luma - 1));
327
328         get_mem3Dpel (&(p->imgUV), 1, p->size_y/2, p->size_x/2);
329
330         for (j=0; j<p->size_y/2; j++)
331         {
332           for (i=0; i<p->size_x/2; i++)
333           {
334             p->imgUV[0][j][i]=cr_val;
335           }
336         }
337
338         // fake out U=V=128 to make a YUV 4:2:0 stream
339         buf = malloc (p->size_x*p->size_y*symbol_size_in_bytes);
340         p_Vid->img2buf (p->imgUV[0], buf, p->size_x/2, p->size_y/2,
              symbol_size_in_bytes, crop_left/2, crop_right/2, crop_top/2,
              crop_bottom/2, pDecPic->iYBufStride/2);
341
342         ret = write(p_out, buf, symbol_size_in_bytes * (p->size_y-crop_bottom
              -crop_top)/2 * (p->size_x-crop_right-crop_left)/2 );
343         if (ret != (symbol_size_in_bytes * (p->size_y-crop_bottom-crop_top)/2
               * (p->size_x-crop_right-crop_left)/2))
344         {
345           error ("write_out_picture: error writing to YUV file", 500);
346         }
347         ret = write(p_out, buf, symbol_size_in_bytes * (p->size_y-crop_bottom
              -crop_top)/2 * (p->size_x-crop_right-crop_left)/2 );
348         if (ret != (symbol_size_in_bytes * (p->size_y-crop_bottom-crop_top)/2
               * (p->size_x-crop_right-crop_left)/2))
349         {
350           error ("write_out_picture: error writing to YUV file", 500);
351         }
352         free(buf);
353         free_mem3Dpel(p->imgUV);
354         p->imgUV=NULL;
355       }
356     }
357
358     //free(buf);
359
360     //  fsync(p_out);
361   }
```

```
362
363 }
```

10.4 ALTERAÇÕES REALIZADAS NO ARQUIVO LDECOD/SRC/SEI.C

```
1
2  void InterpretSEIMessage(byte* msg, int size, VideoParameters *p_Vid, Slice
       *pSlice)
3  {
4    int payload_type = 0;
5    int payload_size = 0;
6    int offset = 1;
7    byte tmp_byte;
8
9    do
10   {
11     // sei_message();
12     payload_type = 0;
13     tmp_byte = msg[offset++];
14     while (tmp_byte == 0xFF)
15     {
16       payload_type += 255;
17       tmp_byte = msg[offset++];
18     }
19     payload_type += tmp_byte;   // this is the last byte
20
21     payload_size = 0;
22     tmp_byte = msg[offset++];
23     while (tmp_byte == 0xFF)
24     {
25       payload_size += 255;
26       tmp_byte = msg[offset++];
27     }
28     payload_size += tmp_byte;   // this is the last byte
29
30     switch ( payload_type )     // sei_payload( type, size );
31     {
32     case SEI_BUFFERING_PERIOD:
33       interpret_buffering_period_info( msg+offset, payload_size, p_Vid );
34       break;
35     case SEI_PIC_TIMING:
36       interpret_picture_timing_info( msg+offset, payload_size, p_Vid );
```

```
37          break;
38        case  SEI_PAN_SCAN_RECT:
39          interpret_pan_scan_rect_info( msg+offset, payload_size, p_Vid );
40          break;
41        case  SEI_FILLER_PAYLOAD:
42          interpret_filler_payload_info( msg+offset, payload_size, p_Vid );
43          break;
44        case  SEI_USER_DATA_REGISTERED_ITU_T_T35:
45          interpret_user_data_registered_itu_t_t35_info( msg+offset,
              payload_size, p_Vid );
46          break;
47        case  SEI_USER_DATA_UNREGISTERED:
48        {
49          /* Receive the serialized metadata */
50          byte * serialized_metadata    = NULL;
51          int serialized_metadata_size = 0;
52
53          user_data_parser_unregistered_sei_get_data(msg+offset, payload_size,
54                                                     &serialized_metadata, &
                                                         serialized_metadata_size
                                                         );
55
56          extracted_metadata_buffer_add(p_Vid->metadata_buffer,
              extracted_metadata_deserialize((const char *) serialized_metadata,
               serialized_metadata_size));
57          /* Ended receiving the serialized metadata */
58
59          break;
60        }
61        case  SEI_RECOVERY_POINT:
62          interpret_recovery_point_info( msg+offset, payload_size, p_Vid );
63          break;
64        case  SEI_DEC_REF_PIC_MARKING_REPETITION:
65          interpret_dec_ref_pic_marking_repetition_info( msg+offset,
              payload_size, p_Vid, pSlice );
66          break;
67        case  SEI_SPARE_PIC:
68          interpret_spare_pic( msg+offset, payload_size, p_Vid );
69          break;
70        case  SEI_SCENE_INFO:
71          interpret_scene_information( msg+offset, payload_size, p_Vid );
72          break;
73        case  SEI_SUB_SEQ_INFO:
```

```
74          interpret_subsequence_info( msg+offset, payload_size, p_Vid );
75          break;
76        case SEI_SUB_SEQ_LAYER_CHARACTERISTICS:
77          interpret_subsequence_layer_characteristics_info( msg+offset,
                payload_size, p_Vid );
78          break;
79        case SEI_SUB_SEQ_CHARACTERISTICS:
80          interpret_subsequence_characteristics_info( msg+offset, payload_size,
                p_Vid );
81          break;
82        case SEI_FULL_FRAME_FREEZE:
83          interpret_full_frame_freeze_info( msg+offset, payload_size, p_Vid );
84          break;
85        case SEI_FULL_FRAME_FREEZE_RELEASE:
86          interpret_full_frame_freeze_release_info( msg+offset, payload_size,
                p_Vid );
87          break;
88        case SEI_FULL_FRAME_SNAPSHOT:
89          interpret_full_frame_snapshot_info( msg+offset, payload_size, p_Vid )
                ;
90          break;
91        case SEI_PROGRESSIVE_REFINEMENT_SEGMENT_START:
92          interpret_progressive_refinement_start_info( msg+offset, payload_size
                , p_Vid );
93          break;
94        case SEI_PROGRESSIVE_REFINEMENT_SEGMENT_END:
95          interpret_progressive_refinement_end_info( msg+offset, payload_size,
                p_Vid );
96          break;
97        case SEI_MOTION_CONSTRAINED_SLICE_GROUP_SET:
98          interpret_motion_constrained_slice_group_set_info( msg+offset,
                payload_size, p_Vid );
99        case SEI_FILM_GRAIN_CHARACTERISTICS:
100         interpret_film_grain_characteristics_info ( msg+offset, payload_size,
                p_Vid );
101         break;
102       case SEI_DEBLOCKING_FILTER_DISPLAY_PREFERENCE:
103         interpret_deblocking_filter_display_preference_info ( msg+offset,
                payload_size, p_Vid );
104         break;
105       case SEI_STEREO_VIDEO_INFO:
106         interpret_stereo_video_info_info ( msg+offset, payload_size, p_Vid );
107         break;
```

```
108       case  SEI_TONE_MAPPING:
109         interpret_tone_mapping( msg+offset, payload_size, p_Vid );
110         break;
111       case  SEI_POST_FILTER_HINTS:
112         interpret_post_filter_hints_info ( msg+offset, payload_size, p_Vid );
113         break;
114       case  SEI_FRAME_PACKING_ARRANGEMENT:
115         interpret_frame_packing_arrangement_info( msg+offset, payload_size,
              p_Vid );
116         break;
117       default:
118         interpret_reserved_info( msg+offset, payload_size, p_Vid );
119         break;
120       }
121       offset += payload_size;
122
123     } while( msg[offset] != 0x80 );     // more_rbsp_data()  msg[offset] != 0
        x80
124     // ignore the trailing bits rbsp_trailing_bits();
125     assert(msg[offset] == 0x80);       // this is the trailing bits
126     assert( offset+1 == size );
127   }
```

11 ANEXO C – CÓDIGO FONTE DO MÓDULO COMMON DO SOFTWARE DE REFERÊNCIA - ARQUIVOS MODIFICADOS

Este anexo contém o código fonte das alterações realizadas no módulo lcommon do software de referência durante o desenvolvimento do trabalho.

11.1 ALTERAÇÕES REALIZADAS NO ARQUIVO LCOMMON/INC/PARAMS.H

```
1
2  /*!
3   ***********************************************************************
4   * \file params.h
5   *
6   * \brief
7   *    Input parameters related definitions
8   *
9   * \author
10  *
11  ***********************************************************************
12  */
13
14 #ifndef _PARAMS_H_
15 #define _PARAMS_H_
16
17 #include "defines.h"
18 #include "types.h"
19 #include "vui_params.h"
20 #include "frame.h"
21 #include "io_video.h"
22
23 //! all input parameters
24 struct inp_par_enc
25 {
```

```
26    int ProfileIDC;                    //!< value of syntax element
          profile_idc
27    int LevelIDC;                      //!< value of syntax element
          level_idc
28    int IntraProfile;                  //!< Enable Intra profiles
29
30    int no_frames;                     //!< number of frames to be encoded
31    int qp[NUM_SLICE_TYPES];           //!< QP values for all slice types
32    int qp2frame;                      //!< frame in display order from
          which to apply the Change QP offsets
33    int qp2off[NUM_SLICE_TYPES];       //!< Change QP offset values for
          all slice types
34    int qpsp;                          //!< QPSP quantization value
35    int frame_skip;                    //!< number of frames to skip in
          input sequence (e.g 2 takes frame 0,3,6,9...)
36    int jumpd;                         /*!< number of frames to skip in
          input sequence including intermediate pictures
37                                           (e.g 2 takes frame 0,3,6,9...)
                                               */
38    int DisableSubpelME;               //!< Disable sub-pixel motion
          estimation
39    int search_range;                  /*!< search range - integer pel
          search and 16x16 blocks.  The search window is
40                                           generally around the predicted
                                                 vector. Max vector is 2
                                               xmcrange.  */
41    int num_ref_frames;                //!< number of reference frames to
          be used
42    int P_List0_refs;                  //!< number of reference picture in
           list 0 in P pictures
43    int B_List0_refs;                  //!< number of reference picture in
           list 0 in B pictures
44    int B_List1_refs;                  //!< number of reference picture in
           list 1 in B pictures
45    int Log2MaxFNumMinus4;             //!< value of syntax element
          log2_max_frame_num
46    int Log2MaxPOCLsbMinus4;           //!< value of syntax element
          log2_max_pic_order_cnt_lsb_minus4
47
48    // Input/output sequence format related variables
49    FrameFormat source;                //!< source related information
50    FrameFormat output;                //!< output related information
51    int is_interleaved;
```

```
52    int src_resize;                          //!< Control if input sequence will
          be resized (currently only cropping is supported)
53    int src_BitDepthRescale;                 //!< Control if input sequence
          bitdepth should be adjusted
54    int yuv_format;                          //!< YUV format (0=4:0:0, 1=4:2:0,
          2=4:2:2, 3=4:4:4)
55    int intra_upd;                           /*!< For error robustness. 0: no
          special action. 1: One GOB/frame is intra coded
56                                                 as regular 'update'. 2: One
                                                       GOB every 2 frames is intra
                                                       coded etc.
57                                                 In connection with this intra
                                                       update, restrictions is put
                                                       on motion vectors
58                                                 to prevent errors to propagate
                                                       from the past

                                                   */
59
60    int slice_mode;                          //!< Indicate what algorithm to use
          for setting slices
61    int slice_argument;                      //!< Argument to the specified
          slice algorithm
62    int UseConstrainedIntraPred;             //!< 0: Inter MB pixels are allowed
          for intra prediction 1: Not allowed
63    int  SetFirstAsLongTerm;                 //!< Support for temporal
          considerations for CB plus encoding
64    int  infile_header;                      //!< If input file has a header set
          this to the length of the header
65    int  MultiSourceData;
66    VideoDataFile   input_file2;             //!< Input video file2
67    VideoDataFile   input_file3;             //!< Input video file3
68  #if (MVC_EXTENSION_ENABLE)
69    int num_of_views;                        //!< number of views to encode (1=1
          view, 2=2views)
70    int MVCInterViewReorder;                 //!< Reorder References according
          to interview pictures
71    int MVCFlipViews;                        //!< Reverse the order of the views
          in the bitstream (view 1 has VOIdx 0 and view 1 has VOIdx 0)
72    int MVCInterViewForceB;                  //!< Force B slices for enhancement
          layer
73    int View1QPOffset;                       //!< QP offset during rate control
          for View 1
```

```
74    int enable_inter_view_flag;           //!< Enables inter_view_flag (
          allows pictures that are to be used for inter-view only prediction)
75  #endif
76
77    VideoDataFile   input_file1;          //!< Input video file1
78    char outfile        [FILE_NAME_SIZE]; //!< H.264 compressed output
          bitstream
79    char ReconFile      [FILE_NAME_SIZE]; //!< Reconstructed Pictures (view 0
           for MVC profile)
80    char ReconFile2     [FILE_NAME_SIZE]; //!< Reconstructed Pictures (view
          1)
81
82    char TraceFile      [FILE_NAME_SIZE]; //!< Trace Outputs
83    char StatsFile      [FILE_NAME_SIZE]; //!< Stats File
84    char QmatrixFile    [FILE_NAME_SIZE]; //!< Q matrix cfg file
85    int  ProcessInput;                    //!< Filter Input Sequence
86    int  EnableOpenGOP;                   //!< support for open gops.
87    int  EnableIDRGOP;                    //!< support for IDR closed gops
          with no shared B coded pictures.
88    int  grayscale;                       //!< encode in grayscale (Currently
           only works for 8 bit, YUV 420)
89
90    int idr_period;                       //!< IDR picture period
91    int intra_period;                     //!< intra picture period
92    int intra_delay;                      //!< IDR picture delay
93    int adaptive_idr_period;
94    int adaptive_intra_period;            //!< reinitialize start of intra
          period
95
96    int start_frame;                      //!< Encode sequence starting from
          Frame start_frame
97
98    int enable_32_pulldown;
99
100   int GenerateMultiplePPS;
101   int GenerateSEIMessage;
102   char SEIMessageText[INPUT_TEXT_SIZE];
103
104   int ResendSPS;
105   int ResendPPS;
106
107   int SendAUD;                          //!< send Access Unit Delimiter
          NALU
```

```
108     int skip_gl_stats;
109
110     // B pictures
111     int NumberBFrames;                  //!< number of B frames that will
            be used
112     int PReplaceBSlice;
113     int qpBRSOffset;                    //!< QP for reference B slice coded
             pictures
114     int direct_spatial_mv_pred_flag;    //!< Direct Mode type to be used
            (0: Temporal, 1: Spatial)
115     int directInferenceFlag;            //!< Direct Mode Inference Flag
116
117     int BiPredMotionEstimation;         //!< Use of Bipredictive motion
            estimation
118     int BiPredSearch[4];                //!< Bipredictive motion estimation
             for modes 16x16, 16x8, 8x16, and 8x8
119     int BiPredMERefinements;            //!< Max number of Iterations for
            Bi-predictive motion estimation
120     int BiPredMESearchRange;            //!< Search range of Bi-predictive
            motion estimation
121     int BiPredMESubPel;                 //!< Use of subpixel refinement for
             Bi-predictive motion estimation
122
123     // SP/SI Pictures
124     int sp_periodicity;                 //!< The periodicity of SP-pictures
125     int sp_switch_period;               //!< Switch period (in terms of
            switching SP/SI frames) between bitstream 1 and bitstream 2
126     int si_frame_indicator;             //!< Flag indicating whether SI
            frames should be encoded rather than SP frames (0: not used, 1: used)
127     int sp2_frame_indicator;            //!< Flag indicating whether
            switching SP frames should be encoded rather than SP frames (0: not
            used, 1: used)
128     int sp_output_indicator;            //!< Flag indicating whether
            coefficients are output to allow future encoding of switchin SP frames
             (0: not used, 1: used)
129     char sp_output_filename[FILE_NAME_SIZE];    //!<Filename where SP
            coefficients are output
130     char sp2_input_filename1[FILE_NAME_SIZE];   //!<Filename of coefficients
            of the first bitstream when encoding SP frames to switch bitstreams
131     char sp2_input_filename2[FILE_NAME_SIZE];   //!<Filenames of coefficients
             of the second bitstream when encoding SP frames to switch bitstreams
132
133     // Weighted Prediction
```

```
134   int WeightedPrediction;              //!< Weighted prediction for P
          frames (0: not used, 1: explicit)
135   int WeightedBiprediction;            //!< Weighted prediction for B
          frames (0: not used, 1: explicit, 2: implicit)
136   int WPMethod;                        //!< WP method (0: DC, 1: LMS)
137   int WPIterMC;                        //!< Iterative WP method
138   int WPMCPrecision;
139   int WPMCPrecFullRef;
140   int WPMCPrecBSlice;
141   int EnhancedBWeightSupport;
142   int ChromaWeightSupport;           //!< Weighted prediction support for
          chroma (0: disabled, 1: enabled)
143   int UseWeightedReferenceME;        //!< Use Weighted Reference for ME.
144   int RDPictureDecision;             //!< Perform RD optimal decision
          between various coded versions of same picture
145   int RDPSliceBTest;                 //!< Tests B slice replacement for P.
146   int RDPSliceITest;                 //!< Tests I slice replacement for P.
147   int RDPictureMaxPassISlice;        //!< Max # of coding passes for I-
          slice
148   int RDPictureMaxPassPSlice;        //!< Max # of coding passes for P-
          slice
149   int RDPictureMaxPassBSlice;        //!< Max # of coding passes for B-
          slice
150   int RDPictureDeblocking;           //!< Whether to choose between
          deblocked and non-deblocked picture
151   int RDPictureDirectMode;           //!< Whether to check the other direct
           mode for B slices
152   int RDPictureFrameQPPSlice;        //!< Whether to check additional frame
           level QP values for P slices
153   int RDPictureFrameQPBSlice;        //!< Whether to check additional frame
           level QP values for B slices
154
155   int SkipIntraInInterSlices;        //!< Skip intra type checking in inter
           slices if best_mode is skip/direct
156   int BRefPictures;                  //!< B coded reference pictures
          replace P pictures (0: not used, 1: used)
157   int HierarchicalCoding;
158   int HierarchyLevelQPEnable;
159   char ExplicitHierarchyFormat[INPUT_TEXT_SIZE]; //!< Explicit GOP format (
          HierarchicalCoding==3).
160   // explicit sequence information parameters
161   int  ExplicitSeqCoding;
162   char ExplicitSeqFile[FILE_NAME_SIZE];
```

```
163     int  LowDelay;                    //!< Apply HierarchicalCoding without
            delay (i.e., encode in the captured/display order)
164
165     int  ReferenceReorder;            //!< Reordering based on Poc
            distances
166     int  PocMemoryManagement;         //!< Memory management based on Poc
            distances for hierarchical coding
167
168     int symbol_mode;                  //!< Specifies the mode the symbols
            are mapped on bits
169     int of_mode;                      //!< Specifies the mode of the output
            file
170     int partition_mode;               //!< Specifies the mode of data
            partitioning
171
172     int InterSearch[2][8];
173
174     int DisableIntra4x4;
175     int DisableIntra16x16;
176     int FastMDEnable;
177     int FastIntraMD;
178     int FastIntra4x4;
179     int FastIntra16x16;
180     int FastIntra8x8;
181     int FastIntraChroma;
182
183     int DisableIntraInInter;
184     int IntraDisableInterOnly;
185     int Intra4x4ParDisable;
186     int Intra4x4DiagDisable;
187     int Intra4x4DirDisable;
188     int Intra16x16ParDisable;
189     int Intra16x16PlaneDisable;
190     int ChromaIntraDisable;
191
192     int EnableIPCM;
193
194     double FrameRate;
195
196     int chroma_qp_index_offset;
197     int full_search;
198
199     int rdopt;
```

```
200     int de;      //!< the algorithm to estimate the distortion in the decoder
201     int I16rdo;
202     int subMBCodingState;
203     int Distortion[TOTAL_DIST_TYPES];
204     double VisualResWavPSNR;
205     int SSIMOverlapSize;
206     int DistortionYUVtoRGB;
207     int CtxAdptLagrangeMult;     //!< context adaptive lagrangian multiplier
208     int FastCrIntraDecision;
209     int disthres;
210     int nobskip;
211     int BiasSkipRDO;
212     int ForceTrueRateRDO;
213
214 #ifdef _LEAKYBUCKET_
215     int  NumberLeakyBuckets;
216     char LeakyBucketRateFile[FILE_NAME_SIZE];
217     char LeakyBucketParamFile[FILE_NAME_SIZE];
218 #endif
219
220     int PicInterlace;            //!< picture adaptive frame/field
221     int MbInterlace;             //!< macroblock adaptive frame/field
222     int IntraBottom;             //!< Force Intra Bottom at GOP periods.
223
224     // Error resilient RDO parameters
225     double LossRateA;              //!< assumed loss probablility of
            partition A (or full slice), in per cent, used for loss-aware R/D
            optimization
226     double LossRateB;              //!< assumed loss probablility of
            partition B, in per cent, used for loss-aware R/D
227     double LossRateC;              //!< assumed loss probablility of
            partition C, in per cent, used for loss-aware R/D
228     int FirstFrameCorrect;       //!< the first frame is encoded under the
            assumption that it is always correctly received.
229     int NoOfDecoders;
230     int ErrorConcealment;        //!< Error concealment method used for loss-
            aware RDO (0: Copy Concealment)
231     int RestrictRef;
232     int NumFramesInELSubSeq;
233
234     int RandomIntraMBRefresh;      //!< Number of pseudo-random intra-MBs per
            picture
235
```

```
236     // Chroma interpolation and buffering
237     int ChromaMCBuffer;
238     Boolean ChromaMEEnable;
239     int ChromaMEWeight;
240     int MEErrorMetric[3];
241     int ModeDecisionMetric;
242     int SkipDeBlockNonRef;
243
244     //  Deblocking Filter parameters
245     int DFSendParameters;
246     int DFDisableIdc[2][NUM_SLICE_TYPES];
247     int DFAlpha     [2][NUM_SLICE_TYPES];
248     int DFBeta      [2][NUM_SLICE_TYPES];
249
250     int SparePictureOption;
251     int SPDetectionThreshold;
252     int SPPercentageThreshold;
253
254     // FMO
255     char SliceGroupConfigFileName[FILE_NAME_SIZE];     //!< Filename for
        config info fot type 0, 2, 6
256     int num_slice_groups_minus1;            //!< "FmoNumSliceGroups" in
        encoder.cfg, same as FmoNumSliceGroups, which should be erased later
257     int slice_group_map_type;
258
259     unsigned int *top_left;                          //!< top_left and
        bottom_right store values indicating foregrounds
260     unsigned int *bottom_right;
261     byte *slice_group_id;                    //!< slice_group_id is for slice
        group type being 6
262     int *run_length_minus1;                 //!< run_length_minus1 is for
        slice group type being 0
263
264     int slice_group_change_direction_flag;
265     int slice_group_change_rate_minus1;
266     int slice_group_change_cycle;
267
268     int redundant_pic_flag;    //! encoding of redundant pictures
269     int pic_order_cnt_type;    //! POC type
270
271     int context_init_method;
272     int model_number;
273     int Transform8x8Mode;
```

```
274     int ReportFrameStats;
275     int DisplayEncParams;
276     int Verbose;
277
278     //! Rate Control parameters
279     int RCEnable;
280     int bit_rate;
281     int SeinitialQP;
282     unsigned int basicunit;
283     int channel_type;
284     int RCUpdateMode;
285     double RCIoverPRatio;
286     double RCBoverPRatio;
287     double RCISliceBitRatio;
288     double RCBSliceBitRatio[RC_MAX_TEMPORAL_LEVELS];
289     int    RCMinQP[NUM_SLICE_TYPES];
290     int    RCMaxQP[NUM_SLICE_TYPES];
291     int    RCMaxQPChange;
292
293     // Motion Estimation related parameters
294     int    UseMVLimits;
295     int    SetMVXLimit;
296     int    SetMVYLimit;
297
298     // Search Algorithm
299     SearchType SearchMode;
300
301     // UMHEX related parameters
302     int UMHexDSR;
303     int UMHexScale;
304
305     // EPZS related parameters
306     int EPZSPattern;
307     int EPZSDual;
308     int EPZSFixed;
309     int EPZSTemporal;
310     int EPZSSpatialMem;
311     int EPZSBlockType;
312     int EPZSMinThresScale;
313     int EPZSMaxThresScale;
314     int EPZSMedThresScale;
315     int EPZSSubPelGrid;
316     int EPZSSubPelME;
```

```
317     int EPZSSubPelMEBiPred;
318     int EPZSSubPelThresScale;
319
320     // Lambda Params
321     int UseExplicitLambdaParams;
322     int UpdateLambdaChromaME;
323     double LambdaWeight[6];
324     double FixedLambda[6];
325
326     char QOffsetMatrixFile[FILE_NAME_SIZE];          //!< Quantization Offset
            matrix cfg file
327     int  OffsetMatrixPresentFlag;                    //!< Enable Explicit
            Quantization Offset Matrices
328
329     int AdaptiveRounding;                            //!< Adaptive Rounding
            parameter based on JVT-N011
330     int AdaptRoundingFixed;                          //!< Global rounding for
            all qps
331     int AdaptRndPeriod;                              //!< Set period for
            adaptive rounding of JVT-N011 in MBs
332     int AdaptRndChroma;
333     int AdaptRndWFactor [2][NUM_SLICE_TYPES];        //!< Weighting factors for
             luma component based on reference indicator and slice type
334     int AdaptRndCrWFactor[2][NUM_SLICE_TYPES];       //!< Weighting factors for
             chroma components based on reference indicator and slice type
335
336 //////////////////////////////////////////////////////////////////////
337     // Fidelity Range Extensions
338     int ScalingMatrixPresentFlag;
339     int ScalingListPresentFlag[12];
340
341     int cb_qp_index_offset;
342     int cr_qp_index_offset;
343     // Lossless Coding
344     int LosslessCoding;
345
346     // Fast Mode Decision
347     int EarlySkipEnable;
348     int SelectiveIntraEnable;
349     int DisposableP;
350     int DispPQPOffset;
351
352     //Redundant picture
```

```
353     int NumRedundantHierarchy;    //!< number of entries to allocate redundant
            pictures in a GOP
354     int PrimaryGOPLength;         //!< GOP length of primary pictures
355     int NumRefPrimary;            //!< number of reference frames for primary
            picture
356
357     // tone mapping SEI message
358     int ToneMappingSEIPresentFlag;
359     char ToneMappingFile[FILE_NAME_SIZE];     //!< ToneMapping SEI message cfg
            file
360
361     // prediction structure
362     int PreferDispOrder;        //!< Prefer display order when building the
            prediction structure as opposed to coding order
363     int PreferPowerOfTwo;       //!< Prefer prediction structures that have
            lengths expressed as powers of two
364     int FrmStructBufferLength; //!< Number of frames that is populated every
            time populate_frm_struct is called
365
366     int separate_colour_plane_flag;
367     double WeightY;
368     double WeightCb;
369     double WeightCr;
370     int UseRDOQuant;
371     int RDOQ_DC;
372     int RDOQ_CR;
373     int RDOQ_DC_CR;
374     int RDOQ_QP_Num;
375     int RDOQ_CP_Mode;
376     int RDOQ_CP_MV;
377     int RDOQ_Fast;
378
379     int EnableVUISupport;
380     // VUI parameters
381     VUIParameters VUI;
382     // end of VUI parameters
383
384     int  MinIDRDistance;
385     int stdRange;                          //!< 1 - standard range, 0 - full
            range
386     int videoCode;                         //!< 1 - 709, 3 - 601:  See
            VideoCode in io_tiff.
387
```

```
388   /* KATCIPIS adding configuration for Object detection/ tracking */
389   int object_detection_enable;                    //!< Enable/Disable
          object detection.
390   int object_detection_min_width;                 //!< Min width of
          the object that will be detected.
391   int object_detection_min_height;                //!< Min height of
          the object that will be detected.
392   int object_detection_search_hysteresis;         //!< Search for new
          object hysteresys (in frames).
393   int object_detection_tracking_hysteresis;       //!< Confirm tracked
           object existence hysteresis (in frames).
394   char object_detection_training_file[FILE_NAME_SIZE]; //!< File containing
           the training info used on the object detection.
395 };
396
397 #endif
```

12 ANEXO D – ALTERAÇÕES REALIZADAS NA CONFIGURAÇÃO DO CODIFICADOR DO SOFTWARE DE REFERÊNCIA

Essas alterações representam as alterações realizadas no arquivo de configuração que vem junto com a distribuição do software de referência, apenas os campos que sofreram alterações são apresentados aqui, os que não são apresentados foram mantidos com o mesmo valor que o arquivo original.

```
1
2  ProfileIDC                = 66 # Profile IDC (66=baseline, 77=main, 88=
       extended; FREXT Profiles: 100=High, 110=High 10, 122=High 4:2:2, 244=
       High 4:4:4, 44=CAVLC 4:4:4 Intra, 118=Multiview High Profile, 128=Stereo
        High Profile)
3  IntraPeriod               = 12  # Period of I-pictures  (0=only first)
4  DisableSubpelME           = 1   # Disable Subpixel Motion Estimation (0=off/
       default, 1=on)
5  SearchRange               = 16  # Max search range
6  MEDistortionHPel          = 0   # Select error metric for Half-Pel ME    (0:
       SAD, 1: SSE, 2: Hadamard SAD)
7  MEDistortionQPel          = 0   # Select error metric for Quarter-Pel ME (0:
       SAD, 1: SSE, 2: Hadamard SAD)
8  MDDistortion              = 0   # Select error metric for Mode Decision  (0:
       SAD, 1: SSE, 2: Hadamard SAD)
9  NumberReferenceFrames = 1   # Number of previous frames used for inter
       motion search (0-16)
10 GenerateMultiplePPS    = 0   # Transmit multiple parameter sets. Currently
       parameters basically enable all WP modes (0: diabled, 1: enabled)
11 NumberBFrames             = 0  # Number of B coded frames inserted (0=not used
       )
12 HierarchicalCoding       = 0  # B hierarchical coding (0= off, 1= 2 layers,
        2= 2 full hierarchy, 3 = explicit)
13 BiPredMESubPel           = 0   # Bipredictive ME Subpixel Consideration (0:
       disabled, 1: single level, 2: dual level)
14 SymbolMode               =  0  # Symbol mode (Entropy coding method: 0=UVLC,
```

```
    1=CABAC)
15 WPMCPrecision            =  0     # Improved Motion Compensation Precision
    using WP based methods.
16 RDPictureDecision        =  0     # Perform multiple pass coding and make
    RD optimal decision among them
17 FastCrIntraDecision    =  1  # Fast Chroma intra mode decision (0:off, 1:on
    )
18 UseExplicitLambdaParams  =  2    # Use explicit lambda scaling parameters
    (0:disabled, 1:enable lambda weight, 2: use explicit lambda value)
19 FixedLambdaISlice        =  29.0  # Fixed Lambda value for I slices
20 FixedLambdaPSlice        =  29.0  # Fixed Lambda value for P slices
21 FixedLambdaBSlice        =  29.0  # Fixed Lambda value for B slices
22 FixedLambdaRefBSlice     =  29.0  # Fixed Lambda value for Referenced B
    slices
23 FixedLambdaSPSlice       =  29.0  # Fixed Lambda value for SP slices
24 FixedLambdaSISlice       =  29.0  # Fixed Lambda value for SI slices
25 RateControlEnable        = 1      # 0 Disable, 1 Enable
26 Bitrate                  = 996000 # Bitrate(bps)
27 BasicUnit                = 1      # Number of MBs in the basic unit
28 RCMaxQPPSlice            = 44     # maximum P Slice QP value for rate control
29 RCMaxQPBSlice            = 46     # maximum B Slice QP value for rate control
30 RCMaxQPISlice            = 36     # maximum I Slice QP value for rate control
31 RCMaxQPSISlice           = 36     # maximum SI Slice QP value for rate
    control
32 Transform8x8Mode         = 0      # (0: only 4x4 transform, 1: allow using 8
    x8 transform additionally, 2: only 8x8 transform)
33 OffsetMatrixPresentFlag  = 1     # Enable Explicit Offset Quantization
    Matrices (0: disable 1: enable)
34 AdaptRoundingFixed       = 1     # Enable Global Adaptive rounding for all
    qps (0: disable, 1: enable - default/old)
35 AdaptRndPeriod           = 16    # Period in terms of MBs for updating
    rounding offsets.
36 AdaptRndWFactorIRef      = 4     # Adaptive Rounding Weight for I/SI slices
    in reference pictures /4096
37 AdaptRndWFactorPRef      = 4     # Adaptive Rounding Weight for P/SP slices
    in reference pictures /4096
38 AdaptRndWFactorBRef      = 4     # Adaptive Rounding Weight for B slices in
    reference pictures /4096
39 AdaptRndWFactorINRef     = 4     # Adaptive Rounding Weight for I/SI slices
    in non reference pictures /4096
40 AdaptRndWFactorPNRef     = 4     # Adaptive Rounding Weight for P/SP slices
    in non reference pictures /4096
41 AdaptRndWFactorBNRef     = 4     # Adaptive Rounding Weight for B slices in
```

```
    non reference pictures /4096
42  AdaptRndCrWFactorIRef     = 4     # Chroma Adaptive Rounding Weight for I/SI
    slices in reference pictures /4096
43  AdaptRndCrWFactorPRef     = 4     # Chroma Adaptive Rounding Weight for P/SP
    slices in reference pictures /4096
44  AdaptRndCrWFactorBRef     = 4     # Chroma Adaptive Rounding Weight for B
    slices in reference pictures /4096
45  AdaptRndCrWFactorINRef    = 4     # Chroma Adaptive Rounding Weight for I/SI
    slices in non reference pictures /4096
46  AdaptRndCrWFactorPNRef    = 4     # Chroma Adaptive Rounding Weight for P/SP
    slices in non reference pictures /4096
47  AdaptRndCrWFactorBNRef    = 4     # Chroma Adaptive Rounding Weight for B
    slices in non reference pictures /4096
48  UseRDOQuant               =  0 # Use Rate Distortion Optimized Quantization
    (0=disable, 1=enable)
49  RDOQ_DC                   =  0 # Enable Rate Distortion Optimized
    Quantization for DC components (0=disable, 1=enable)
50  RDOQ_CR                   =  0 # Enable Rate Distortion Optimized
    Quantization for Chroma components (0=disable, 1=enable)
51  RDOQ_DC_CR                =  0 # Enable Rate Distortion Optimized
    Quantization for Chroma DC components (0=disable, 1=enable)
52  RDOQ_QP_Num               =  1 # 1-9: Number of QP tested in RDO_Q (I/P/B
    slice)
53  RDOQ_CP_Mode              =  1 # copy Mode from first QP tested
54  RDOQ_CP_MV                =  1 # copy MV from first QP tested
55  RDOQ_Fast                 =  1 # Fast RDOQ decision method for multiple QPs
56  SearchMode                = -1    # Motion estimation mode
```

Inserção de metadados referentes a detecção de padrões no H.264

Tiago César Katcipis[1]

[1]Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC)
Santa Catarina – SC – Brazil

katcipis@inf.ufsc.br

Abstract. *To store and transmit video the use of encoders has been increasing, MPEG 4 part 10, also known as H.264, stands out among these encoders. This work integrates the Haar classifier to the reference MPEG 4 part 10 encoder to detect objects and explores the motion estimation algorithm of the encoder to track the detected objects. Detected objects and tracking information are represented in the form of metadata and bitstream are transported in the video messages using Supplemental Enhancement Information. The tests showed that the generated video is in accordance with the MPEG 4 part 10, together with a satisfactory computational performance. The results obtained in the decoder, retrieving the metadata and presenting them, were also satisfactory, showing the viability of building a object tracking system embedded on the encoder.*

Resumo. *Para realizar o armazenamento e transmissão de vídeo tem se utilizado cada vez mais codificadores, dentre esses codificadores se destaca o MPEG 4 parte 10, também conhecido como H.264. Este trabalho integra um classificador Haar ao codificador de referência do padrão MPEG 4 parte 10 para realizar a detecção de objetos e explora o algoritmo de estimativa de movimento do codificador para realizar o tracking do objeto. Os objetos detectados e as informações de tracking são representados na forma de metadados e são transportados no bitstream do vídeo utilizando mensagens Supplemental Enhancement Information. Os testes realizados mostram que o codificador gerou vídeos em conformidade com o padrão MPEG 4 parte 10, junto com um desempenho computacional satisfatório. Os resultados obtidos no decodificador, ao recuperar os metadados e apresentá-los, também foram satisfatórios mostrando a viabilidade de construir um sistema de tracking de objetos embutido no codificador.*

Introdução

Atualmente tem se tornado comum em soluções de segurança o uso de detecção de padrões como detecção facial, detecção de objetos específicos, cercas virtuais, alarmes, etc. Dispositivos de gravação de vídeo em alta definição estão se tornando cada vez mais acessíveis, estando presentes até mesmo em celulares, porém como é inviável dispor de longos trechos de vídeo em alta resolução sem compactação pois estes consomem um grande espaço de armazenamento e não é possível transmiti-los em larga escala com os meios de comunicação existentes, esses vídeos em alta definição são usualmente compactados.

A maior parte dos algoritmos de identificação de objetos e algoritmos biométricos trabalham com informações não codificadas, nesse caso, o processamento de multiplos vídeos exigiria que esses vídeos fossem decodificados primeiro e então processados. Alguns algoritmos de identificação de objetos trabalham com informações codificadas, um exemplo é o detector de faces proposto em [e Shang Hong Lai 2009], mas na conclusão do mesmo é possível se observar que apesar do resultado ser satisfatório, o processamento em um identificador de objetos utilizando informações brutas foi superior (no caso a comparação foi realizada com o classificador Haar do OpenCV).

Considerando que a busca por objetos de interesse fosse realizada no vídeo compactado (não sendo necessário decodificar o vídeo), em um caso de uso como o de um aeroporto onde existem muitas câmeras de segurança, ainda seria necessário um grande poder computacional para realizar a busca de objetos de interesse em todos vídeos ao mesmo tempo, já que normalmente se espera que um sistema de segurança tenha um tempo de resposta rápido.

Dessa maneira é interessante obter a maior quantidade possível de metadados a respeito de objetos de interesse, na fonte do vídeo, de maneira integrada ao processo de codificação, reaproveitando ao máximo qualquer informação que o processo de codificação possa fornecer. Ao invés de analisar os vídeos, será realizada uma análise dos metadados. Se um vídeo muito longo não possui nenhum metadado, não será necessário analisá-lo. Como metadados pode-se citar a presença de um objeto de interesse (sua posição e tamanho), padrões de movimento (útil em cercas virtuais) ou o objeto de interesse em alta resolução não compactado (facilita o processamento posterior desse objeto em um algoritmo biométrico).

Ao utilizar ao máximo as informações que o próprio codificador gera para realizar a identificação de padrões é interessante transportar os metadados gerados dentro do próprio bitstream do vídeo, isso facilita o desenvolvimento de um chip codificador na solução, pois não é necessário que os metadados encontrados sejam enviados a aplicação para serem transportados de outra maneira. Neste artigo será apresentado um estudo da integração de um algoritmo de detecção de padrões com as informações de estimativa de movimento calculadas pelo codificador MPEG 4 parte 10, gerando um *tracker* de objetos, capaz de enviar também objetos de interesse não compactados.

A estrutura deste artigo apresenta-se da seguinte forma. A seção explica como os metadados são extraídos e representados no sistema. A seção 0.2 apresenta as alterações realizadas no codificador, utilizando os componentes explicados em . Na seção 0.2 serão apresentados os testes realizados com o sistema desenvolvido. A seção 0.6 conclui este artigo, apresentando as contribuições deste trabalho e os trabalhos futuros.

Extração e representação dos metadados

0.1. Módulo extracted_metadata

A classe *Extracted_Metadata* abstrai a serialização, desserialização, salvamento e destruição dos diferentes tipos de metadados. O método *serialize* é responsável pela serialização, uma vez serializado o metadado será inserido diretamente no *bitstream* do vídeo como uma mensagem SEI (*Supplemental Enhancement Information*).

Foram implementadas duas subclasses da classe *Extracted_Metadata*, para repre-

sentar os metadados extraídos:

- *Extracted_Y_Image* : Representa o plano luma de um objeto de interesse, é composto basicamente do plano luma com o seu comprimento e altura. Ao detectar um objeto de interesse é possível extrair apenas o objeto e inserí-lo no bitstream. O objeto pode ser recuperado no decoder e salvo em um arquivo.
- *Extracted_Object_Bounding_Box* : Representa uma caixa delimitadora envolta de um objeto de interesse. Ela é composta de um identificador único do objeto, as coordenadas (x, y) da caixa, sua altura e seu comprimento. Com as informações de coordenadas da caixa delimitadora é possível desenhar a caixa delimitadora diretamente no vídeo e com o identificador único é possível realizar o *tracking* do objeto ao longo do vídeo.

0.2. Módulo metadata_extractor

Este módulo é responsável pela extração de metadados a partir de quadros brutos e das informações de estimativa de movimento fornecidas. Extraindo metadados do tipo *ExtractedObjectBoundingBox* e *ExtractedYImage*. Para realizar a detecção de um objeto de interesse no quadro foi utilizado o classificador Haar do OpenCV. O termo histerese se refere a quantidade de quadros necessários para realizar a execução do classificador Haar, gerando um atraso configurável na detecção de um novo objeto, ou ao confirmar a existência de um objeto previamente detectado. O método *extract_object_bounding_box* extrai do quadro uma caixa delimitadora que representa a área onde se encontra o objeto de interesse no quadro e retorna como um objeto *ExtractedMetadata*.

As histereses configuradas, junto com as informações de estimativa de movimento, visam utilizar os algoritmos já existentes no codificador para reduzir o custo computacional do *tracking* de um objeto no vídeo. Ao invés de realizar o processamento de todos os quadros brutos no classificador Haar para realizar o *tracking* do objeto ao longo do vídeo, a histerese diminui o custo computacional por atualizar a posição do objeto em intervalos fixos. Os vetores de movimento calculados pelo codificador são utilizados para suavizar o *tracking*, estimando a nova posição do objeto enquanto a histerese de *tracking* não estoura.

O mesmo se dá quando o objeto já foi detectado, não é necessário executar o classificador em todos os quadros, os vetores de estimativa de movimento nos dão uma idéia aproximada da posição do objeto, até que seja alcançada a histerese de *tracking*.

Alterações realizadas no codificador

O codificador utilizado no desenvolvimento do sistema foi o que se encontra no software de referência desenvolvido pelo JVT (Joint Video Team) e hospedado pelo Instituto Fraunhofer para telecomunicações, Instituto Heinrich Hertz. Esta implementação foi utilizada por ela ser referência para pesquisas e verificação de conformidade com a norma, a versão utilizada foi a 17.2.

Dentro da função *encode_one_frame*, o processamento do quadro bruto é realizado logo após a chamada de função *process_image*, neste momento também ocorre a extração do metadado e o mesmo é salvo no bitstream na forma de mensagem SEI *Unregistered Userdata*. Além dos quadros brutos, outra informação importante para o extrator

de metadados são os vetores de movimento dos blocos. Na função *code_a_plane* antes da chamada de função *DeblockFrame* todo o processo de estimativa de movimento já está completo, neste ponto é que os vetores de estimativa de movimento do quadro são repassados ao extrator de metadados para que ele realize a estimativa de movimento do objeto detectado (se o extrator estiver realizando *tracking*).

Testes realizados

Todos os testes utilizaram o mesmo arquivo de treinamento *haarcascade_frontalface_alt.xml*, que realiza a busca de faces frontais, este arquivo vem junto com a distribuição do OpenCV (que pode ser encontrada em http://sourceforge.net/projects/opencvlibrary/files) no diretório *data/haarcascades*. O tamanho mínimo do objeto de interesse configurado em todos os testes foi de 30 x 30 pixels.

A configuração da máquina e o sistema operacional utilizados nos testes foi:

- Processador Pentium(R) Dual-Core E5200 á 2.50GHz.
- 4 gigabytes de memória RAM.
- Sistema operacional Ubuntu 11.04 32 bits.

Foram realizados 5 testes em cada vídeo, todos com a mesma configuração de codificação:

- Sem *tracking* de objetos.
- Com *tracking* ativado, com histerese de busca = 1 e histerese de *tracking* = 1. Dessa maneira as informações de estimativa de movimento do codificador não são utilizadas, o *tracking* é realizado utilizando apenas o classificador Haar.
- Com *tracking* ativado, com histerese de busca = 5 e histerese de *tracking* = 10. Essa configuração faz um menor uso das informações de estimativa de movimento para realizar *tracking* de um objeto.
- Com *tracking* ativado, com histerese de busca = 10 e histerese de *tracking* = 30. Essa configuração depende mais das informações de estimativa de movimento para realizar *tracking* de um objeto.
- Com *tracking* ativado, com histerese de busca = 10 e histerese de *tracking* = 60. Essa configuração é a que mais depende mais das informações de estimativa de movimento para realizar *tracking* de um objeto.

0.3. Vídeo Foreman - CIF - 300 quadros

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/foreman_cif.y4m e possui 3 situações diferentes, uma face frontal, a face fica de perfil em alguns momentos, e depois a face sai do vídeo.

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 300.
- comprimento = 352 pixels.
- altura = 288 pixels.
- espaço de cor = YUV, 4:2:0.

Com a detecção de objetos ocorrendo em todos os quadros percebe-se uma alta taxa de falsos negativos, já que o rosto no vídeo fica de perfil em alguns momentos e o classificador Haar não consegue identificar o rosto em perfil, perdendo assim o *tracking* do rosto. Com a detecção de objetos em todos os quadros também ocorreu um falso positivo quando a camêra se move e deixa de gravar o rosto. A medida que a histerese de *tracking* é aumentada, além de não ocorrer o falso positivo, os movimentos que o rosto realiza são acompanhados com maior suavidade e com uma menor taxa de falsos negativo (com a histerese de *tracking* configurada para 60 quadros não ocorreu nenhum falso negativo).

0.4. Vídeo Akiyo - QCIF - 300 quadros

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/akiyo_qcif.y4m e consiste basicamente de 300 quadros positivos (existe o objeto de interesse ao longo de todo o vídeo, nesse caso uma face frontal).

Especificações do vídeo:

- quadros por segundo = 29.97.
- total de quadros = 300.
- comprimento = 176 pixels.
- altura = 144 pixels.
- espaço de cor = YUV, 4:2:0.

Como este vídeo é composto de apenas uma pessoa, realizando pequenos movimentos suaves, os resultados com histerese de *tracking* alta foram bons. A ativação de detecção de objetos em todos os quadros, utilizando apenas o classificador Haar para realizar o *tracking*, tornou o processo de codificação aproximadamente 10,94% mais lento.

0.5. Vídeo *Pedestrian area* - 375 quadros - 1080p (1920 x 1080)

Esse vídeo pode ser encontrado em http://media.xiph.org/video/derf/y4m/pedestrian_area_1080p25.y4m. Ele consiste basicamente de vários pedestres caminhando em uma rua, possui faces frontais e faces em perfil se movendo continuamente.

Especificações do vídeo:

- quadros por segundo = 25.
- total de quadros = 375.
- comprimento = 1920 pixels.
- altura = 1080 pixels.
- espaço de cor = YUV, 4:2:0.

Com a detecção de objetos ocorrendo em todos os quadros ocorre uma grande quantidade de falsos positivos, e a limitação de realizar o *tracking* de apenas um objeto por vez ficou bem evidente já que neste vídeo existem diversas pessoas andando ao mesmo tempo o sistema consegue realizar o *tracking* de algumas faces por um curto periodo de tempo, mas logo depois detecta outra face e passa a realizar o *tracking* dessa face ao invés da que foi previamente detectada.

A utilização das histereses alterou um pouco o comportamento do sistema (em geral ele manteve uma alta taxa de falsos positivos), a primeira detecção que ocorre com sucesso realiza o *tracking* da face corretamente, mas após essa primeira detecção ocorre uma série de falsos positivos, e em algumas detecções positivas o *tracking* não consegue acompanhar a face por ela estar se movendo rápido demais.

0.6. Vídeo *Pedestrian area* - 375 quadros - 720p (1280 x 720)

Neste teste a resolução do vídeo *Pedestrian area* foi reduzida de *1080p* (1920 x 1080) para *720p* (1280 x 720), visando a constatação da diferença do impacto do sistema de *tracking* ao processar o mesmo vídeo com resoluções diferentes. O escalonamento do vídeo foi realizado utilizando o Gstreamer.

A diminuição da resolução do vídeo não gerou nenhuma diferença na qualidade do *tracking*, porém foi possível perceber um aumento no atraso no tempo total de codificação em todas as configurações.

Conclusão

Testes realizados tanto com o decodificador de referência como com o Gstreamer mostraram que o uso de mensagens *Supplemental Enhancement Information* do tipo *Unregistered Userdata* para transportar os metadados diretamente no *bitstream* do vídeo não alteraram a conformidade do mesmo com o padrão, sendo possível exibir um vídeo com metadados embutidos em qualquer decodificador MPEG 4 parte 10. Executar o classificador Haar em todos os quadros do vídeo mostrou um aumento considerável no custo computacional do codificador.

Utilizou-se informações de estimativa de movimento calculadas pelo codificador para estimar o movimento do objeto, evitando a necessidade de executar o classificador Haar em todos os quadros do vídeo, dessa maneira constatou-se uma significativa redução do custo computacional reutilizando informações geradas pelo processo de codificação.

Como toda a extração do metadado e inserção dele dentro do *bitstream* é feita internamente no codificador isso facilita a construção de um chip codificador H.264 que realize *tracking* de objetos, esse chip codificador poderia ter grande parte do classificador Haar acelerado também em hardware, dentro do chip. Como pode ser visto em [e Shih-Lien Lu 2008], obtêm-se um grande aumento no desempenho da detecção de objetos ao utilizar o classificador implementado em uma FPGA.

Este artigo mostrou a viabilidade de se utilizar a estimativa de movimento gerada pelo codificador para auxiliar o *tracking* de objetos e do envio dessas informações através do *bitstream* do vídeo. Além das informações de *tracking* foi possível enviar o objeto detectado não compactado como metadado, o que pode ser útil em algoritmos de identificação biométrica. No momento da análise dos vídeos é necessário analisar apenas os metadados que estão inseridos no *bitstream*, grande parte do processamento já foi realizado durante o processo de codificação.

Como possíveis trabalhos futuros, cita-se:

- Fazer um melhor uso das informações geradas pelo processo de codificação para gerar heurísticas mais inteligentes. Um exemplo seria utilizar histereses dinâmicas, quando existe muito movimento no vídeo as histereses diminuem, mas quando não existe movimento as histereses aumentam.
- Estender o sistema para realizar a detecção e *tracking* de múltiplos objetos (com mesma forma) simultaneamente.
- Buscar no classificador Haar cálculos que já possam ter sido feitos pelo codificador, melhorando a integração dos dois.

- Utilizar outro algoritmo de detecção de padrões que tenha uma maior interseção com os algoritmos presentes no codificador.
- Utilizar apenas as informações de estimativa de movimento para a construção de cercas virtuais ou alarmes que não se importem com a forma do objeto mas com padrões de movimento suspeitos.
- Desenvolver um chip codificador H.264 integrado ao classificador Haar integrado no chip, realizando a detecção e *tracking* de objetos em tempo real. Cita-se [e Shih-Lien Lu 2008] como exemplo de um classificador Haar acelerado em hardware.

References

e Shang Hong Lai, S. S. Z. (2009). *FACE DETECTION DIRECTLY FROM H.264 COMPRESSED VIDEO WITH CONVOLUTIONAL NEURAL NETWORK*. IEEE.

e Shih-Lien Lu, C. G. (2008). *Novel FPGA Based Haar Classifier Face Detection Algorithm Acceleration*. IEEE.

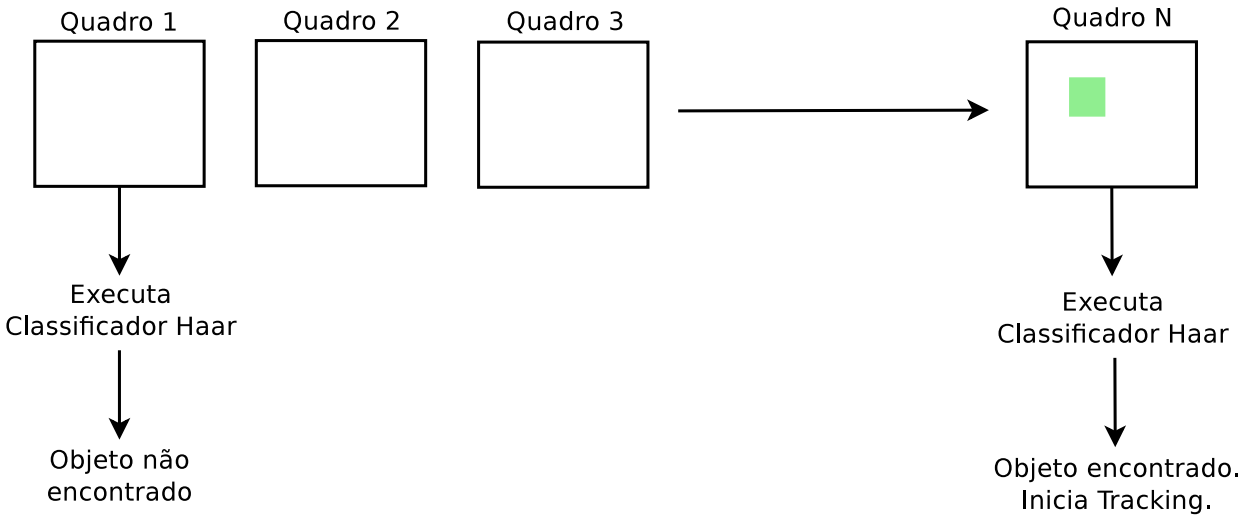

Figure 1. Exemplo do funcionamento da histerese de busca.

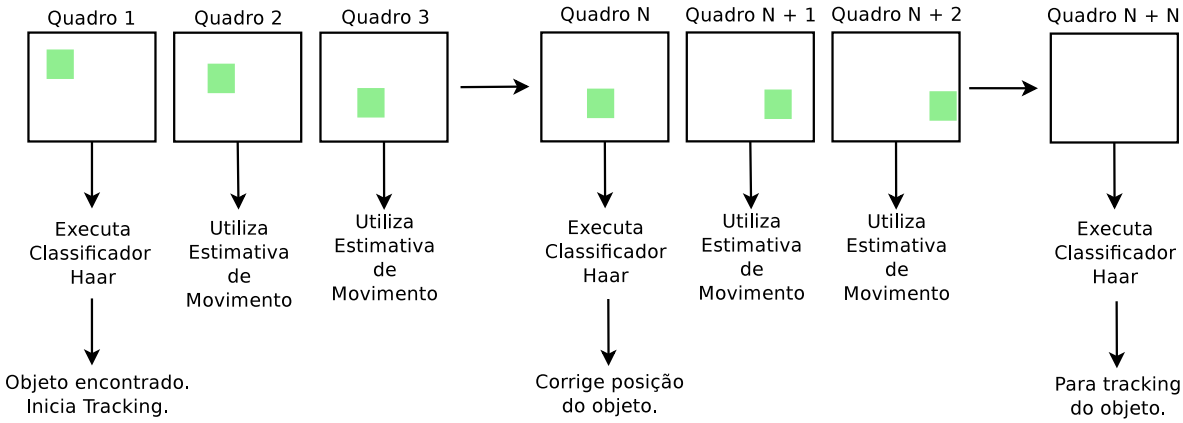

Figure 2. Exemplo do funcionamento da histerese de *tracking*.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Atraso no tempo total codificação	0%	9,87%	1,75%	0,85%	0,65%
Aumento bit-stream	0%	0,36%	0,43%	0,55%	0,66%

Table 1. Desempenho do sistema com o vídeo Foreman.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Atraso no tempo total codificação	0%	10,94%	1,27%	0,59%	0,39%
Aumento bit-stream	0%	3,83%	3,77%	3,71%	3,71%

Table 2. Desempenho do sistema com o vídeo Akiyo.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Atraso no tempo total codificação	0%	5,27%	0,08%	0,04%	0,005%
Aumento bit-stream	0%	0,207%	0,209%	0,211%	0,210%

Table 3. Desempenho do sistema com o vídeo *Pedestrian area*.

	***Tracking* Desabilitado**	***Tracking* habilitado, sem histerese**	**Histerese de busca = 5, *tracking* = 10**	**Histerese de busca = 10, *tracking* = 30**	**Histerese de busca = 10, *tracking* = 60**
Atraso no tempo total codificação	0%	11,09%	0,84%	0,44%	0,23%
Aumento bit-stream	0%	0,34%	0,37%	0,34%	0,35%

Table 4. Desempenho do sistema com o vídeo *Pedestrian area - 720p*.