5
Experimentos e Resultados

Neste capítulo, serão explicados os experimentos e resultados obtidos sobre o desempenho do *codec* DVC implementado e simulado pela ferramenta Open DVC.

Por questões de desempenho, optou-se pela versão da ferramenta **Open DVC Lab v1.0** para realização dos experimentos, ou seja, sua versão com chamada via linha de comando. Esta versão tem execução mais simples e rápida, já que ambas as versões da ferramenta possuem o mesmo resultado de saída e executam o experimento com as mesmas entradas, sendo diferentes apenas na interface gráfica elaborada que contempla a versão 2.0.

A medida objetiva de distorção utilizada foi o PSNR (Razão Sinal/Ruído de Pico, em inglês *Peak Signal-to-Noise Ratio*). Ressalta-se que vários autores consideraram que, apesar de ser uma medida precisa da distorção do ponto de vista estatístico, o PSNR não pode ser considerado uma medida definitiva da qualidade do vídeo codificado, uma vez que essa medida definitiva só poderia ser aferida pelo complexo comportamento do Sistema Visual Humano (SVH), de simulação computacional impossível, sendo necessária, portanto, a avaliação real de um vídeo e seus quadros por uma população mínima, e com diversos perfis, para definir a acerácea da técnica implementada.

Porém, devido à complexidade e ao custo dessa medida subjetiva, a aferição objetiva ainda é a mais utilizada na literatura pertinente e aceita como parâmetro de medição científico das técnicas de codificação de vídeo. Mesmo assim, esses testes podem ser realizados futuramente e seus resultados correlacionados com o resultado da medição objetiva.

Além do teste do PSNR, serão realizados testes de tempo de codificação, através dos quais se tem uma medida da complexidade e esforço computacional requeridos. Os testes de tempo de codificação vêm sendo utilizados em publicações científicas [48], podendo ser considerados bons parâmetros para comparação da complexidade entre diversos *codecs*. Com essa medida podemos responder, por exemplo, se realmente conseguimos deslocar a complexidade do

codec para o decodificador atingindo uma razoável simplicidade no codificador, premissas da arquitetura DVC.

O *framework* DVC proposto foi implementado inicialmente na ferramenta de programação matemática MATLAB, gerando um código interpretado, que tem um tempo de processamento bem superior ao código implementado em linguagem orientada a objetos. Porém, as medidas objetivas atingiram praticamente o mesmo resultado, a menos de algumas diferenças de precisão inerentes de cada linguagem, como era de se esperar, já que a linguagem ou ferramenta pouco deve influenciar nos resultados numéricos dos cálculos.

Posteriormente, o *codec* foi implementado em Java e as medidas de qualidade também foram definidas num dos pacotes para compararmos os resultados e a sincronia entre os dois *codecs*, que devem retratar as mesmas operações e consequentemente os mesmo resultados.

5.1
Configurações e condições de execução dos experimentos

Como fora dito no capítulo 2 desta dissertação, o trabalho com os maiores detalhes sobre resultados e metodologias utilizadas foi o desenvolvido pelo projeto DISCOVER, assim, utilizaremos as mesmas sequências e metodologias do referido trabalho para termos alguma referência em relação à qualidade do *codec* proposto. Para tanto, utilizaremos as sequências **Coastguard**, **Foreman**, **Hall Monitor** e **Soccer**, o que nos dará uma diversidade interessante de características no que se refere às sequências em relação à textura e grau de movimento, importante para chegarmos a conclusões mais consistentes e embasarmos nossos estudos. Essa diversidade é importante de ser analisada, pois isso vai gerar uma maior dificuldade no decodificador por ocasião da geração da informação lateral, principalmente nas operações de estimação do movimento e compensação do movimento para sequências com alto grau de movimento, testando assim, se o *codec* pode ser considerado eficiente para diferentes tipos de vídeo, o que está dentro da realidade cotidiana. Tais características são detalhadas abaixo para cada uma das sequências utilizadas:

- *Coastguard*: dois barcos se movimentando em um lago a baixa velocidade. Alta a média textura no plano de fundo na parte superior do vídeo, com água na parte inferior, que lhe confere características de difícil codificação. Baixo a médio movimento (figura 5.1).
- *Foreman*: vista aproximada (close-up) no rosto do capataz, seguida de um plano aberto da construção. Médio movimento (figura 5.2).
- *Hall Monitor*: Duas pessoas circulando a baixa velocidade em ambiente estático. Plano de fundo constante com grau de textura médio. Baixo movimento (figura 5.3).
- *Soccer*: partida de futebol, com diferentes cores, vários graus de movimento, plano de fundo com diferentes graus de textura. Alto movimento (figura 5.4).

Figura 5.1 – Quadros 0, 4 e 9 da sequência *Coastguard*: água e movimentação dos barcos

Figura 5.2 – Quadros 0, 4 e 9 da sequência *Foreman*: *close-up* da face com expressões, pouco movimento de cena

Figura 5.3 – Quadros 0, 4 e 9 da sequência *Hall Monitor*: pouca movimentação e variação de cena

Figura 5.4 – Quadros 0, 4 e 9 da sequência *Soccer*: muito movimento e textura

Todas as sequências que foram utilizadas e simuladas sua codificação e posterior decodificação, seguiram os mesmos parâmetros, a saber:

- *Group of Picture* de 2 (GOP-2);
- *Quarter Common Intermediate Format* (QCIF, 176 colunas x 144 linhas);
- Taxa de quadro de 30 fps (frames por segundo, 30 Hz);
- Subamostragem de Chroma 4:2:0;
- Número total de quadros: 300 quadros;
- Duração da sequência de vídeo: 10 segundos.

As informações e explicações sobre os parâmetros acima apresentados estão no Apêndice A deste trabalho.

Todos os testes foram realizados no mesmo computador, equipado com dois processadores Intel Xeon E5620 com quatro núcleos cada um (oito núcleos no total), com velocidade de 2.4GHz, 12MB de memória cache e memória RAM real de 20 GB. Este computador funcionou nativamente com um sistema operacional

Windows 7 Professional Edition de 64 bits, sobre o qual foram feitas virtualizações heterogêneas para diversificar os testes e enriquecer os resultados. Destacamos também, que foram feitos testes em computadores de menor capacidade de processamento e menor configuração, sendo os resultados de desempenho consideravelmente diferentes. Esses testes podem ser futuramente documentados para uma análise mais profunda dos requisitos computacionais do dispositivo real por ocasião da implementação do *codec*.

Os testes completos com as quatro sequências de vídeo seguiram a configuração que se mostra na figura 5.5, onde escolhemos três Vetores de Qualidade da Matriz de Quantização, dos dezoito existentes (numerados de 0 a 17), como fora explicado no capítulo 3. Para todas as sequências, realizamos a codificação e decodificação com os vetores número 5, 10 e 17. Foi criada uma estrutura de diretórios com o diretório de nível superior chamado **pasta_testes_dvc**, onde temos quatro subdiretórios, um para cada sequência de vídeo, com a seguinte sintaxe:

testes_*nomedaseqdevideo*_420_qcif_10s_300f_30fps, onde *nomedaseqdevideo* é o nome da sequência utilizada, por exemplo, Foreman, Hall Monitor, etc; 420 é a subamostragem de chroma, que no caso é 4:2:0; QCIF é a resolução de linhas e colunas correspondendo a 176 colunas por 144 linhas; 10s é a duração do vídeo; 300f é a quantidade total de quadros, que no caso são 300 e 30fps diz respeito à taxa de quadros, que no caso é de 30 quadros por segundo ou 30Hz.

Abaixo de cada subdiretório das sequências de vídeo, temos outros subdiretórios, para cada um dos testes a serem realizados, de acordo com os Vetores de Qualidade utilizados, portanto temos as pastas **mtx_quali_5**, **mtx_quali_10** e **mtx_quali_17**, correspondendo ao local onde estão as entradas de cada um dos testes e onde serão criadas e armazenadas suas saídas, repetidas quatro vezes, uma para cada vídeo.

Cada teste de cada sequência de vídeo, possui em seu subdiretório a sequência propriamente dita, como no exemplo da figura 5.6, **foreman_420_qcif_10s_300f_30fps.yuv**, além do *codec* através da ferramenta Open DVC **opendvclabv1.jar** e o arquivo de configuração **DVCConfig.cfg**, que é um textual com as configurações que serão lidas e executadas pelo *codec*,

definindo quais parâmetros serão utilizados nas operações e quais serão as saídas da codificação.

A figura 5.7, através do arquivo DVCConfig.cfg, exemplifica o teste para a sequência Foreman, codificada com a qualidade do vetor 5. Neste arquivo a linha 1 define o total de 300 quadros da sequência, a linha 2 define a resolução de 176 colunas de pixels em cada quadro, a linha 3 define a resolução de 144 linhas de pixels em cada quadro, a linha 4 define o vetor 5 do Vetor de Qualidade da Matriz de Quantização, a linha 5 define o local e o arquivo .yuv que será a entrada da sequência de vídeo bruto a ser codificada, a linha 6 define a subamostragem de chroma, que é 4:2:0, a linha 7 define a taxa de quadros em quadros por segundo, que no caso é 30, a linha 8 define o tamanho do bloco de divisão do quadro, com o tamanho 16 ou 4x4, a linha 9 define o padrão de cores da codificação, que no caso é para a sequência em Preto e Branco (PB), as linhas 10 e 11 definem as saídas da codificação, ou seja, as *bitstreams* a serem enviadas para o canal de comunicação, na linha 10 a *bitstream* originada pela codificação Wyner-Ziv e na linha 11 a *bitstream* originada pela codificação Intra-frame. As linhas 12 e 13 definem os arquivos de saída do *codec*, o arquivo .yuv reconstruído depois de codificado e decodificado e o arquivo onde serão armazenadas as estatísticas e informações do experimento. Vale observar que os arquivos das linhas 10 a 13 seguem uma sintaxe de formação que informa a sequência de vídeo que fora utilizada, a resolução de linhas e colunas e a qualidade requerida.

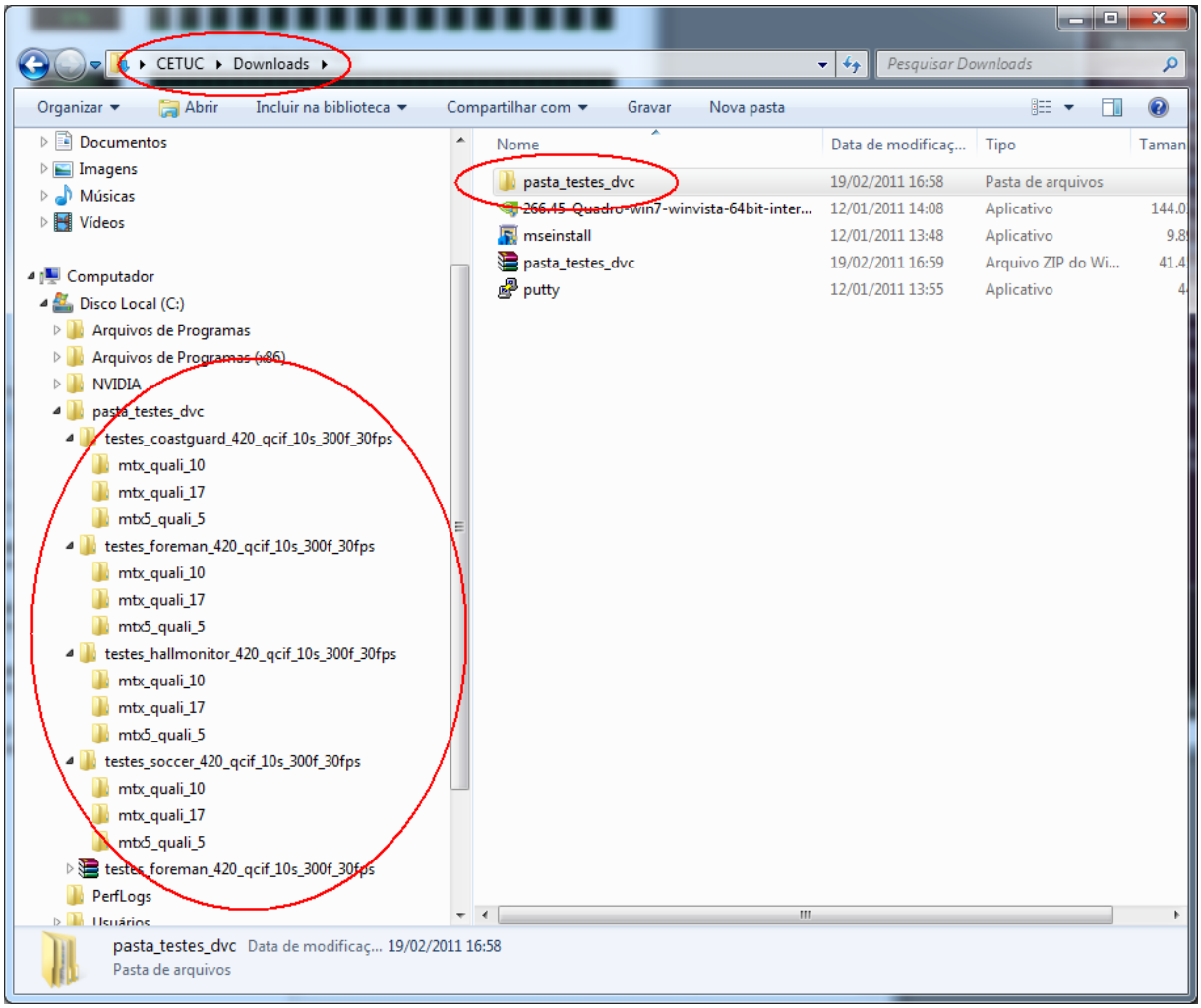

Figura 5.5 – Organização dos diretórios para a execução dos experimentos, vista pela interface gráfica

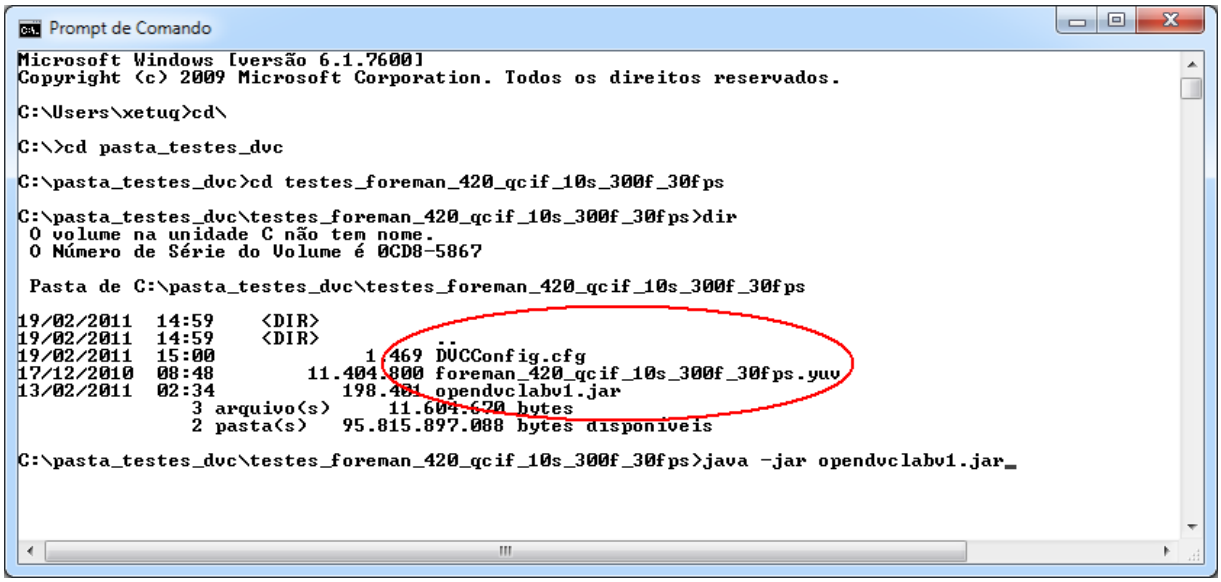

Figura 5.6 – Organização interna de um diretório de teste, vista pela linha de comando

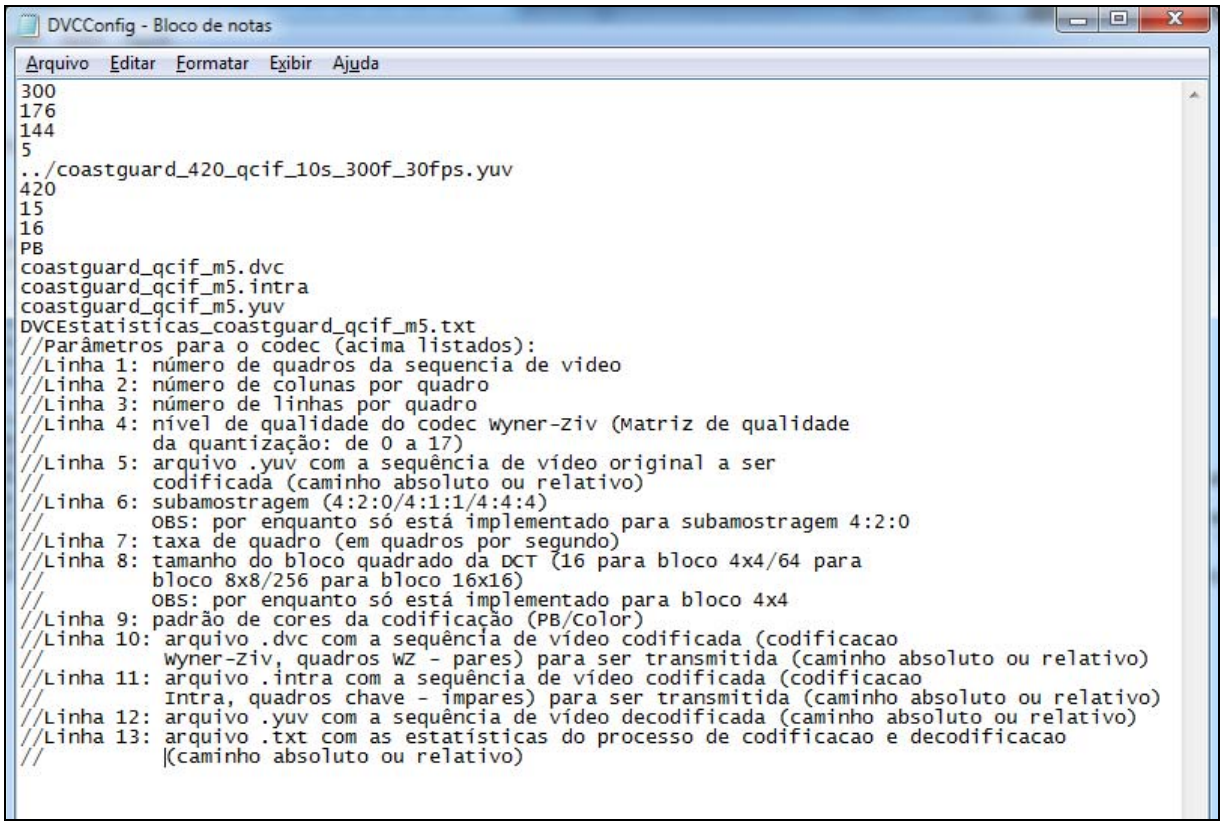

Figura 5.7 – Arquivo DVCConfig.cfg com exemplo de conteúdo interno

Uma vez organizado o diretório do experimento e configurado o arquivo dos parâmetros DVCConfig.cfg, a ferramenta Open DVC executa o processo de codificação e decodificação via linha de comando do executável **opendvclabv1.jar**, através do comando **java –jar opendvclabv1.jar**, conforme está ilustrado na figura 5.8.

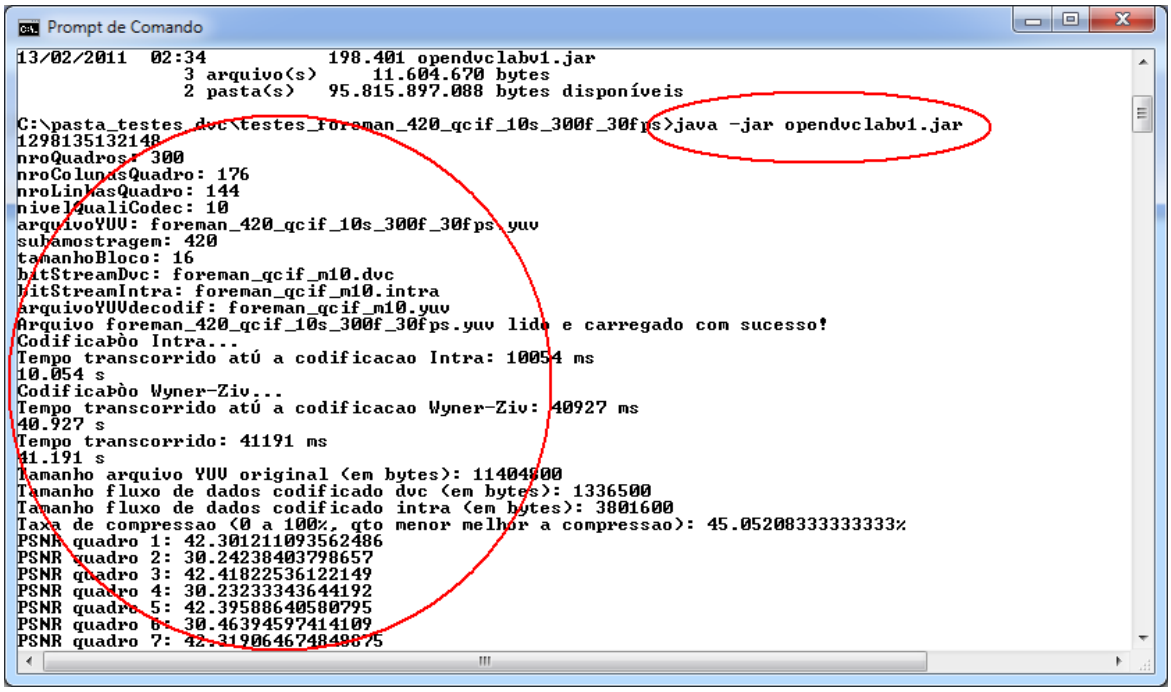

Figura 5.8 – Execução do aplicativo Java da ferramenta Open DVC via linha de comando com as saídas dos resultados iniciais

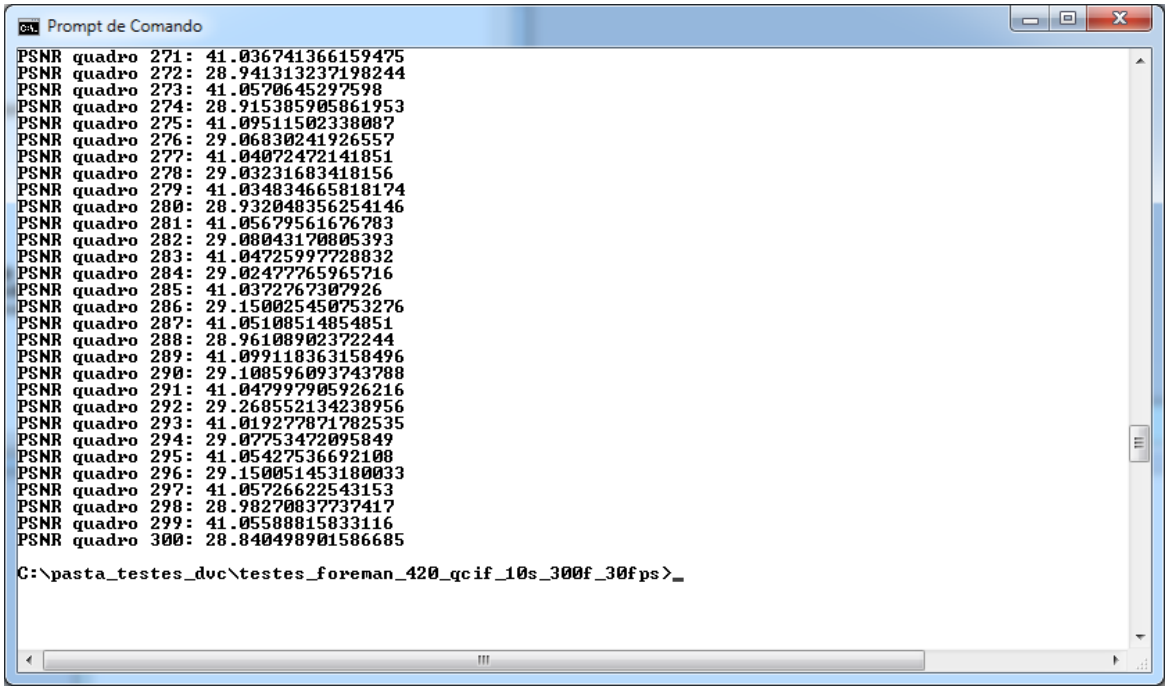

Figura 5.9 – Resultados finais de uma execução da ferramenta Open DVC, PSNR dos últimos quadros

Cabe ressaltar que este procedimento via linha de comando é apenas para garantir que as mesmas informações e estatísticas que foram armazenadas no arquivo definido na linha 13 do DVCConfig.cfg sejam mostrados em tempo de execução por ocasião da codificação, como vemos na figura 5.8. Caso queira, o usuário pode simplesmente dar um clique duplo no ícone do executável numa janela de exploração, em qualquer sistema operacional, que o *codec* será executado da mesma forma. Outra observação é que o único requisito para este programa funcionar é ter a Máquina Virtual Java (conhecida como JRE) instalada no seu computador, o que é comum hoje em dia, já que muitas aplicações de internet e *desktop* utilizam este programa.

Por fim, vemos na figura 5.10, que no mesmo diretório onde foi realizado o teste e chamado o executável do Open DVC, foram criados os arquivos de saída, conforme havíamos definido no arquivo de configuração. Esse é o comportamento padrão da ferramenta, porém, esses arquivos podem ser criados e armazenados em qualquer lugar, inclusive na rede, bastando para isso utilizar o caminho absoluto ou relativo por ocasião da configuração do DVCConfig.cfg.

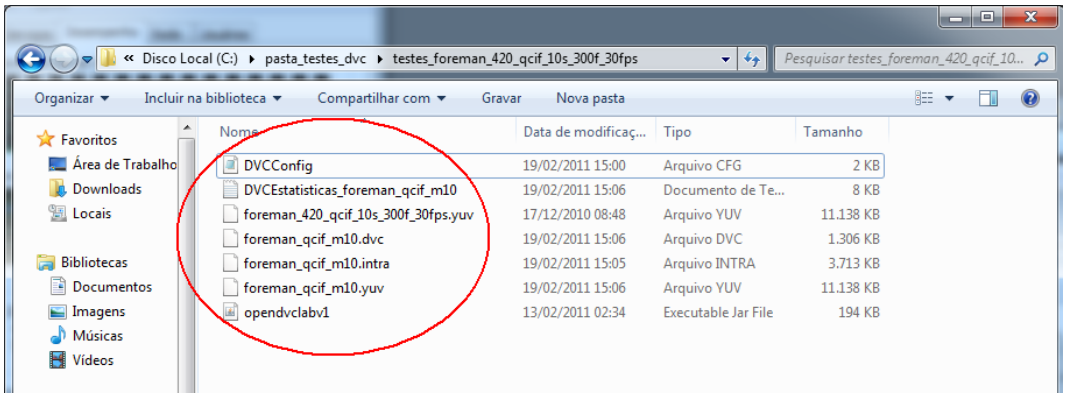

Figura 5.10 – Arquivos de saída da execução da ferramenta Open DVC

Analisando este primeiro teste, quanto ao desempenho do sistema, para uma única execução do *codec*, verificou-se que o consumo de processamento foi baixo, com pico de apenas 7% de uso de CPU, porém com um aumento maior no consumo de memória, que saiu de 2 GB com o sistema operacional apenas com os processos nativos em segundo plano, para quase 3 GB por ocasião da execução da codificação. Isso se justifica pela grande quantidade de operações matriciais, inclusive de multiplicação, onde um grande espaço de memória RAM é alocado

em tempo de execução. Quanto às unidades de tempo do processamento, mostradas na figura 5.8, verificou-se que a codificação Intra demorou um pouco mais de 10 segundos, enquanto o processo completo demorou um pouco mais de 41 segundos. Seguiremos esse padrão de análise e comparação para verificarmos a diferença de esforço computacional entre as diversas configurações.

Para analisarmos o comportamento da codificação em diferentes sistemas operacionais e com tamanho de endereçamento de memória diferentes, o *codec* foi testado através da virtualização de quatro máquinas sobre o mesmo computador, anteriormente descrito, sendo cada uma das máquinas configuradas para consumir até 4096 MB de memória RAM, nível que não foi atingido em sua plenitude em nenhuma dos testes realizados. A máquina virtual utilizada foi a VirtualBox da Oracle [51], que suportou para os experimentos, quatro configurações de sistemas operacionais, conforme descrito abaixo e ilustrado nas figuras 5.11 e 5.12:

- *Windows 7 Professional Edition* de 64 bits;
- *Windows 7 Professional Edition* de 32 bits;
- *Linux Ubuntu 10.04* de 32 bits;
- *Linux Ubuntu 10.04* de 64 bits.

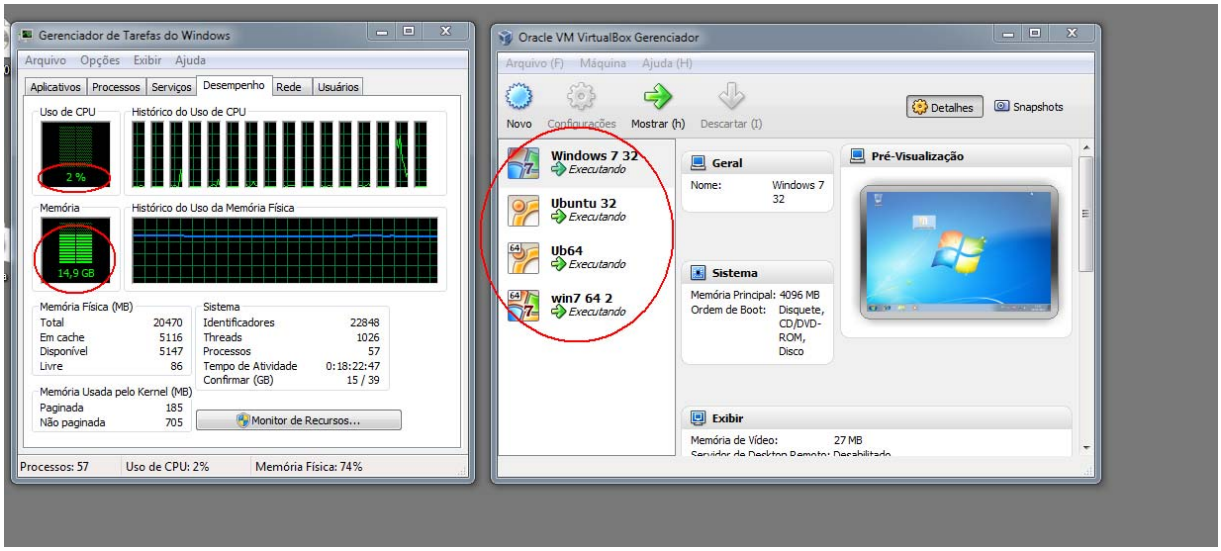

Figura 5.11 – Vista do Virtual Box executando os quatro sistemas operacionais em máquinas virtuais e o Gerenciador de Tarefas mostrando a carga computacional da tarefa

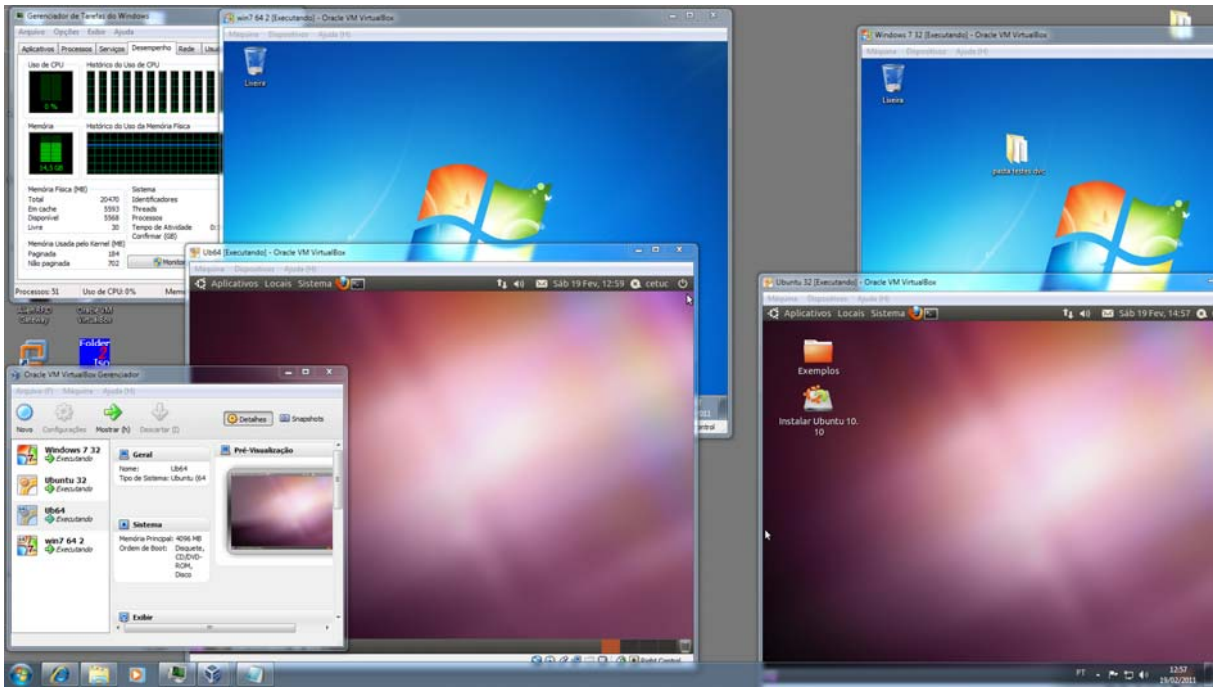

Figura 5.12 – Janela gráfica mostrando os quatro sistemas operacionais utilizados nos experimentos

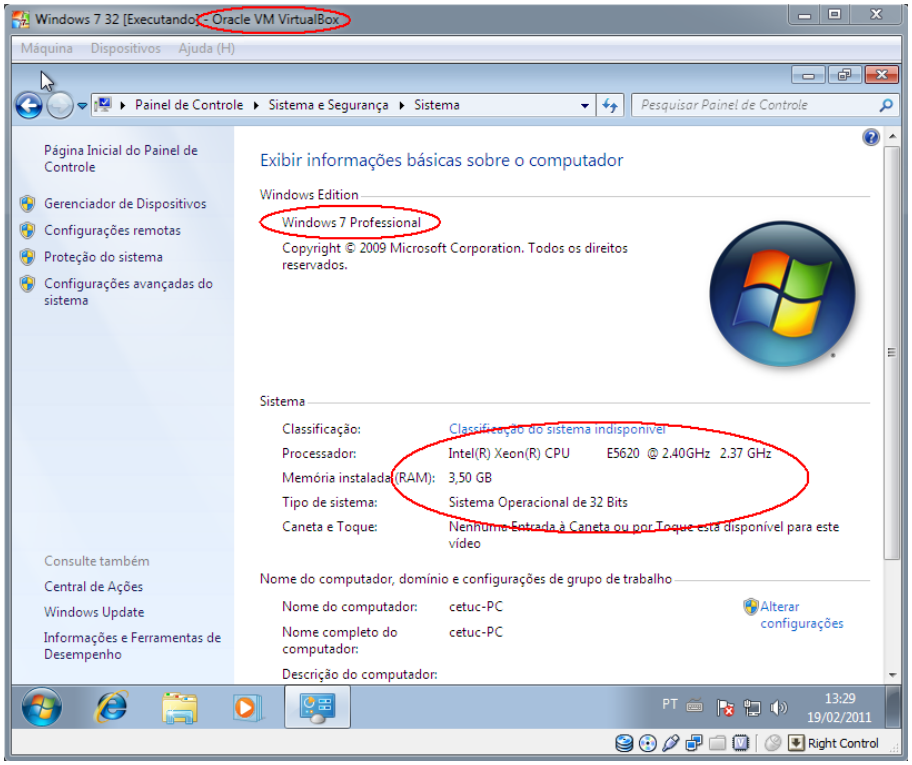

Figura 5.13 – Máquina virtual com o *Windows 7 Professional Edition* de 32 bits utilizado

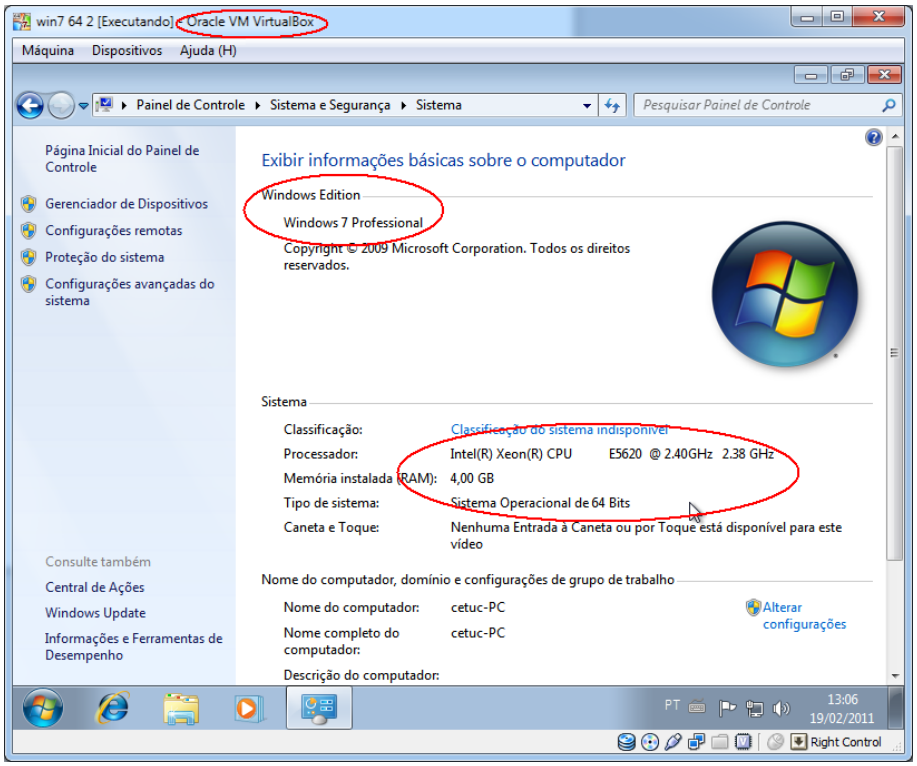

Figura 5.14 – Máquina virtual com o *Windows 7 Professional Edition* de 64 bits utilizado

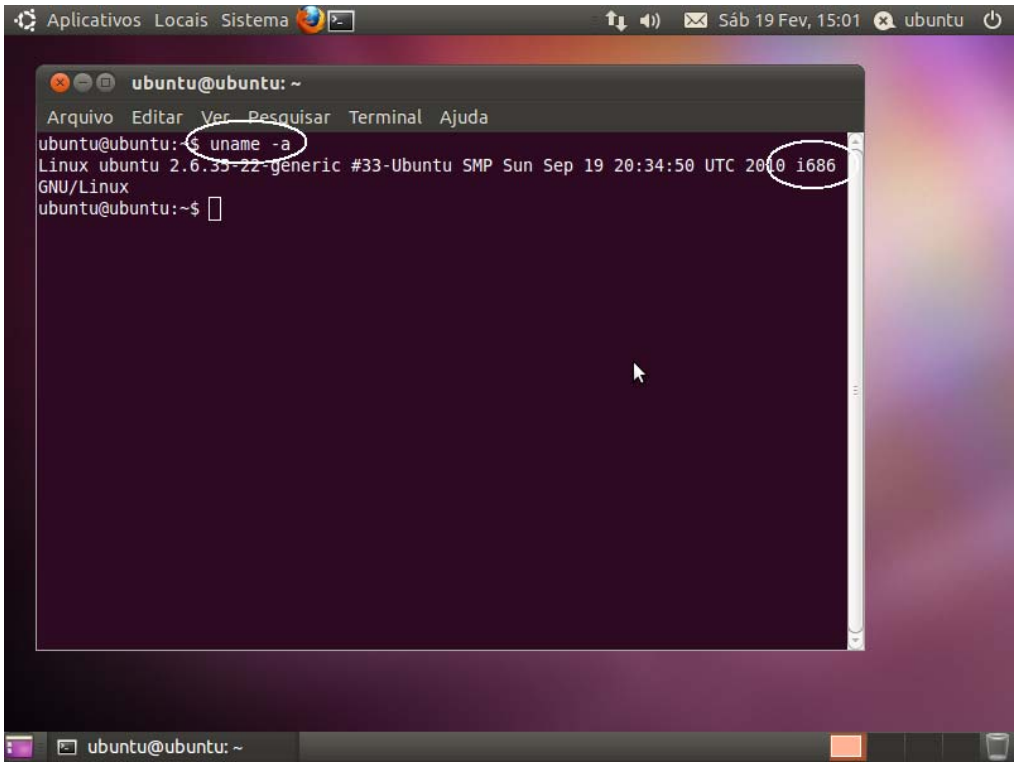

Figura 5.15 – Máquina virtual com o *Linux Ubuntu 10.04* de 32 bits utilizado

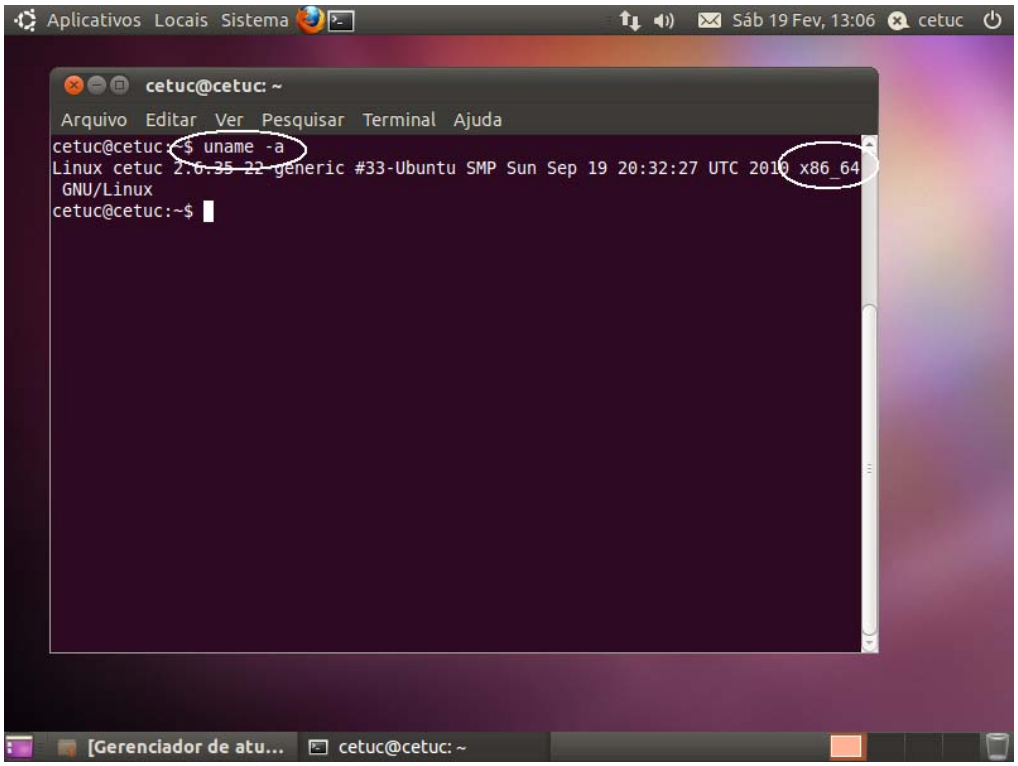

Figura 5.16 – Máquina virtual com o *Linux Ubuntu 10.04* de 64 bits utilizado

5.2
Resultados

A partir dos experimentos realizados, cuja organização foi explicada na seção anterior, os resultados foram coletados através de arquivos texto e a renderização gráfica feita para expressar visualmente o desempenho do *codec*.

Os primeiros gráficos que serão apresentados são os gráficos que mostram a variação da qualidade objetiva, através do PSNR, dos quadros Wyner-Ziv das sequências de vídeo.

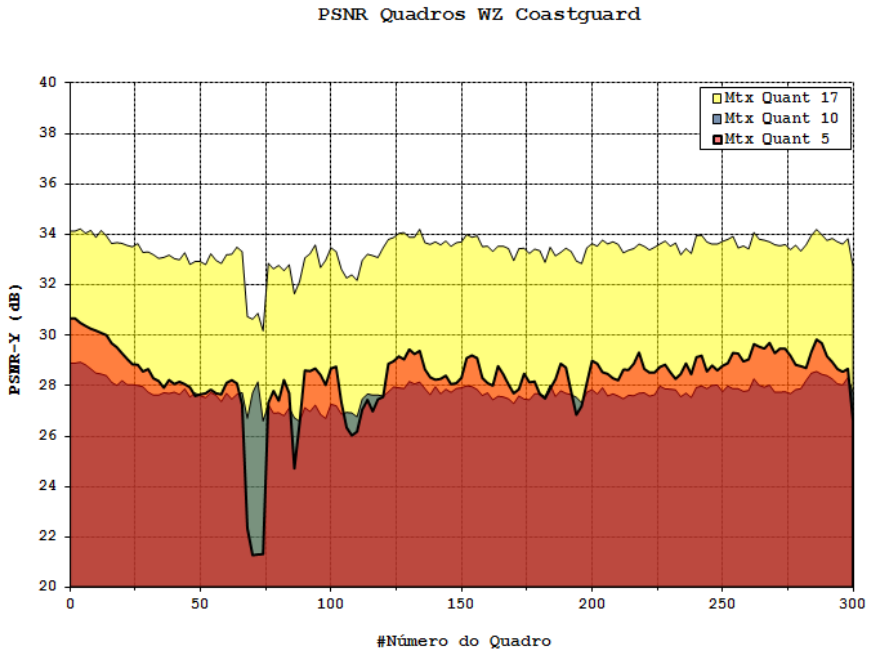

Figura 5.17 – PSNR WZ da sequência de vídeo *Coastguard*

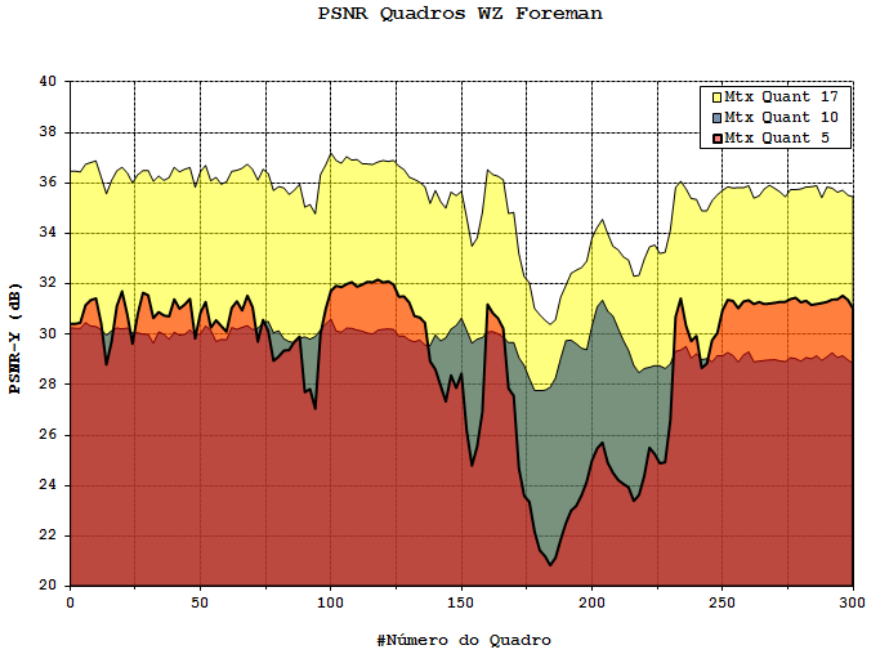

Figura 5.18 – PSNR WZ da sequência de vídeo *Foreman*

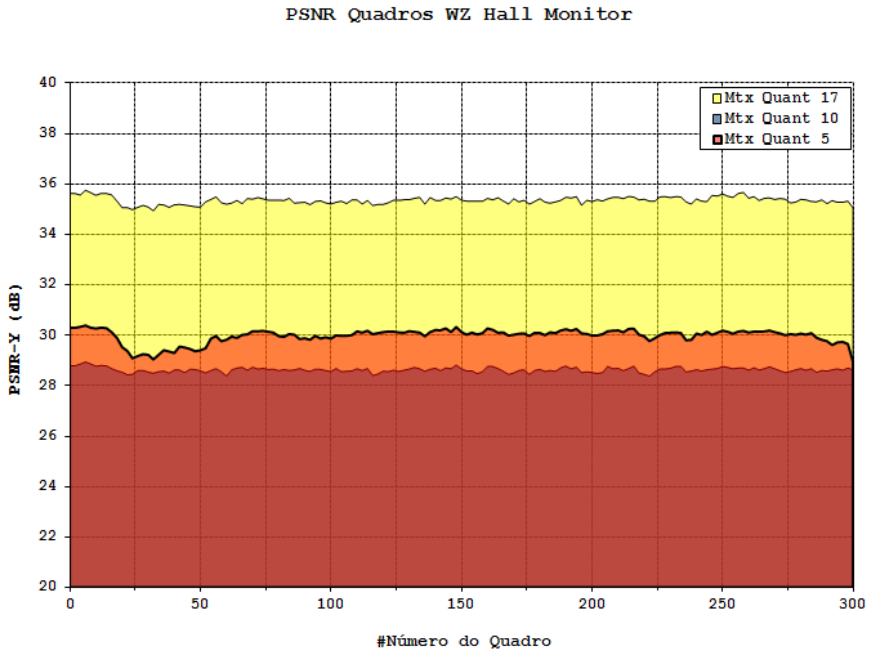

Figura 5.19 – PSNR WZ da sequência de vídeo *Hall Monitor*

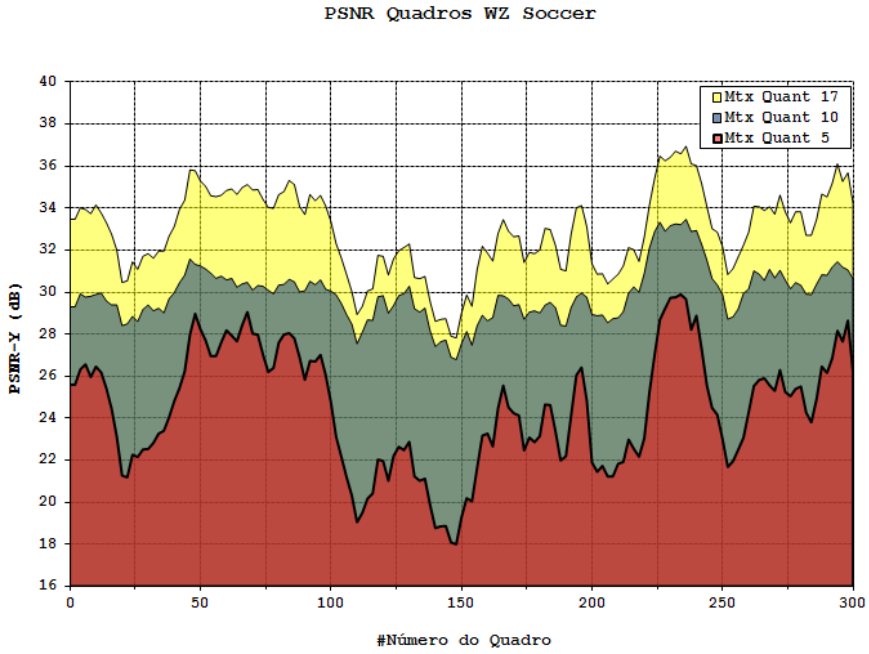

Figura 5.20 – PSNR WZ da sequência de vídeo *Soccer*

Como se pode ver nas figuras 5.17, 5.18, 5.19 e 5.20, o *codec* tem um desempenho muito bom quando utilizamos a Matriz de Quantização com o vetor de melhor qualidade, o décimo oitavo vetor, que é o vetor 17 (as matrizes são

numeradas de 0 a 17) em comparação com os outros dois apresentados. O custo dessa melhor qualidade, em termos de PSNR, é o aumento da taxa de transmissão, como veremos mais adiante.

Uma observação importante é que nos vetores intermediários, devido às suas distribuições e características, explicadas no capítulo 3 deste trabalho, o aumento da taxa e da quantidade de informação transmitida não necessariamente se concretiza numa melhora de qualidade, fazendo com que para alguns quadros, nas sequências de mais movimento, tenhamos um desempenho pior com o vetor 10 do que com o vetor 5, como ocorreu em alguns pontos do *Costguard* e *Foreman*.

Além da análise da qualidade dos quadros Wyner-Ziv, também é importante analisarmos a variação do PSNR entre os quadros chave, ou seja, codificados através do codificador intra-frame e os quadros Wyner-Ziv, uma vez que essa variação pode ser importante na reconstrução do quadro Wyner-Ziv, já que os quadros chave fornecem os dados para a geração da informação lateral. Essas variações serão apresentadas nas figuras 5.21, 5.22, 5.23 e 5.24.

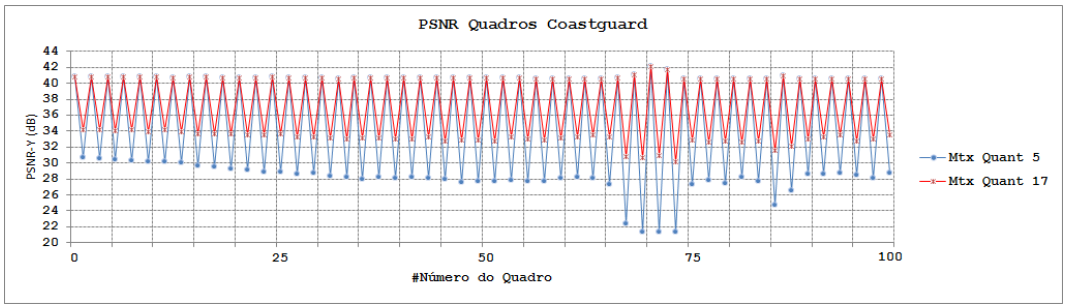

Figura 5.21 – PSNR dos 100 primeiros quadros da sequência de vídeo *Coastguard*

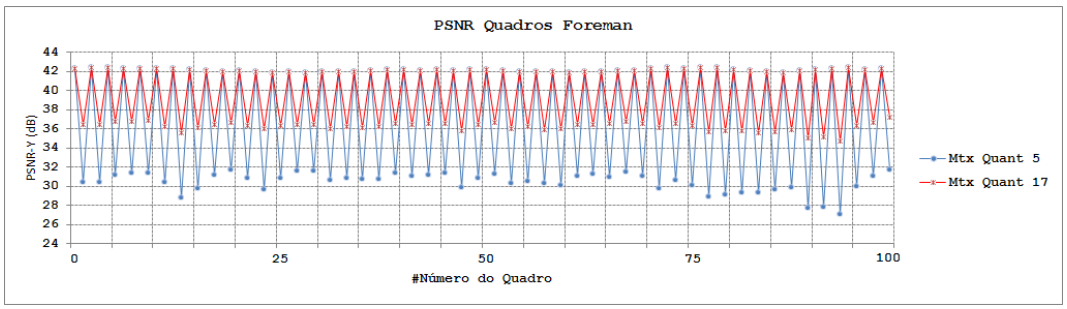

Figura 5.22 – PSNR dos 100 primeiros quadros da sequência de vídeo *Foreman*

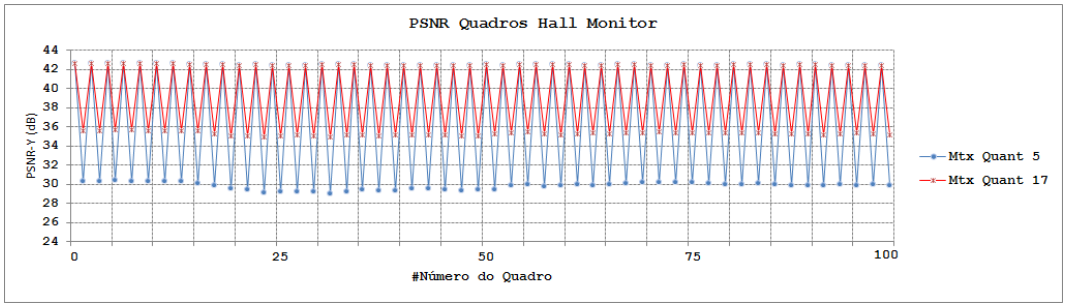

Figura 5.23 – PSNR dos 100 primeiros quadros da sequência de vídeo *Hall Monitor*

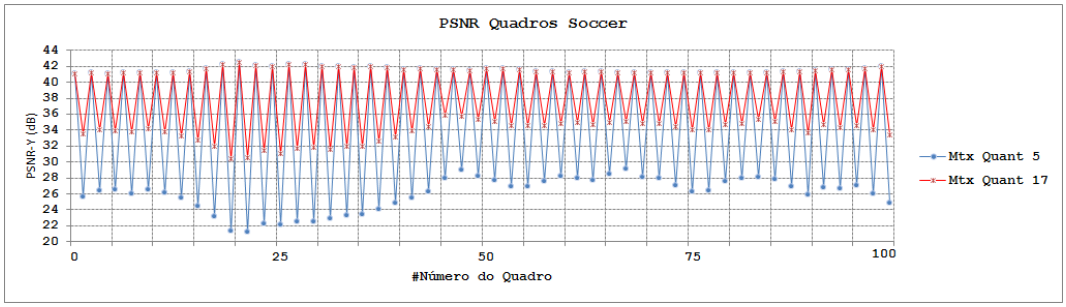

Figura 5.24 – PSNR dos 100 primeiros quadros da sequência de vídeo *Soccer*

A principal observação que pode ser feita em relação às figuras 5.21, 5.22, 5.23 e 5.24 é a variação bem maior entre os quadros chave e os quadros Wyner-Ziv por ocasião da utilização do vetor de quantização de menor qualidade, o Mtx Quant 5, o que é consequência da forma deste vetor em relação ao Mtx Quant 17. O primeiro tem uma quantidade grande de zeros, fazendo com que a taxa seja bem menor do que a taxa do último, o que se reflete também na qualidade final do quadro, como foi observado nos conjuntos das figuras 5.21, 5.22, 5.23 e 5.24, onde analisamos apenas o PSNR dos quadros WZ.

Outro detalhe importante, é que esta variação é maior nas sequências ou na parte destas que possuem de médio a alto movimento, fazendo com que possam ocorrer defeitos na reprodução do vídeo se utilizarmos uma matriz para baixas taxas nos momentos em que a cena ganha velocidade ou quando muda de cenário.

As figuras 5.25, 5.26, 5.27 e 5.28 apresentam os gráficos de taxa-distorção do Open DVC em relação aos codecs comerciais mais conhecidos no cotidiano, ou seja, H.264/AVC, nos perfis Intra e No Motion, além de compararmos também com o H.263+. Além disso, também compararemos o Open DVC com o codec do projeto DISCOVER [48].

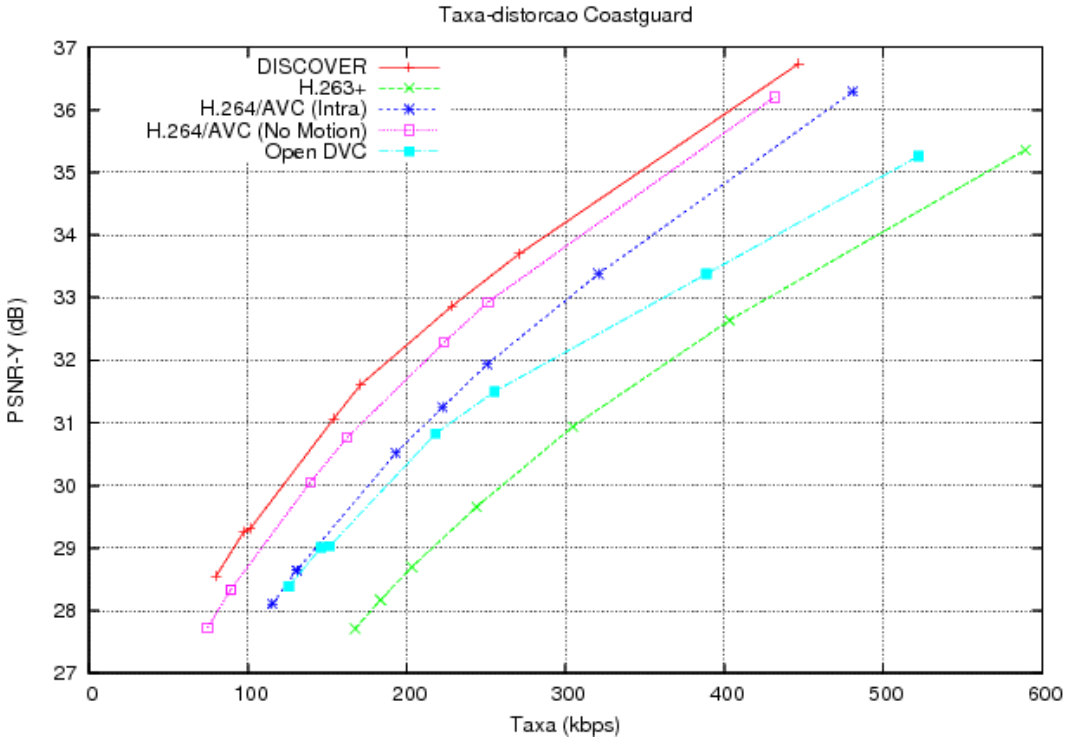

Figura 5.25 – Taxa-distorção da sequência de vídeo *Coastguard*

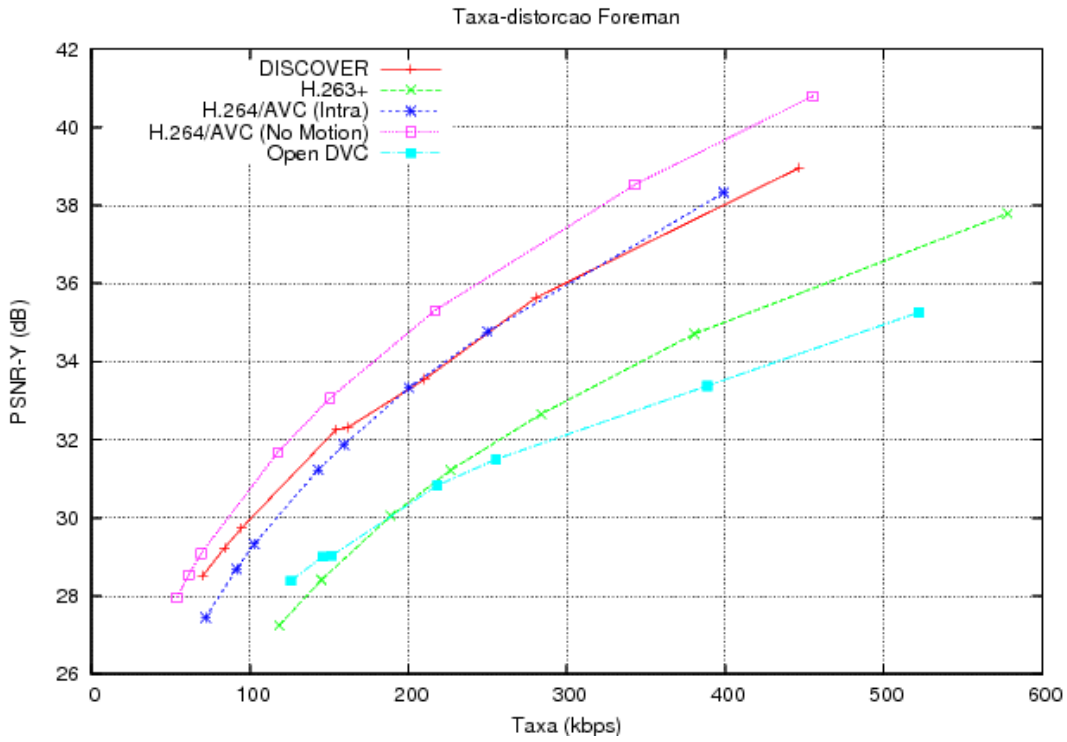

Figura 5.26 – Taxa-distorção da sequência de vídeo *Foreman*

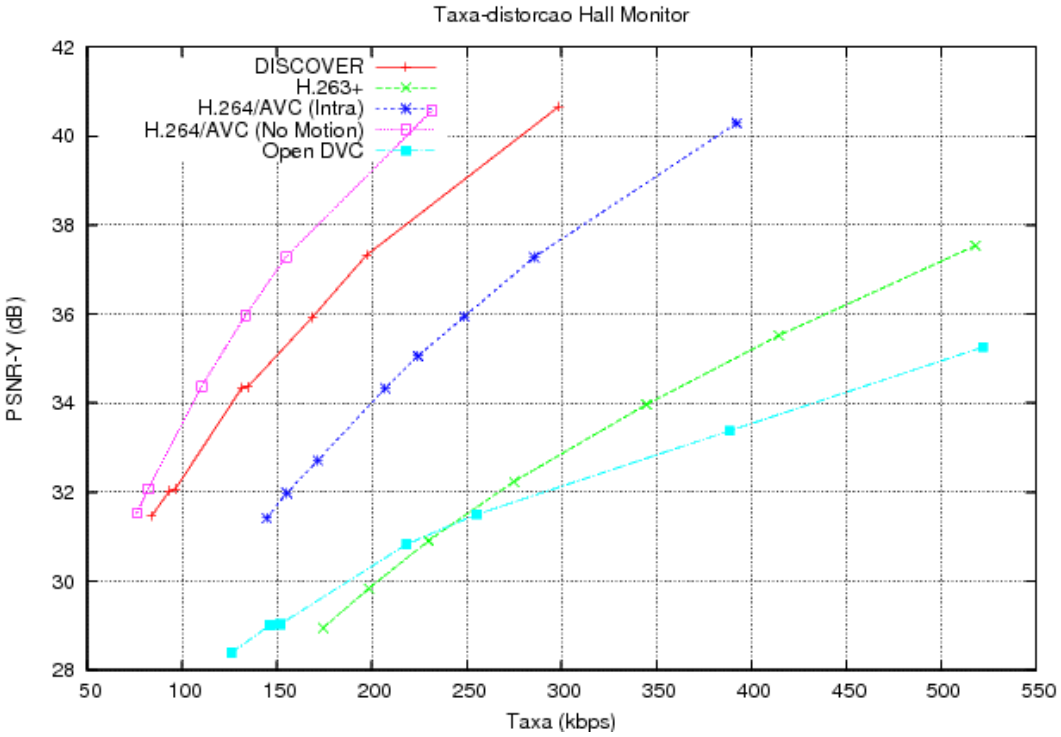

Figura 5.27 – Taxa-distorção da sequência de vídeo *Hall Monitor*

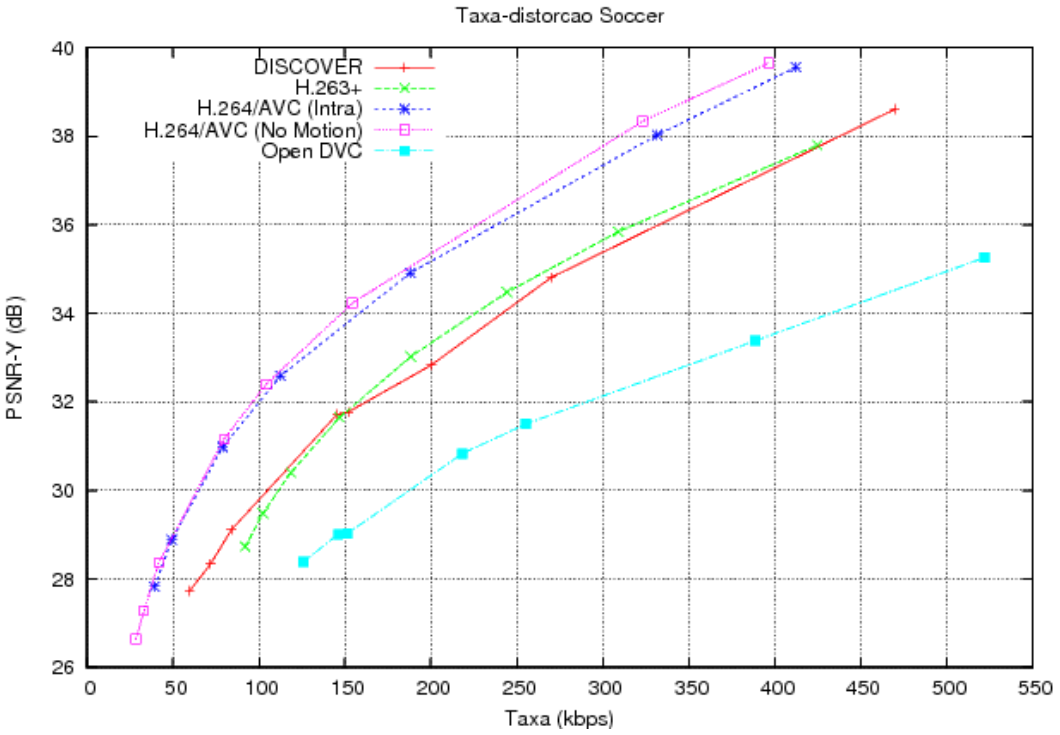

Figura 5.28 – Taxa-distorção da sequência de vídeo *Soccer*

Em relação à comparação da taxa-distorção do *codec* Open DVC com os demais *codecs*, pode ser percebido que o fato de ter-se optado por uma implementação de melhor desempenho computacional, reduzindo ao máximo a complexidade do codificador, fez com que ele perdesse em relação a alguns *codecs* em alguns cenários.

Maiores conclusões sobre os comparativos dos *codecs* serão feitas na próxima seção.

Por último, vamos apresentar alguns gráficos e ilustrações para analisarmos o esforço computacional do *codec* Open DVC.

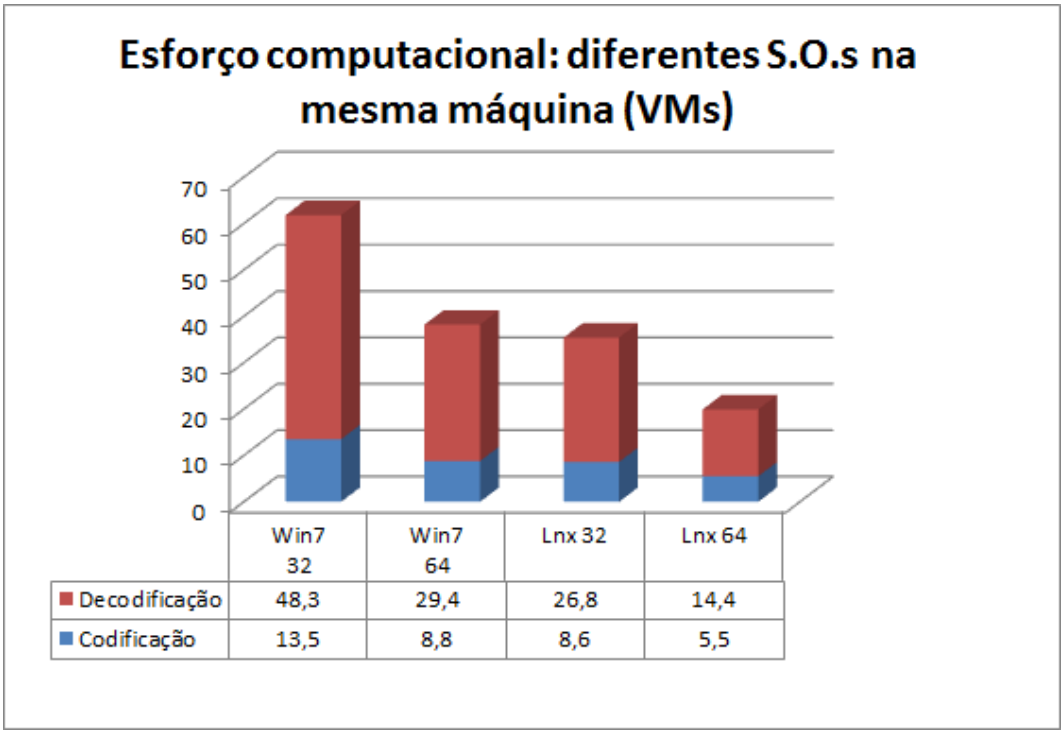

Figura 5.29 – Esforço Computacional: quatro S.O.s na mesma máquina através de máquinas virtuais (codificação e decodificação em segundos)

A figura 5.29 apresenta o resumo do esforço computacional de uma execução realizada de forma isolada, na máquina especificada na seção anterior e que foi utilizada para a execução dos experimentos desta dissertação, através das quatro máquinas virtuais que foram inicializadas na Virtual Box, como já fora explicado, instaladas no sistema operacional nativo da máquina.

Analisando o gráfico, podemos ver a superioridade da mesma operação para os mesmos parâmetros com os sistemas operacionais Linux sobre o sistema

operacional Windows, a saber, codificação e decodificação da sequência Foreman utilizando o Vetor de Qualidade 10 da Matriz de Quantização. Até mesmo a ferramenta rodando no Linux de 32 bits teve um desempenho ligeiramente superior ao funcionamento no Windows de 64 bits. Quando o Linux passou para a máquina de 64 bits a diferença foi muito maior.

Outro teste realizado foi de análise da execução simultânea de quatro execuções com os mesmos parâmetros. Foi feito um script para disparar as quatro linhas de execução em Linux 64 bits, já que o mesmo obteve o melhor desempenho.

Mesmo com quatro execuções simultâneas, o esforço de execução foi equilibrado de forma distribuída e equitativa, como pode ser visto tanto nas figuras 5.31 e 5.32, que mostram a distribuição do esforço entre os núcleos de processamento, quanto no gráfico da figura 5.34, que mostra que a medida foi praticamente a mesma nos quatro cores.

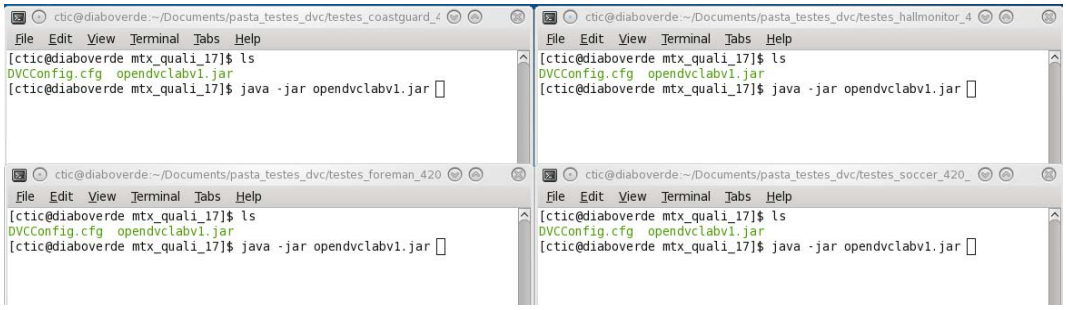

Figura 5.30 – *Script* disparando os quatro processos de codificação simultaneamente

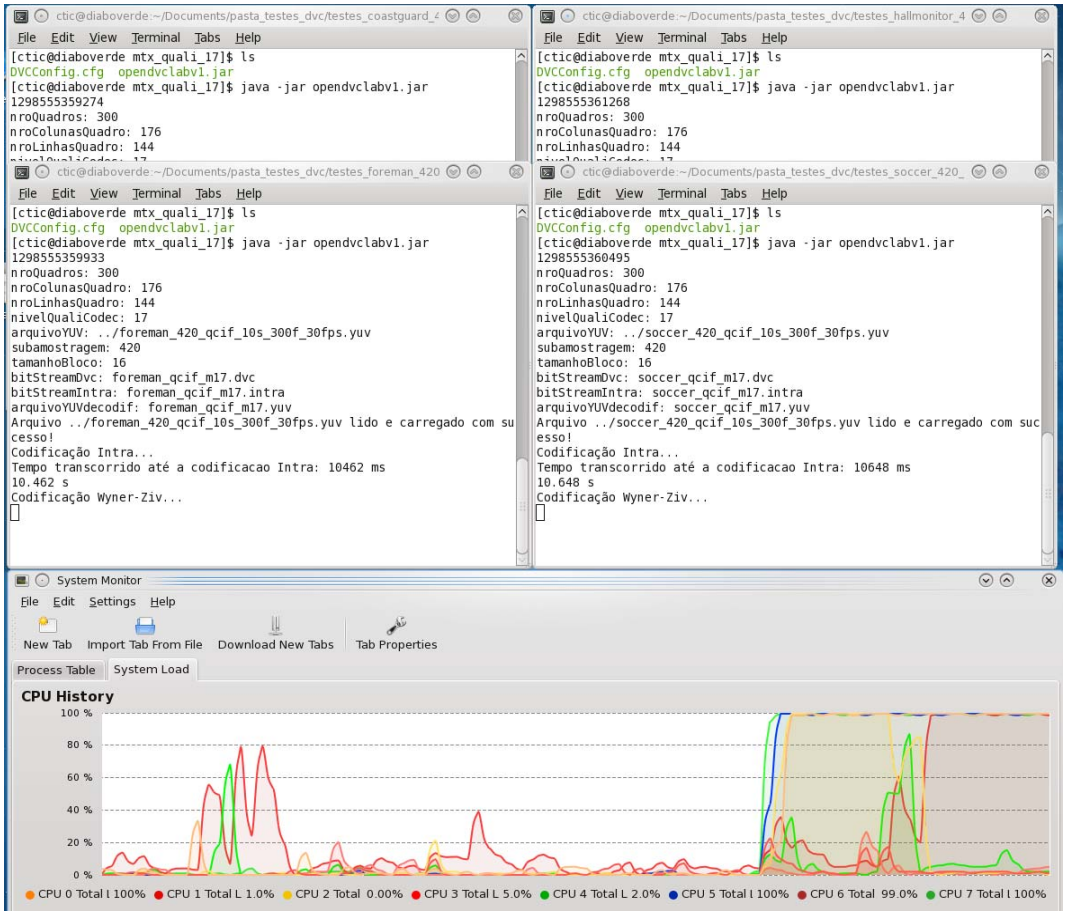

Figura 5.31 – Início da execução simultânea dos quatro processos de codificação: vide no gráfico rampa na carga de quatro dos núcleos

A figura 5.31 mostra o início da execução simultânea de quatro instâncias do *codec*, mostrando que para tal operação, o sistema operacional disponibilizou um núcleo para cada linha de execução, o que pode ser visto no início da rampa do gráfico, mostrando um ótimo controle de balanço de processamento por parte dos processadores modernos e dos sistemas operacionais de ponta.

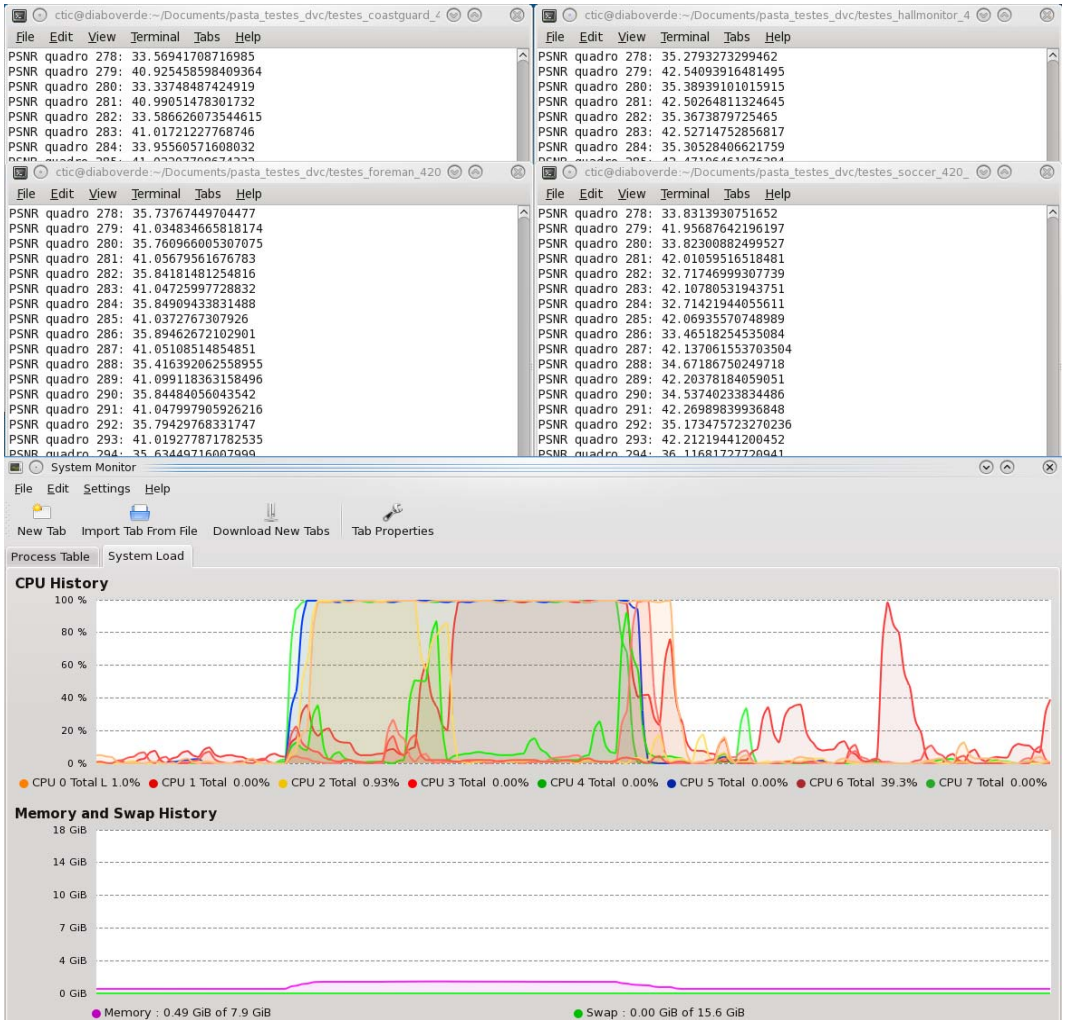

Figura 5.32 – Fim da execução simultânea dos quatro processos de codificação: vide no gráfico rampa e descida na carga de quatro dos núcleos

A figura 5.32 é uma continuidade da anterior, porém mostrando o fim da execução das *threads*, onde o computador retorna ao seu estado normal de carga de processamento, após passar por alto nível de estresse por ocasião da execução do *codec*.

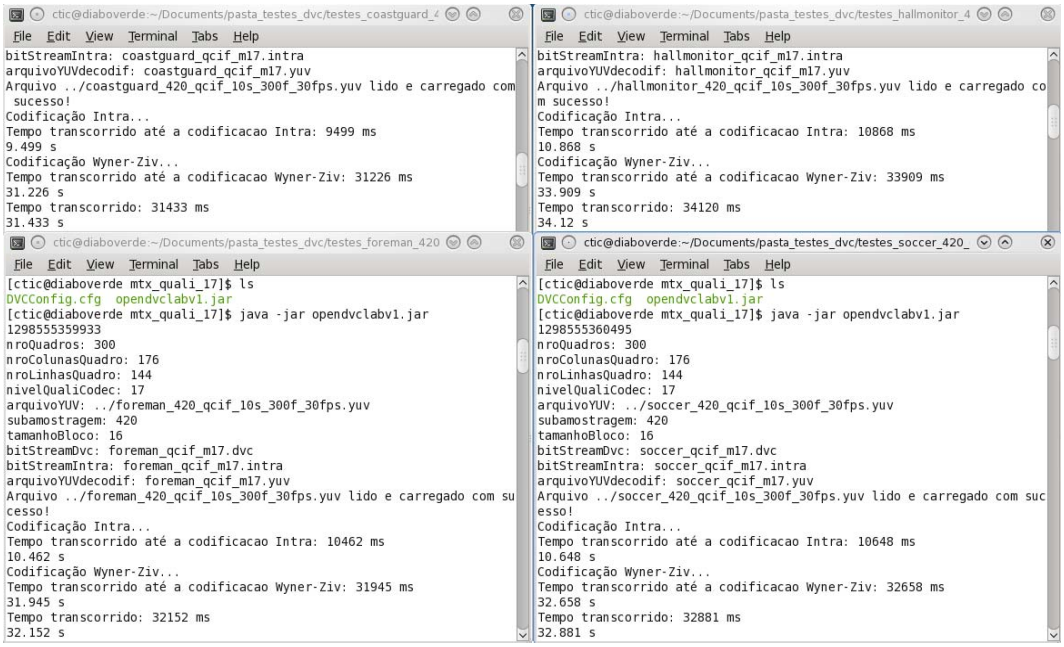

Figura 5.33 – Saída das quatro codificações, estimativas temporais

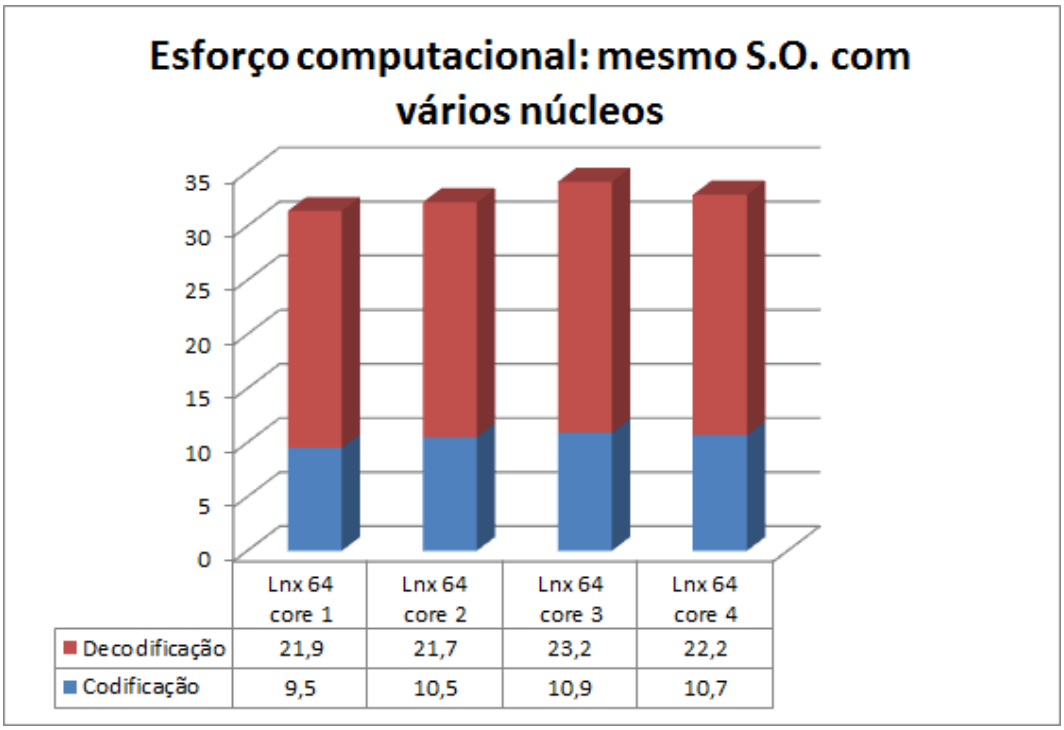

Figura 5.34 – Esforço Computacional: as quatro execuções no mesmo Sistema Operacional: equilíbrio de carga computacional (codificação e decodificação em segundos)

5.3
Conclusões

Diante dos resultados apresentados e das análises feitas, pode-se concluir que o *codec* DVC implementado pela ferramenta de simulação Open DVC, pode ser de grande importância, tanto para aprendizado e estudo da tecnologia DVC quanto como um protótipo para ser implementado o codec embarcado em um produto. Isso pode ser levado seriamente em consideração, se verificarmos que seu desempenho pode ser comparado aos dos *codecs* comerciais mais utilizados, sendo que, podem ser utilizadas técnicas para melhorar seu desempenho, através da otimização da informação lateral, o que será discutido nos próximos capítulos.

Além disso, o *codec* se mostrou extremamente leve e promissor, principalmente na codificação, o que é crítico para o cenário DVC, dando assim espaço para trabalharmos na decodificação, onde supomos ter abundância de recursos para sua melhoria.