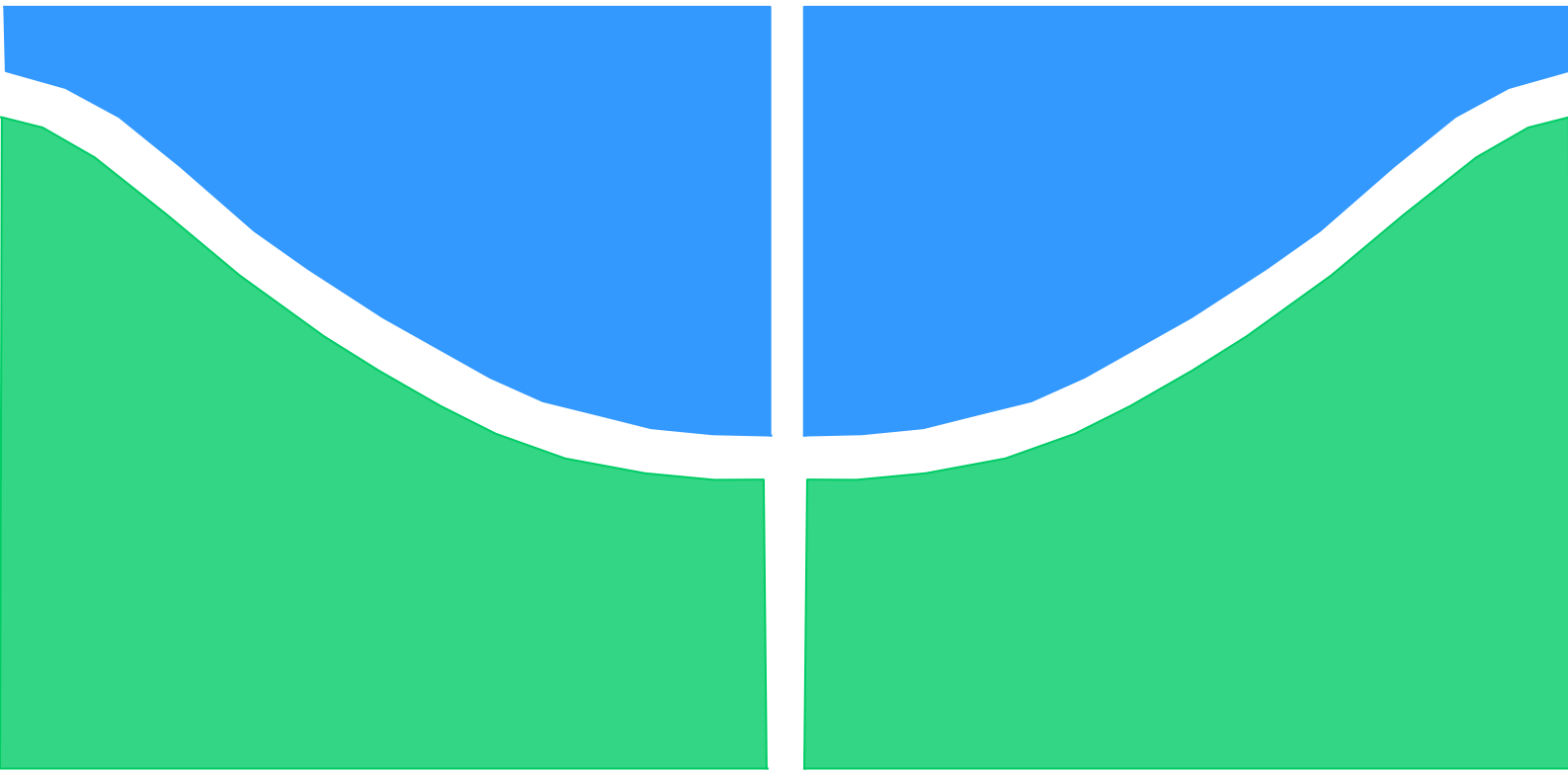

TRABALHO DE CONCLUSÃO DE CURSO

MONITORAMENTO DE VÍDEO POR MEIO DO COMPUTADOR RASPBERRY PI

Felipe Leite Scheidemantel

Brasília, julho de 2015

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

Faculdade de Tecnologia

TRABALHO DE CONCLUSÃO DE CURSO

MONITORAMENTO DE VÍDEO POR MEIO DO COMPUTADOR RASPBERRY PI

Felipe Leite Scheidemantel

Banca Examinadora

Prof. Ricardo Zelenovsky, UnB/ Departamento de Engenharia Elétrica	_____
Prof. Alexandre Ricardo Soares Romariz, UnB/ Departamento de Engenharia Elétrica	_____
Prof. José Edil Guimarães de Medeiros, UnB/ Departamento de Engenharia Elétrica	_____

Agradecimentos

Agradeço a todos que me apoiaram nesta longa jornada. Um especial abraço aos meus amigos, à minha família e à minha namorada. Non scholae sed vitae discimus.

Felipe Leite Scheidemantel

RESUMO

O presente trabalho apresenta a implementação de uma central de monitoração de vídeo via rede por meio do microcontrolador *Raspberry Pi.* Três diferentes protocolos de transmissão de vídeo são configurados: *streaming* direto, protocolo MJPEG*,* e protocolo RTSP. As vantagens e desvantagens de cada protocolo são comparadas. Também são demonstradas aplicações de interfaceamento de *hardware* do *Raspberry Pi*, com a elaboração de um simples circuito de comunicação via radiofrequência, utilizado para o acionamento ou desligamento da central, e de um circuito de controle de um *display* de cristal líquido, usado para monitorar o estado da central.

ABSTRACT

This work presents a video monitoring network system implementation with the Raspberry Pi micro-controller. Three different streaming protocols are configured: direct streaming, MJPEG protocol, and RTSP protocol. The protocols' performances are then compared. Hardware interfacing is also explored, with the project of a radio-frequency communication system, designed to activate or deactivate the video monitoring system, and the project of an LCD display control circuit, used to monitor the video system status.

SUMÁRIO

1 INTRODUÇÃO

1.1. ASPECTOS GERAIS

O advento da eletrônica digital foi responsável por diversas inovações tecnológicas que, apesar do profundo impacto que causaram nas atividades humanas, são atualmente lugar-comum na vida da maioria dos habitantes do planeta. Desde o surgimento do transistor e sua crucial influência na arquitetura de computadores digitais observa-se uma tendência de deslocamento do uso profissional ao uso pessoal. Os primeiros computadores digitais foram construídos ao longo das décadas de 1930 e 1940 e eram frequentemente usados para fins militares, especialmente para a quebra de códigos de mensagens criptografadas. No entanto, até mesmo nessas longínquas décadas, percebe-se uma grande preocupação, por parte dos engenheiros, com a possibilidade de programação de computadores. O *ethos* dessa época é bem refletido nas declarações do célebre matemático e pioneiro da computação Alan Turing, que apresentou a ideia da máquina universal como uma máquina capaz de efetuar qualquer operação matemática, desde que esta possa ser representada por um algoritmo. A ideia da máquina universal (hoje chamada de Máquina Universal de Turing ou *Universal Turing Machine*), ganhou força e levou ao surgimento do computador pessoal tão comum nos dias atuais.

O *Raspberry Pi*, o cerne deste texto, simboliza mais um passo nessa evolução computacional, e representa um esforço por parte de educadores para expandir o conhecimento de programação e eletrônica para pessoas de todas as idades. Como prova de sua utilidade e acessibilidade, este trabalho de graduação lidará com a implementação de um projeto bastante prático: uma central de monitoramento de vídeo.

1.2. MONITORAMENTO DE VÍDEO

Em se tratando de monitoramento de vídeo, existem duas alternativas tradicionais: o antigo circuito fechado de monitoração, também conhecido como CCTV, e a mais moderna câmera de vídeo via IP. O circuito fechado de monitoração apresenta diversas restrições de custo e praticidade, principalmente quando é implementado em ambientes domésticos, visto que exige uma estrutura própria de cabeamento, além de monitores exclusivos para observação das gravações. As câmeras de vídeo via IP possuem a vantagem de poder aproveitar a estrutura de dados existente na rede doméstica. O usuário pode, com a posse do endereço IP da câmera, acessar a gravação em qualquer computador e até mesmo controlar a câmera. Câmeras IP podem ser compradas pelo preço inicial de cerca de R$ 200,00.

Tendo em vista a crescente preocupação da população com segurança, somada à ascendente cultura DIY, ou *do it yourself* (faça você mesmo), onde internautas compartilham conhecimento por redes sociais para que qualquer pessoa possa realizar projetos outrora restritos a pessoas

especializadas, este trabalho tem como objetivo demonstrar o monitoramento de vídeo por meio de um pequeno microcontrolador de custo acessível.

1.3. METODOLOGIA

Neste texto serão abordados três diferentes protocolos de transmissão de vídeo via rede. Primeiramente será estudada a transmissão via conexão direta, que representa o protocolo mais simples. Em seguida, analisar-se-á o protocolo MJPEG, um protocolo de transmissão de vídeo amplamente utilizado por *webcams*. Finalmente, será estudado o protocolo RTSP, que é frequentemente adotado na transmissão de vídeo via rede. Os três protocolos serão comparados, e, com base em suas vantagens e desvantagens, terão indicadas diferentes aplicações práticas. Ademais, para que o leitor adquira uma boa compreensão dos fundamentos de transmissão de vídeo via rede e das limitações práticas acarretadas, serão estudados conceitos básicos de vídeo e a técnica de compressão MPEG, usada pelo microcontrolador *Raspberry Pi*.

Para que possa ser implementada uma central de monitoramento de vídeo de alto grau de praticidade, serão estudadas algumas opções de interfaceamento do *Raspberry Pi*. Estudar-se-á um método de acionamento remoto da central de monitoramento, por meio de módulos de comunicação via radiofrequência. Também será discutida a programação de um *display* de cristal líquido, conectado ao *Raspberry Pi* para que o usuário possa acompanhar o funcionamento da central de monitoramento com maior facilidade.

2. O RASPBERRY PI

2.1 PROPÓSITO E POSSIBILIDADES

O *Raspberry Pi* é um computador de baixo custo com as dimensões físicas de um cartão de crédito. Assim como os computadores *desktop* modernos, possui saída de vídeo, saída de áudio, interface *Universal Serial Bus* (**USB**) e interface *Ethernet*. O *Raspberry Pi* será neste texto doravante referido como Raspberry**,** Raspberry Pi e Rpi**.**

É importante se perguntar por que motivos esse pequeno computador, de especificações técnicas inferiores quando comparadas às especificações de *desktops* modernos, vem causando grande agitação no ambiente de engenharia eletrônica em todas as partes do mundo. Para a obtenção dessa resposta, deve-se ter primeiramente o conhecimento do propósito para o qual o Raspberry foi concebido. A Figura 1 ilustra o *Raspberry Pi* em sua versão mais recente, o modelo B+.

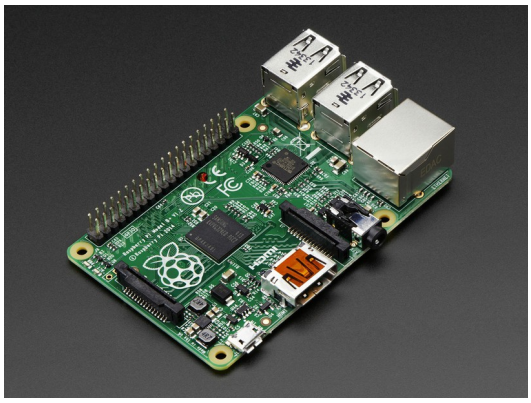

Figura 1: Raspberry Pi Modelo B+ [1].

O Raspberry Pi foi idealizado no ano de 2006, nos laboratórios de ciência da computação da Universidade de Cambridge, localizada no Reino Unido. Os professores Eben Upton, Rob Mullins, Jack Lang, e Alan Mycroft deparavam-se com um cenário acadêmico desanimador, pois as notas de programação dos alunos de ciência da computação decaíam gradativamente, ano a ano. Mais preocupante que a progressiva queda de notas, havia uma mudança de paradigma no ambiente universitário. Na década de 1990, os alunos de ciência da computação ingressavam na universidade com amplo conhecimento em programação, conhecimento este decorrente da imensa experiência que adquiriram por *hobby*. No entanto, os alunos ingressantes após a década de 2000, embora entusiasmados com o curso, demonstravam um conhecimento prático bastante limitado, ou até inexistente. Independentemente das causas que levaram à mudança de mentalidade dos alunos, era preciso reverter o fluxo, e reacender o interesse pela programação por conta própria. A reação materializou-se no Raspberry Pi, mas sua influência rapidamente excedeu o ambiente da ciência da computação e da programação, criando fortes raízes no mundo da eletrônica. Pouco depois do lançamento, surgia uma enxurrada de diferentes projetos e aplicações do pequeno computador. Alguns exemplos da ampla gama de experimentos realizados com o Raspberry Pi são: controlador de aparelho micro-ondas, aparelhos de *video-game*, telefones celulares controlados por tela LCD sensível ao toque, rádio definido por software (*software-defined radio*), servidores de dados, e, até mesmo um trabalho de pós-graduação em processamento de sinais [2].

A ideia fundamental por trás do Raspberry é utilizar um processador **ARM** para que o usuário tenha o poder de utilizar o **Rpi** como microcontrolador. Entretanto, o **Rpi** não poderia ser definido como um computador convencional de baixo custo se não possuísse um sistema operacional. Como tal, possui o sistema operacional *Raspbian,* uma variação do sistema operacional Debian, de arquitetura Linux. Tal escolha não se deu apenas pelo fato de sistemas operacionais Linux possuírem código aberto, ou *open-source*. Sistemas operacionais Linux são excelentes plataformas de ensino de programação, como será visto adiante. O interesse de engenheiros eletrônicos no produto, no entanto, depende da existência da interface do Raspberry Pi com o mundo analógico, ou, em outras palavras da *General Purpose Interface* (**GPI**).

O Raspberry Pi possui diversos pinos denominados *GPIO pins*, ou *General Purpose Input/Output pins*, cuja função é servir de entrada ou saída de sinais. Esses pinos formam a interface GPI citada anteriormente. Essa interface, somada à mistura de um poderoso processador ARM, uma **GPU** (*Graphics Processor Unit*), um sistema operacional, e uma conexão com a internet, propicia ao usuário um leque praticamente ilimitado de possibilidades de projetos eletrônicos. Destarte, o antigo conceito de *Internet of Things,* ou internet das coisas, onde diversos equipamentos de *hardware* estão conectados à Internet e têm a capacidade de conversar entre si, ganhou novo fôlego com o advento de microcontroladores como o Raspberry Pi e o Arduino. Neste texto será demonstrada a implementação de monitoramento de vídeo pelo Raspberry e suas possíveis interfaces de controle. O exemplo demonstrado neste trabalho trata-se apenas de um nicho das possíveis aplicações do pequeno computador, mas é um nicho de bastante interesse, tendo em vista a crescente preocupação da população com segurança, e o desejo desta em poder monitorar ambientes de interesse. O autor tem a expectativa, portanto, de suscitar futuro desenvolvimento de produtos de baixo custo que atendam a essa importante demanda.

Componente de fundamental importância para este projeto, a câmera projetada exclusivamente para o Raspberry Pi é apenas um exemplo de periféricos desenvolvidos para o pequeno computador. O Raspberry Pi também conta com o módulo *Compute*, mostrado na Figura 2, que consiste um kit de desenvolvimento de protótipos de alto poder de processamento com 120 pinos de interface GPI [3]. Existem também inúmeros periféricos desenvolvidos por terceiros, o que expande ainda mais as possibilidades de uso do Raspberry.

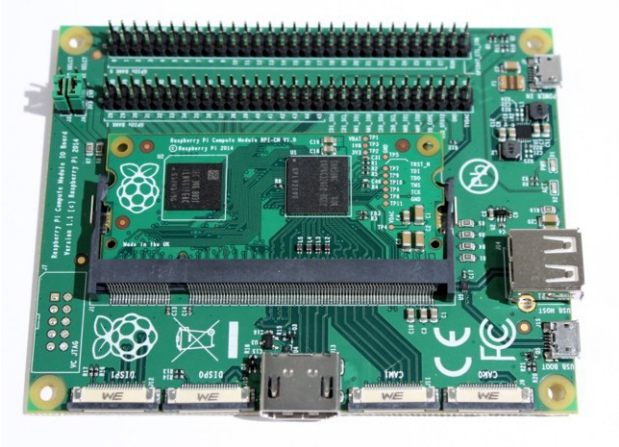

Figura 2: O módulo Compute usado para desenvolvimento de protótipos [3].

2.2 O HARDWARE

O Raspberry Pi conta com processador ARM 1176 com relógio padrão de 700 MHz. A unidade processadora de gráficos, ou **GPU** (*Graphics Processing Unit*), é capaz de processar 1 milhão

de píxels por segundo (**1 Gpixel/s**), tendo poder de processamento gráfico semelhante ao do console Xbox. O Raspberry conta, ademais, com 26 pinos de **GPIO**, que incluem uma UART (*Universal Asynchronous Receiver Transmitter*), um barramento **I2C**, um barramento **SPI** com duas funções de *chip select*, saída de áudio **I2S**, e, é claro, as tensões de referência de 3,3 Volts, 5 Volts, e 0 Volts (terra). A *Raspberry Pi Foundation* divulga livremente o esquemático do Raspberry em seu website. As descrições de hardware do Raspberry são expandidas nas próximas seções.

2.2.1 O PROCESSADOR ARM

A arquitetura **ARM** de processadores, assim nomeada em função de sua empresa criadora, a britânica *Arm Holdings,* revolucionou o mercado de sistemas embarcados. A premissa fundamental dos processadores pertencentes a essa família é o conjunto de instruções **RISC,** ou *Reduced Instructions Set Computing,* cujo objetivo é reduzir drasticamente o número de instruções necessárias para que o processador realize determinada ação. Processadores ARM, destarte, requerem substancialmente menos transistores que os processadores de arquitetura x86, os mais utilizados no mercado de *personal computers* ou Pcs [4]. Pode-se portanto abrir um desvio ao presente raciocínio para a observação de uma significativa diferença entre o Raspberry Pi e computadores convencionais: *o Raspberry Pi, apesar de ter sido desenvolvido com o propósito de realizar as funções de computadores convencionais, foi construído com componentes de hardware propícios para sistemas embarcados*. Este ponto foi de grande importância para a imensa gama de aplicações encontradas para o Raspberry no projeto de sistemas embarcados.

A grande vantagem da redução do número de transistores usados na construção do processador é a consequente diminuição de consumo de energia, dissipação de calor, e custo de fabricação. Estes três fatores são cruciais na construção de sistemas leves, portáteis, e alimentados a bateria. Atualmente processadores ARM possuem grande importância no mercado de *smartphones, laptops, tablets*, televisão digital, servidores, e diversos sistemas embarcados. Processadores ARM são os processadores mais fabricados no mundo, com 50 bilhões de unidades produzidas em 2014 [4]. Existem processadores ARM com endereços e aritmética de 32 e 64 bits. A Figura 3 esquematiza as diversas famílias de processadores ARM.

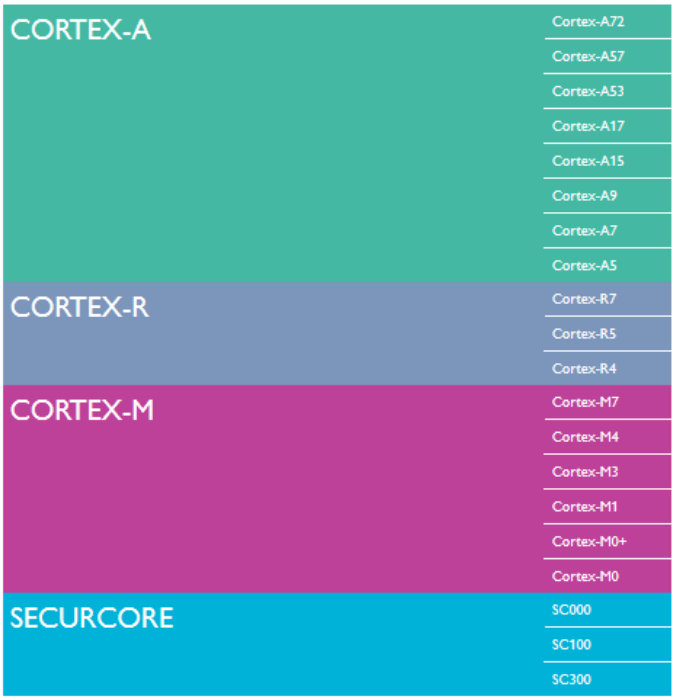

Figura 3: Famílias de processadores ARM [4].

A família de processadores CORTEX-A é amplamente utilizada no projeto de *smartphones, netbooks, eReaders,* e televisores digitais. A família CORTEX-R encontra aplicações no projeto de sistemas de freio automotivo, controladores de armazenamento em massa, e controladores de rede. A família CORTEX-M é utilizada para a implementação de *airbags*, sensores inteligentes, e microcontroladores. Aplicações de alto nível de segurança e processadores de FPGAs (*Field-programmable Gate Array*) são projetados com processadores da famíla SECURCORE.

O processador ARM1176JZF-S**,** usado no Rasberry Pi, pertence à família CORTEX-A e é bastante interessante para uso em sistemas embarcados. A frequência de operação pode ser aumentada para até 1 GHz sem a necessidade do uso de dissipador de calor. Seu consumo de energia é de aproximadamente 0,625mW/GHz. Como pode ser observado na Figura 4, o processador conta com arquitetura de núcleo único (*single core processor*), tendo interfaces para *debug*, controlador de coprocessador, instruções, dados, DMA (*Direct Memory Acess:* permite a componentes de hardware que acessem a memória do sistema sem intermédio do processador, gerando agilidade), e periféricos.

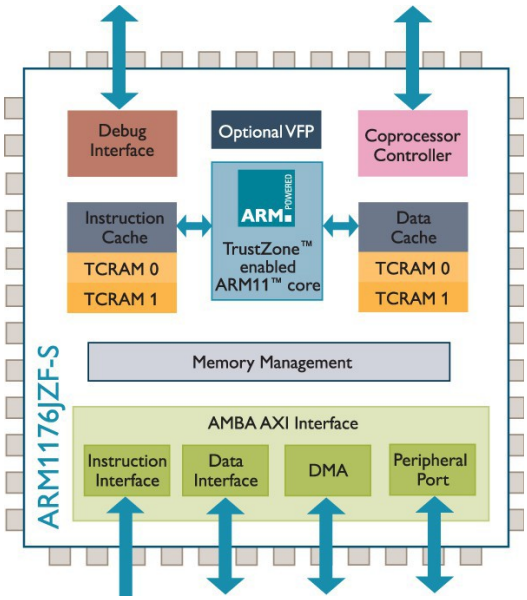

Figura 4: Diagrama funcional do processador ARM1176JZF-S [4].

2.2.2 A CÂMERA

A câmera projetada exclusivamente para o Raspberry Pi custa por volta de US$ 25,00 e pesa apenas 3 gramas. Possui as funções básicas de fotografia e gravação de vídeo, mas conta também com opções avançadas de vídeo, como: gravação em câmera lenta, *time-lapse* e outras formas de gravação. Suporta a gravação de vídeo em alta definição e é capaz de capturar fotos de até 5 *megapixels* [5]. Existem diversas bibliotecas na linguagem de programação *Python* e scripts de *shell* para Linux usadas para controle da câmera. Como pode ser observado na Figura 5, o módulo de câmera possui um conector reservado na própria placa de circuito impresso do *Raspberry Pi*.

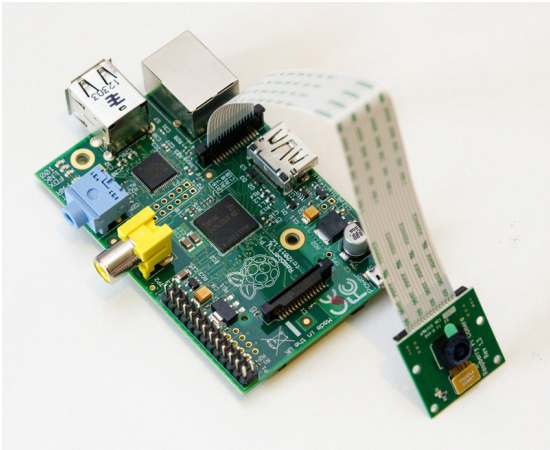

Figura 5: Raspberry Pi com módulo decâmera instalado [6].

2.3 INTERFACEAMENTO

Um dos atributos mais interessantes do Raspberry Pi é, sem dúvida, sua possibilidade de interfaceamento. Conforme dito acima, existem inúmeros protocolos de comunicação com os quais o Raspberry Pi pode se adaptar. Todo o interfaceamento de hardware do Rasperry Pi é realizado por meio dos pinos de GPI, mostrados na Figura 6.

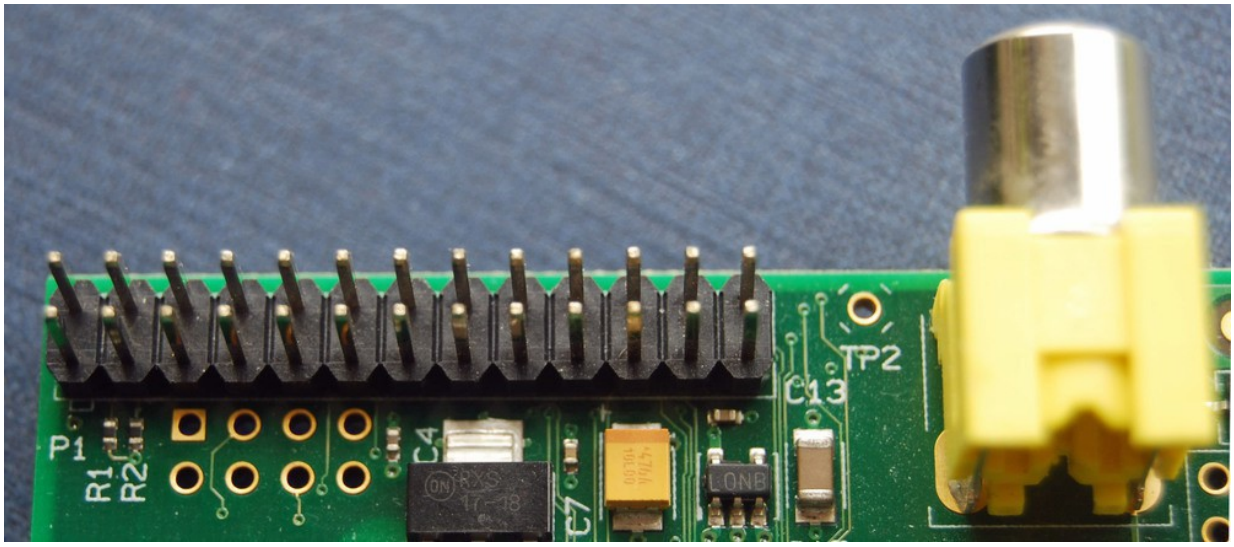

Figura 6: Nesta fotografia são observados os pinos de GPI do Raspberry Pi [7].

Dos 26 pinos vistos acima, 17 pinos servem de interface GPI, enquanto os pinos remanescentes são pinos de alimentação ou terra. O Raspberry fornece duas tensões de alimentação: 5 Volts e 3,3 Volts. No entanto, o nível lógico dos pinos de GPI varia de 0 a 3,3 Volts. É importante que o usuário atente-se a este fato para não danificar a eletrônica do sistema ao conectar pinos à tensões inadequadas. A Figura 7 especifica a função de cada pino da interface GPI.

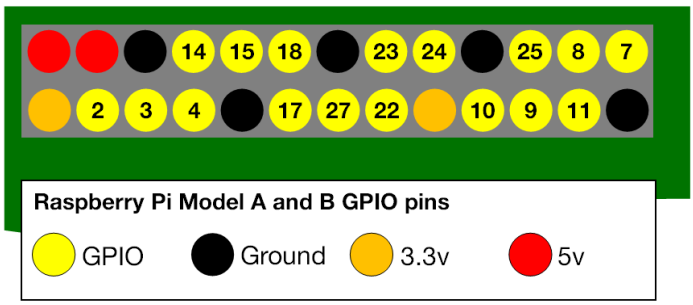

Figura 7: Pinos de GPIO e pinos de alimentação [7].

2.3.1 A INTERFACE GPI

Os pinos de GPIO do Raspberry Pi podem servir para diversas funções. O exemplo mais simples que vem à tona é o comando de um LED via um script escrito em *Python*. Para tanto, basta conectar os terminais do LED conforme ilustra o diagrama ilustrado na Figura 8. O LED, então, pode ser facilmente controlado via um código programado pelo usuário. O fato de o Raspberry Pi ter as funções de um computador agora se torna vantajoso. A depender do código escrito pelo usuário, o controle do LED pode ser feito até mesmo via Internet, para sinalizar eventos como a chegada de uma mensagem eletrônica, por exemplo.

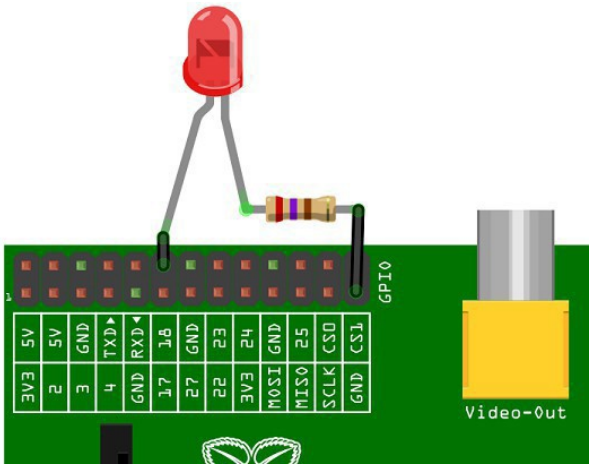

Figura 8: LED controlado via pino GPI [7].

Para que se possa ter um conhecimento mais aprofundado dos pinos de GPIO, é necessário primeiro saber qual é a função de cada um; embora qualquer pino GPIO possa servir para leitura e escrita de *bits*, existem pinos especificamente projetados para determinados protocolos de comunicação, como pode ser visto na Figura 9. Observa-se que os pinos 3 e 5 denominam-se **SDA** e **SCL**, e são utilizados no protocolo de comunicação serial **I2C**. Os pinos 11, 19, 21, 23, 24 e 25, respectivamente denominados **SPI1 CE1, MOSI, MISO, SCLK, SPI0 CE0,** e **SPI0 CE1,** são usados para comunicação via **SPI** (*Serial Peripheral Interface*), um protocolo de comunicação serial via um barramento de 3 fios. Os pinos 8 e 10, respectivamente **TXD** e **RXD**, são pinos de comunicação via serial. O pino 4, chamado **GPCLK0**, é um pino de *General Purpose Clock Signal*, isto é, possui a função de gerar sinais de *clock* para temporização. Os pinos 12 e 13, rotulados **PCM_C** e **PCM_D,** são pinos geradores de modulação em código de pulsos (*pulse code modulation*) e podem ser utilizados em aplicações de áudio e de temporização precisa.

É interessante notar, também, que os pinos possuem dois padrões de numeração, sendo um deles de acordo com a organização espacial dos pinos e outro de acordo com o circuito integrado controlador da interface GPI. A numeração antecedida pelo prefixo BCM representa a numeração

válida para o circuito controlador e, portanto, é a numeração que deve ser levada em consideração no momento da programação de códigos ou *scripts* que utilizem a interface GPI.

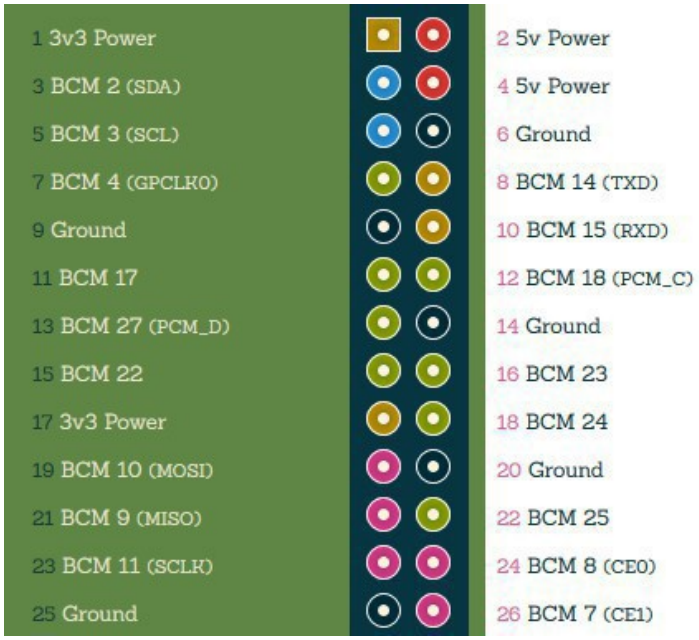

Figura 9: Numeração dos pinos e suas funções específicas [8].

3. CONCEITOS BÁSICOS DE VÍDEO

Embora o vídeo analógico seja considerado um anacronismo na era da televisão digital, sua grande importância histórica traduziu-se em sua significativa influência no vídeo digital. Por mais que apresente diferenças significativas em seus princípios de funcionamento, o vídeo digital possui inúmeros aspectos quantitativos e qualitativos comuns ao vídeo analógio. Conseguintemente, é interessante abrir um breve parêntesis para dissertar sobre aspectos fundamentais de vídeo analógico, visto que a compreensão destes será de grande auxílio no estudo de vídeo digital.

A primeira observação relevante a ser feita é a seguinte: *o vídeo sempre começa em um processo analógico e termina em outro processo analógico.* Por mais que o vídeo digital tenha avançado e aberto uma grande distância ao vídeo analógico, seu objetivo permaneceu inalterado: transformar a luz, uma onda analógica, em tensão, outra grandeza analógica, no começo do processo, e, ao final deste processo, transformar tensões em luz. Este é um dos principais motivos pelos quais vários parâmetros de análise de vídeo analógico mantiveram seu valor intacto na análise de vídeo digital [9]. De certa maneira, pode-se considerar a transmissão de vídeo analógico como um sinal contínuo de tensão contendo intensidade de luz e cor. A conveniência do vídeo digital reside no fato de uma tensão analógica ser facilmente convertida em valores digitais, e a partir daí ser manipulada com alto poder de processamento sem perda de informações [10]. É curioso notar que sinais de vídeo

digital existem no mundo da televisão há muito mais tempo que a própria televisão digital, surgindo primeiramente em geradores de funções e geradores de caracteres.

Os primeiros sinais de vídeo digital surgiram como uma mera representação de sinais analógicos de vídeo NTSC e PAL, padrões usados na transmissão de TV analógica nos Estados Unidos e na Europa. Desde os primórdios da produção de vídeo, engenheiros sempre perceberam a utilidade de se manter os três canais de cor, azul, verde, e vermelho, ou **RGB** (*Red, Green, Blue*) separados, e esta prática permanece inalterada nos dias atuais. O sinal capturado pela câmera sempre é dividido nestas três componentes cromáticas, e essa divisão só é alterada no estágio de processamento digital.

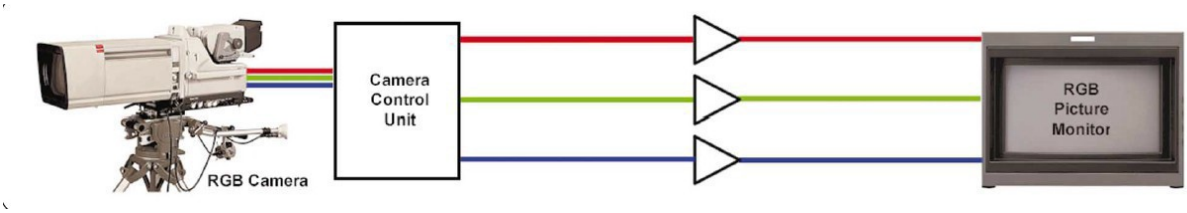

Figura 10: O sinal de vídeo capturado pela câmera sempre é dividido em suas componentes vermelha, verde e azul [10].

A Figura 10 ilustra a configuração mais simples de transmissão de vídeo. Sensores na câmera capturam os três sinais de vídeo monocromáticos e adicionam informações de sincronia, para que o monitor identifique o canto superior esquerdo da imagem transmitida pela câmera. Reitera-se aqui que o processo acima é independente do tipo de sinal de vídeo: digital ou analógico. Embora simples, a configuração acima causa diversas complicações práticas: diferenças nos comprimentos de cabo podem produzir erros de temporização, divergências entre ganhos dos diferentes canais monocromáticos podem produzir alterações na imagem, e tensões de *offset* DC produzem ruído indesejado no monitor [10]. Claramente, torna-se conveniente a junção dos três canais de cor para que possam ser processados simultaneamente. Isso pode ser obtido por meio de codificadores e decodificadores, como demonstra a figura abaixo.

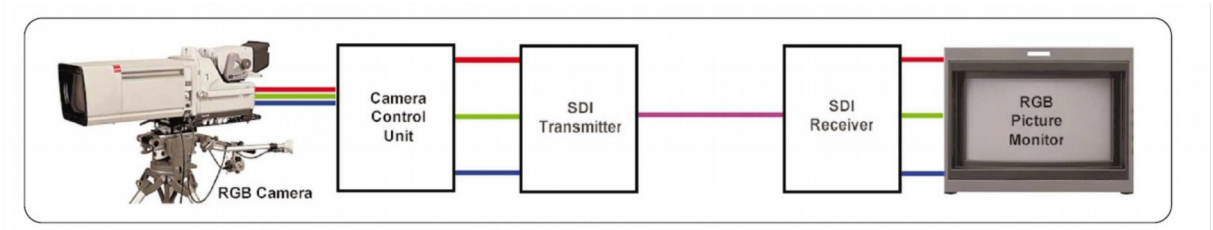

Figura 11: A transmissão digital facilita o processamento de canais cromáticos independentes [10].

Observa-se na Figura 11 que o transmissor e o receptor adequam-se ao padrão **SDI**, ou *Serial Digital Interface*. O padrão SDI é amplamente utilizado na indústria de vídeo digital, pois é capaz de, por meio de um único cabo coaxial, transmitir todas as informações de vídeo necessárias para a reprodução em monitores. Um transmissor SDI de vídeo **SD**, ou *standard definition*, consegue realizar uma transmissão serial de bits a uma taxa de 270 Mbps. Já os transmissores digitais SDI, podem transmitir vídeo digital a uma taxa de 1,45 Gbps ou até mesmo a uma taxa de 3 Gbps.

Como a central de vídeo implementada neste projeto transmitirá seu vídeo via rede, o padrão SDI, por requerer uma estrutura de cabos coaxiais de alta performance, não será utilizado. Entretanto, os conceitos básicos de transmissão digital são facilmente entendidos por meio do estudo da transmissão de vídeo digital via SDI. O Raspberry Pi, assim como nos exemplos supracitados, capturará tensões analógicas correspondentes aos três canais monocromáticos, converterá essas tensões em valores binários digitais, e realizará o processamento digital necessário para realizar a transmissão de vídeo via rede. O vídeo digital possui grande variedade de aplicações, pois, a partir do processamento digital, pode ser facilmente moldado em vários esquemas de transmissão.

3.1 A RESOLUÇÃO DE VÍDEOS

A resolução de um vídeo é definidade pela quantidade de *pixels* contidos na tela. Quanto maior for a resolução, mais detalhada será a imagem. O vídeo digital é reproduzido na tela da seguinte maneira: cada *pixel* do monitor adquire um valor digital correspondente a uma cor. A ordem na qual isso ocorre é: começando no canto superior esquerdo da tela, e a partir daí, da esquerda para a direita, e de cima para baixo. Quando todos os *pixels* da tela recebem a informação de cor, há a formação de um *frame,* ou quadro. Monitores modernos de computadores utilizam o padrão de cor *True Color*, onde cada cor é representada em 32 *bits*. Consequentemente, observa-se que existem mais de 4 bilhões de possibilidades de cor por pixel nesses monitores. Quanto maior for a resolução de vídeo, maior será a banda requerida para transmissão de informações de vídeo e maior será a necessidade de poder de processamento do transmissor de vídeo. Cabe ressaltar que a medida de resolução aqui considerada, o número de *pixels* na tela, não é o único parâmetro de medida de nível de detalhes da imagem: uma alta resolução de nada adiantará se a imagem transmitida em si não apresentar grande número de detalhes. A Figura 12 ilustra uma imagem em diferentes resoluções.

Figura 12: A mesma imagem sob diferentes resoluções. O fator acima da imagem demonstra o número de pixels existentes [10].

3.2 COMPRESSÃO DE VÍDEO

A compressão de vídeo é a redução, por meio de diversas técnicas, das informações necessárias para a transmissão de um vídeo. Para se obter uma visão prática da importância da compressão de vídeo, é pertinente notar que um vídeo gravado em uma resolução *Full HD* de 1920 x 1080 *pixels* é transmitido, sem compressão, a uma taxa de 1,485 Gbps no padrão digital SDI. Esta taxa de *bits*, ou *bitrate*, é simplesmente inviável para transmissão de vídeo, seja a transmissão via rede (uma transmissão para o *site* de conteúdo de mídias *YouTube*, por exemplo) ou via radiofrequência (uma transmissão de TV digital, por exemplo). Uma mídia de alta capacidade, como o *Blu-ray* de 25 GB, por exemplo, seria totalmente preenchida em menos de três minutos. Torna-se necessária, portanto, a compressão do vídeo original. E, em se tratando de compressão, existe sempre uma relação de compromisso fundamental entre qualidade e taxa de bits. Uma das técnicas mais populares de compressão, justamente por conciliar muito bem esses dois polos, é o padrão MPEG [10].

3.2.1 O PADRÃO MPEG

O padrão MPEG (*Motion Picture Experts Group*) é imensamente popular justamente por não ser um padrão, no sentido estrito da palavra. A nomenclatura mais adequada seria, talvez, a nomenclatura de *técnicas* MPEG. Isso ocorre porque o padrão MPEG se refere a uma grande variedade de técnicas, cada uma desenvolvida para algum propósito específico, baseadas em princípios semelhantes [10]. Qualquer técnica de compressão baseia-se em dois princípios cruciais: a melhoria da codificação e a eliminação de redundância [10].

A melhoria da codificação é obtida por meio de uma representação digital mais simples e eficiente da informação. Claramente, tal técnica é inviável para sinais analógicos, o que sinaliza mais uma vantagem da conversão analógico-digital: *o vídeo digital pode ser representado por diferentes codificações*. Até mesmo o padrão mais simples de codificação de vídeo digital apresenta inúmeros *bits* de redundância. O desafio consiste em retirar redundância da mensagem da maneira mais eficiente possível, e para isso existem inúmeras técnicas, como o código de Huffman e a técnica de *run-length coding*, por exemplo [10].

As técnicas de compressão MPEG baseiam-se em uma complexa combinação desses dois princípios fundamentais. Idealmente, essas técnicas resultariam sempre em uma compressão sem perdas, ou *lossless compression*, no jargão de vídeo. Destarte, pode-se recuperar integralmente a informação original a partir da informação comprimida. Infelizmente, a compressão sem perdas dificilmente atende as demandas de redução de dados exigidas por sistema de vídeo [10]. A compressão *lossless* é frequentemente utilizada após a compressão com perdas, para diminuir ainda mais a taxa de transmissão de dados.

A compressão com perdas, ou *lossy compression*, reduz a transmissão de dados por meio da eliminação de informações irrelevantes ou de baixa relevância. É interessante notar que a avaliação da importância da informação depende do contexto da aplicação de compressão. No caso do vídeo digital, a informação julgada irrelevante é aquela que não pode ser percebida pelo sistema visual humano [10]. É possível, por conseguinte, adotar técnicas de compressão de vídeo digital com perdas, cuja qualidade do produto visual final é indissociável da qualidade do produto visual inicial. Uma técnica de compressão que elimina apenas as informações visuais imperceptíveis pelo olho humano é denominada *visually lossless*, no jargão de vídeo. Na transmissão de sinais de televisão digital, por exemplo, é possível reduzir a taxa de bits de 1,485 Gbps para cerca de 35 Mbps, sem que o telespectador note que o sinal sofreu qualquer tipo de compressão [10].

O julgamento das informações irrelevantes contidas em um sinal de vídeo digital é baseado em um fato curioso do sistema visual humano: o olho humano possui uma imensa sensibilidade à luminância do sinal de vídeo. O ser humano é capaz de perceber minúsculas alterações no nível de luz de uma imagem. A sensibilidade à cores, pelo outro lado, é significativamente menor, o que permite a seguinte conclusão: *é importante priorizar a informação de luminância em detrimento da informação de crominância* [10]. Outro importante fator a ser levado em conta é a maior sensibilidade do sistema visual humano à cor verde, em relação as componentes cromáticas azul e vermelha. No entanto, como a imagem capturada pela câmera é dividida em três canais cromáticos (RGB), cada um dos canais possui informação de luminância e crominância, o que dificulta a tarefa de eliminação de informações visualmente irrelevantes. Consequentemente, torna-se precisa a adoção de uma nova representação do sinal de vídeo, por meio de um sinal de luminância (normalmente designado como sinal **Y**), e dois sinais de diferença de cor, comumente designados por **U** e **V**.

Os sinais de diferença de cor baseiam-se em uma representação bastante interessante do espaço de cores, também chamado de *color space*. No espaço de cores, cada cor é representada por um vetor, onde o módulo representa a intensidade da cor, ou sua saturação. A fase do vetor representa a matiz, ou *hue*, da cor. Em se tratando de sinais de vídeo, é bastante comum representar as cores em diagrama vetorial, onde o sinal **U** representa o eixo das abscissas e simboliza a subtração (**B – Y**), ou a diferença de amplitude entre o sinal do canal azul e o sinal de luminância, e o sinal **V** representa o eixo das ordenadas e simboliza a subtração (**R – Y**), ou a diferença de amplitude entre o sinal do canal vermelho e o sinal de luminância. A Figura 13 ilustra a representação vetorial do espaço de cores para um sinal de luminância de amplitude **Y = 0,5**. Esta representação é comumente referida como espaço **YUV**. Na Figura 14 observamos uma imagem representada no espaço YUV e suas respectivas componentes Y, U, e V.

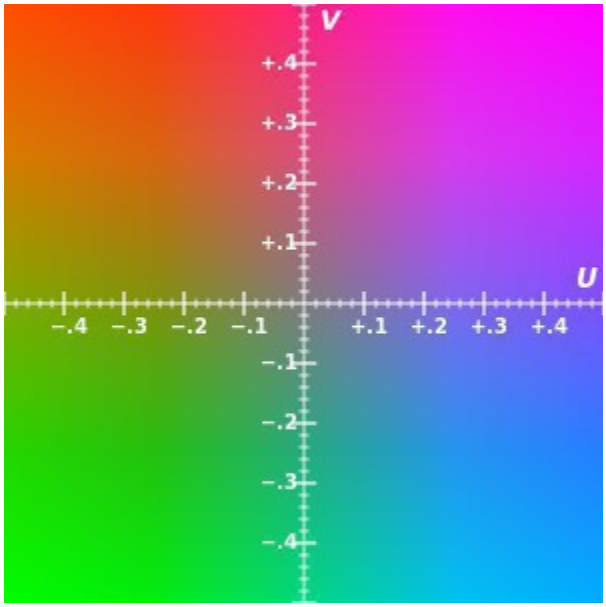

Figura 13: Representação gráfica do espaço YUV [10].

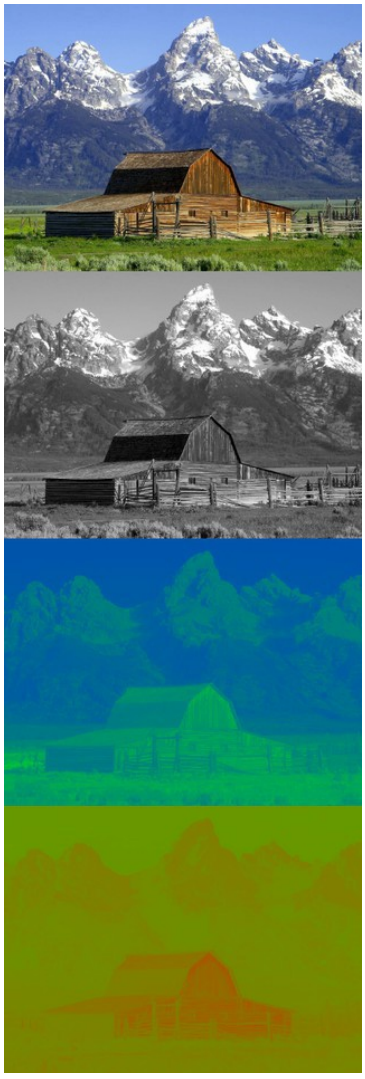

Figura 14: Componentes Y, U, e V da imagem original [10].

O propósito por trás dos sinais de diferença de cor é a redução da banda necessária para transmissão de informação de crominância. Conforme visto anteriormente, a câmera transmite 3 canais de crominância, com informação de luminância contida em cada um desses canais. Com os sinais de diferença de cor, é apenas necessário transmitir os sinais relativos às cores azul e vermelho. Isso ocorre pois é possível recuperar a cor verde a partir da soma dos dois sinais de diferença de cor. Pode-se notar na Figura 13 que a soma vetorial dos sinais **U** e **V** resulta na cor magenta. A partir de uma inversão de aproximadamente 180º do vetor magenta, obtém-se o vetor correspondente à cor verde, eliminando, assim, a necessidade de transmissão de três informações cromáticas. O espaço de cores **YUV** é alternativamente denominado **YPbPr** (para vídeo analógico) e **YCbCr** (para vídeo digital). As matrizes abaixo relacionam a transformação de **RGB** para **YUV**, e o inverso, para vídeos digitais de alta definição. Nota-se que a componente de cor verde (**G**) possui maior peso na determinação da componente de luminância **Y**, em decorrência da maior sensibilidade da visão humana a esta componente.

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Matrizes de conversão entre espaços de cor RGB e YUV.

A compressão MPEG utiliza técnicas digitais para converter uma imagem em sua representação mais eficiente, no sentido de economia de *bits*, no espaço **YUV**. Na Figura 15 observamos que os sinais YPbPr, capturados pela câmera, são transformados, por um conversor analógico-digital, nos sinais YCbCr. Embora o exemplo ilustrado demonstre ainda o passo adicional de codificação dos sinais para o padrão SDI, tal ação não é necessária para a compressão de vídeos via MPEG.

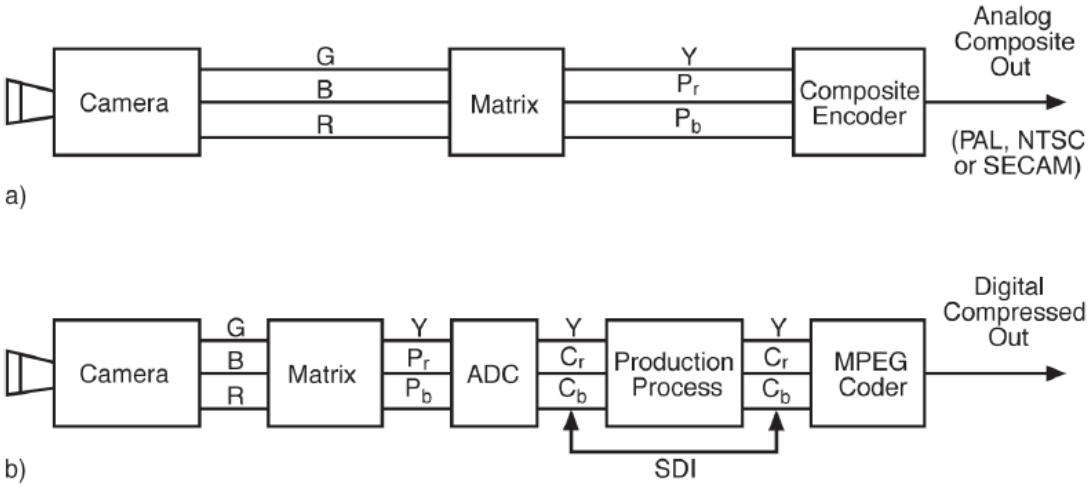

Figura 15: Transformação de cor do espaço RGB para o espaço YUV. Na parte superior da imagem, vemos o processo aplicado em um vídeo analógico. Na parte inferior, observa-se o processo mais complexo, porém manipulável, de conversão para o espaço YUV de um vídeo digital [10].

A compressão de vídeo digital baseia-se em dois princípios fundamentais da teoria da informação: **entropia** e **redundância**. A entropia, de acordo com a teoria da informação, é a quantidade de incerteza contida em uma mensagem. Quanto maior for o grau de aleatoriedade da mensagem, maior será sua entropia. Em se tratando de vídeo, a entropia representa as alterações que ocorrem na imagem entre dois quadros sucessivos. As componentes de informação que permanecem praticamente constantes entre um quadro e outro representam a redundância do vídeo. A redundância pode ser espacial, no caso de um grande conjunto de *pixels* adjacentes que possuam valores bastante semelhantes, ou até idênticos, ou temporal, quando determinadas áreas da imagem permanecem iguais entre um quadro e outro. O modelo de compressão MPEG tem como objetivo transmitir apenas a entropia do sinal, de forma que a redundância possa ser calculada pelo decodificador. O compressor de vídeo ideal é capaz de extrair apenas a entropia do sinal do vídeo, mas tal compressor não existe no mundo prático. Limitações de tempo e custo exigem que a complexidade, e consequentemente a eficiência, do decodificador sejam reduzidos. Uma das grandes vantagens do padrão de compressão MPEG, é a sua versatilidade: o usuário tem a possibilidade de arbitrar o compromisso entre eficiência de compressão e complexidade do processo de compressão. A Figura 16 ilustra as limitações de codificadores MPEG reais e os *tradeoffs* existentes entre fator de compressão e complexidade e latência do codificador. Na implementação da central de vídeo demonstrada ao fim deste trabalho, observar-se-ão as limitações práticas, como latência e erros de compressão, criadas quando o grau de exigência sobre o codificador torna-se demasiadamente alto.

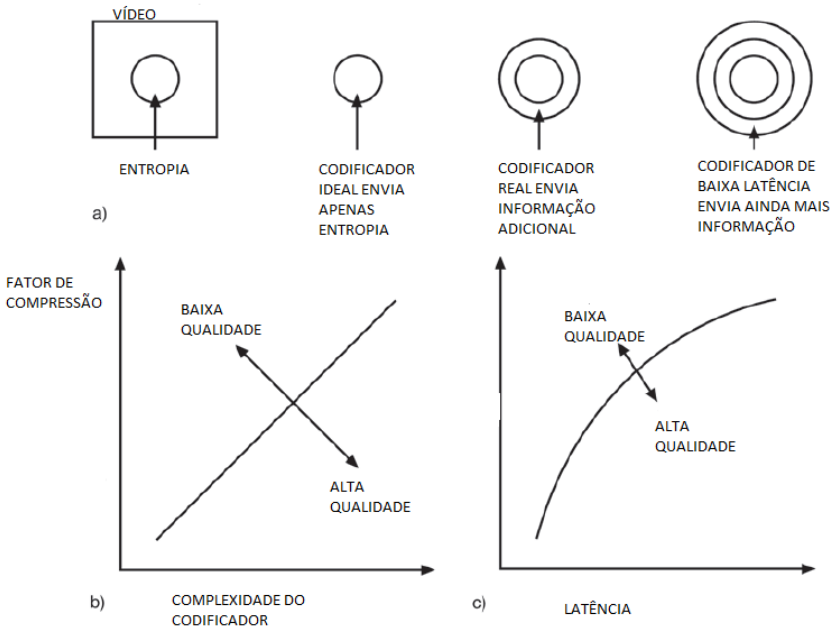

Figura 16: a) Limitações práticas de codificadores. b) Relação entre fator de compressão e complexidade do decodificador. c) Relação entre fator de compressão e latência do decodificador [10].

A entropia de sinais de vídeo varia de acordo com o vídeo em questão. O padrão MPEG permite ao usuário duas opções interessantes: *constant bitrate* (**CBR**) e *variable bitrate* (**VBR**). No caso da opção CBR, a transmissão do vídeo comprimido é feita sempre à mesma taxa de bits, sendo esta pré-definida no momento da compressão. Já na opção VBR, o vídeo é transmitido a uma maior taxa de bits nos momentos em que há maior entropia no sinal de vídeo, e, em momentos de maior redundância, a taxa de bits é reduzida. A opção CBR é comumente usada em emissoras de televisão em função de sua praticidade. A opção VBR é mais utilizada em mídias como DVD e *Blu-ray*, para garantir maior qualidade de vídeo ao usuário [10]. A câmera do Raspberry Pi utiliza o padrão de compressão MPEG-4 *Part 10*, ou H.264, com taxa de *bits* variável.

Existem duas maneiras de explorar a redundância em sinais de vídeo: via codificação *intra-frame*, e codificação *inter-frame*. A codificação *intra-frame* explora maneiras de eliminar a redundância espacial, ao passo que a codificação *inter-frame* visa a eliminar a redundância temporal. A popular técnica de compressão de imagens JPEG baseia-se em uma efetiva codificação intra-frame. A compressão MPEG, por sua vez, faz recurso de ambas as técnicas de codificação para obtenção de um padrão de compressão bastante eficiente.

A codificação *intra-frame* se aproveita da representação da imagem em forma matricial. Em um vídeo digital, cada *frame* é representado por uma matriz de M linhas e N colunas. Cada elemento da matriz representa um *pixel* da imagem, e o produto $M \times N$ resulta na resolução da imagem. O valor de cada elemento da matriz indica, de acordo com a codificação adotada, informações relativas à cor e

à luminância do *pixel*. Cabe ressaltar que um único *frame* pode ser também representado por diversas matrizes, onde cada matriz representa um aspecto da imagem. Como exemplo, pode-se fazer uma representação matricial de uma imagem no espaço YUV por meio de três matrizes, com duas matrizes de crominância (U e V) e uma matriz de luminância (Y).

A técnica de compressão *intra-frame* baseia-se na análise matricial da imagem a partir de uma técnica matemática conhecida como transformada discreta de cosseno, ou *discrete cosine transform* (DCT). A DCT representa, de maneira similar à transformada de Fourier, uma matriz *M x N* por meio de uma matriz de dimensões idênticas, onde cada elemento indica a amplitude de uma componente senoidal. A frequência de cada componente senoidal é obtida a partir da posição do elemento na matriz; elementos de baixa frequência localizam-se no começo das linhas e colunas da matriz. A DCT de uma matriz *A* de dimensões *M x N* é comumente representada pela equação I.

$$B_{pq} = \alpha_p \alpha_q \sum_{p=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad \begin{cases} 0 \leqslant p \leqslant M-1 \\ 0 \leqslant q \leqslant N-1 \end{cases}$$

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{(M)}}, & p=0 \\ \sqrt{\frac{2}{M}}, & 1 \leqslant p \leqslant M-1 \end{cases} \qquad \alpha_q = \begin{cases} \frac{1}{\sqrt{(N)}}, & q=0 \\ \sqrt{\frac{2}{N}}, & 1 \leqslant q \leqslant M-1 \end{cases} \qquad \text{(I)}$$

Onde os valores B_{pq} representam os coeficientes da DCT de *A*.

A matriz *A* pode ser reconstruída a partir da matriz de coeficientes B_{pq} por meio da transformada inversa dada pela equação II:

$$A_{mn} = \sum_{p=0}^{M-1} \sum_{n=0}^{N-1} \alpha_p \alpha_q B_{pq} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad \begin{cases} 0 \leqslant m \leqslant M-1 \\ 0 \leqslant n \leqslant N-1 \end{cases}$$

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{(M)}}, & p=0 \\ \sqrt{\frac{2}{M}}, & 1 \leqslant p \leqslant M-1 \end{cases} \qquad \alpha_q = \begin{cases} \frac{1}{\sqrt{(N)}}, & q=0 \\ \sqrt{\frac{2}{N}}, & 1 \leqslant q \leqslant N-1 \end{cases} \qquad \text{(II)}$$

A frequência espacial de uma imagem, conceito importante na codificação *intra-frame*, é representada pela matriz obtida a partir da aplicação da DCT a um *frame*. A imagem pode, portanto, ser representada por uma matriz de componentes senoidais, e quando as componentes de maior frequência possuem amplitude significativa, a imagem é dita de alta frequência espacial. Considere-se o exemplo da Figura 17, onde há uma imagem que varia gradativamente do branco ao preto e do preto ao branco. Neste caso, por simplicidade, temos apenas um sinal de luminância, onde o valor 255 representa o branco puro e o valor 0 representa o preto puro. A imagem foi construída de tal forma que

a amplitude de cada *pixel x* da imagem é representada pela função senoidal abaixo (os valores da função foram escalonados para que o máximo correspondesse a 255 e o mínimo a 0).

$A(x)=\cos(x)$, cuja frequência é dada por $f=1/(2\pi)Hz$

Figura 17: Região de baixa frequência espacial

A Figura 18 mostra uma imagem de alta frequência espacial, e a função senoidal que determina a amplitude da luminância do *pixel x* é dada por $A(x)=\cos(3x)$

Figura 18: Região de alta frequência espacial

Nos exemplos abaixo é possível observar diversas aplicações da transformada discreta de cosseno. Na Figura 19 aplica-se a transformada a uma simples sequência de números reais, e já é possível observar que apenas os primeiros coeficientes da DCT possuem amplitudes relevantes. Na Figura 20 observa-se a aplicação da transformada em uma linha de 30 *pixels* de uma imagem, com as amplitudes dos coeficientes da DCT decaindo de maneira semelhante. Nas Figuras 21, 22 e 23 observamos a aplicação da transformada sobre uma imagem completa, e sua reconstrução apenas a partir dos coeficientes de magnitude igual ou superior a 10. Percebe-se que é possível eliminar uma quantidade significativa de informação sem comprometer substancialmente a qualidade da imagem reconstruída.

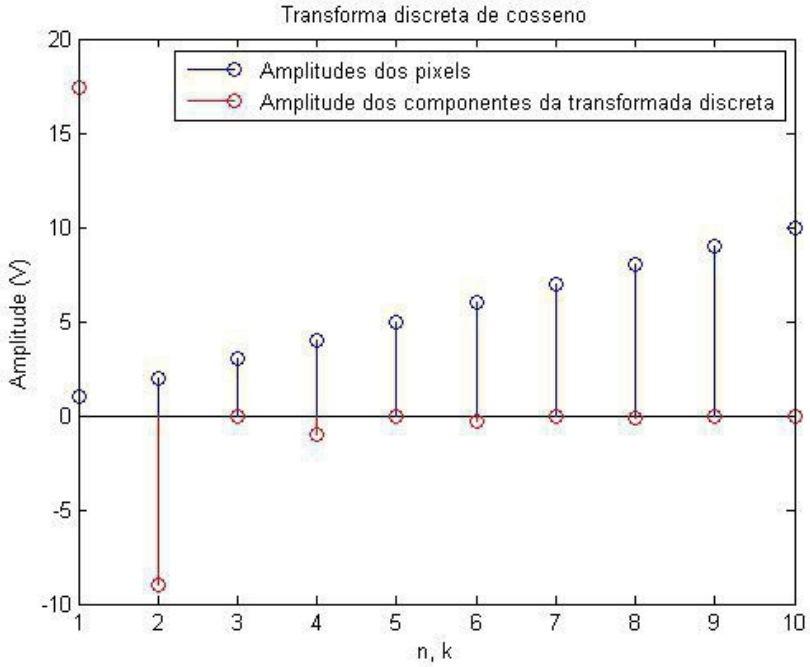

Figura 19: Transformada discreta de cosseno aplicada à uma sequência linear de 10 números.

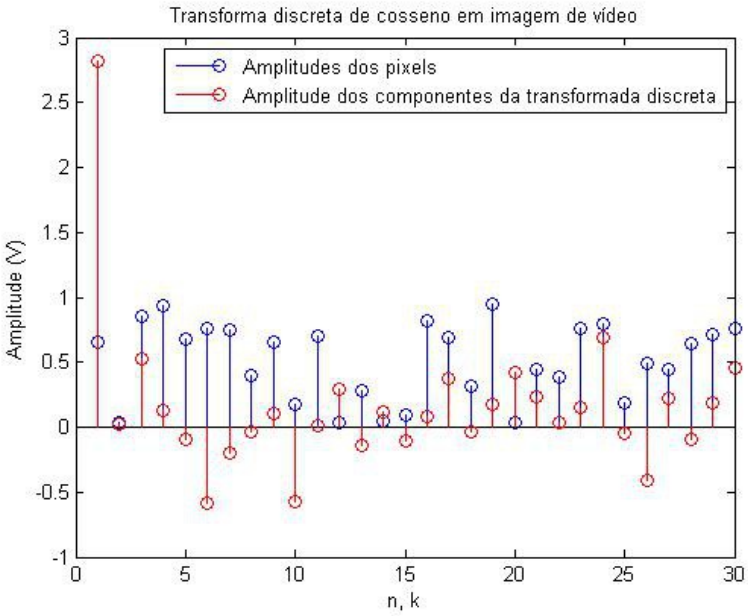

Figura 20: DCT aplicada em uma região de uma imagem de um sinal de vídeo digital. As componentes de magnitude mais relevante são aquelas de baixa frequência espacial.

Figura 21: Imagem JPEG de resolução 512 x 512.

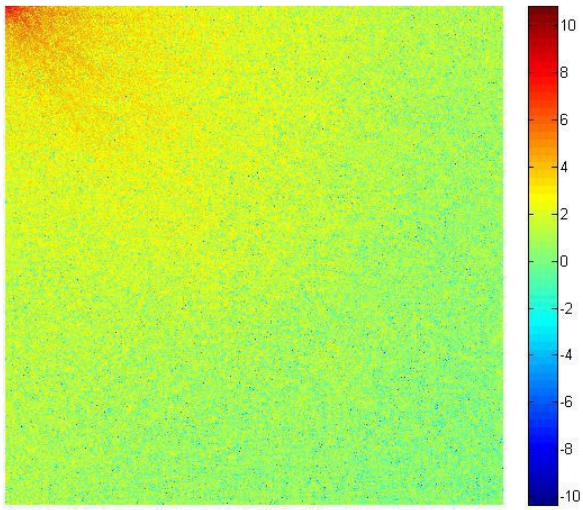

Figura 22: Mapa de temperatura da DCT da imagem anterior. Observa-se que os coeficientes de alta amplitude concentram-se no canto esquerdo superior da matriz, e correspondem a componentes senoidais de baixa frequência.

Figura 23: Imagem reconstruída a partir da DCT inversa. Apenas as componentes da DCT de amplitude igual ou superior a 10 foram usados na reconstrução da imagem.

Conforme as figuras acima demonstram, as componentes de alta frequência espacial possuem amplitude muito baixa, podendo ser descartadas para redução da taxa de transferência de *bits* do vídeo. As transformadas *wavelet* são usadas em imagens de altíssima frequência espacial, e servem o mesmo propósito da DCT. A compressão MPEG faz uso de ambas as transformadas para eliminação de redundância espacial.

A codificação *inter-frame* verifica a diferença entre dois *frames*, ou duas imagens sucessivas dentro de um sinal de vídeo. Caso o decodificador receba uma imagem completa, ele poderá construir a próxima imagem a partir da informação das diferenças entre a nova imagem e a antiga imagem, enviada pelo codificador. Sob esta ótica, o sinal de diferença de informação ocupará maior banda quando houver maior movimento entre duas imagens sucessivas. O padrão MPEG ainda é capaz de detectar o movimento entre duas imagens sucessivas e, a partir daí, mover partes da antiga imagem para outra região, tentando emular o movimento ocorrido no vídeo original. Essa representação do movimento é vetorial e recebe o nome de *motion vector*, ou vetor de movimento. A transmissão do vetor de movimento, do codificador para o decodificador, requer menos informação do que a transmissão do sinal de diferença de informação entre duas imagens sucessivas.

O Raspberry Pi utiliza o padrão de compressão MPEG H.264, padrão este designado para aplicações de baixo poder de processamento, permitindo uma taxa de *bits* máxima de 135 Mbps [10]. Será visto adiante que, para taxas de transmissão muito altas, o vetor de movimento da codificação MPEG H.264 do Raspberry Pi não é suficiente para representar a diferença entre duas imagens

sucessivas, ocasionando erros de compressão. A Figura 24 ilustra a evolução do padrão MPEG no decorrer dos anos, mostrando as mínimas taxas de transmissão de *bits* obtidas com cada padrão.

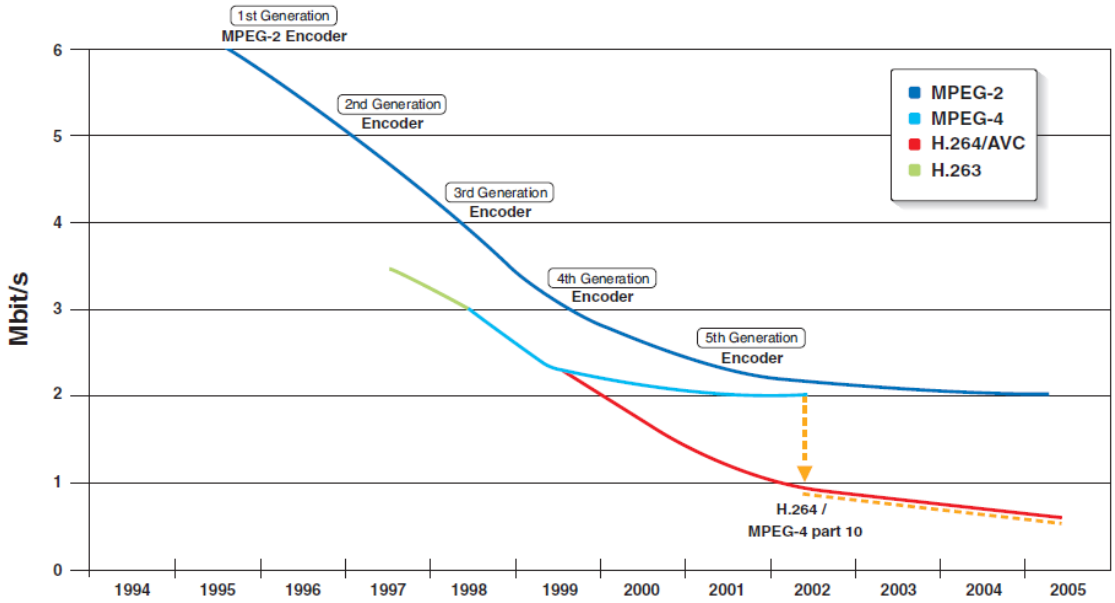

Figura 24: Evolução do padrão MPEG ao longo dos anos. O padrão H.264 é utilizado pelo Raspberry Pi [10].

3.3 STREAMING

A definição de *streaming* traduz-se em um vídeo sem final definido sendo reproduzido praticamente em tempo real. O vídeo comprimido no padrão MPEG é transmitido pelo codificador como um *stream* elementar, ou *elementary stream*. Para maior comodidade no processamento digital de dados, o *stream* elementar é dividido em inúmeros blocos de dados, denominados *packets*, ou pacotes. Este *stream* quantizado é designado por *packetized elementary stream* (**PES**), onde cada pacote possui *headers*, ou cabeçalhos, com informações de identidade relativa ao pacote, e informações de temporização, para que o decodificador saiba como organizar os pacotes recebidos. Em se tratando de *streaming*, um grande indicador da qualidade da transmissão é o tempo de latência. O tempo de latência é o intervalo de tempo que ocorre entre a captação da imagem pela câmera e a subsequente visualização da imagem pelo espectador. Um *streaming* de baixa latência exige maior poder de processamento, pois requer que a informação de vídeo seja transmitida de forma mais rápida.

Antes de ser transmitido, o PES passa por um multiplexador, que o transforma em um *transport stream* (**TS**). O *transport stream*, possui, além dos pacotes relativos ao vídeo comprimido, metadados relativos ao próprio *stream*, como CBR ou VBR, por exemplo. Múltiplos vídeos podem ser multiplexados em único TS, pois no *transport stream* há a transmissão de uma tabela de associação de programas (*program association table,* ou **PAT**), relacionando todos os vídeos sendo transmitidos no

stream. Cada pacote contido no *transport stream* possui tamanho fixo de 188 bytes, e carrega um código de identificação de programa (*program identifier code*, ou **PID**). Todos os pacotes pertencentes a um mesmo PES possuem o mesmo PID. Um eficaz sistema de sincronização, por fim, garante que o decodificador abra e traduza os pacotes do *transport stream* corretamente. A Figura 25 demonstra o fluxo de *streaming* de vídeo. A partir do conhecimento dos conceitos básicos de *streaming* de vídeo, pode-se partir para a configuração do Raspberry Pi como plataforma de *streaming*.

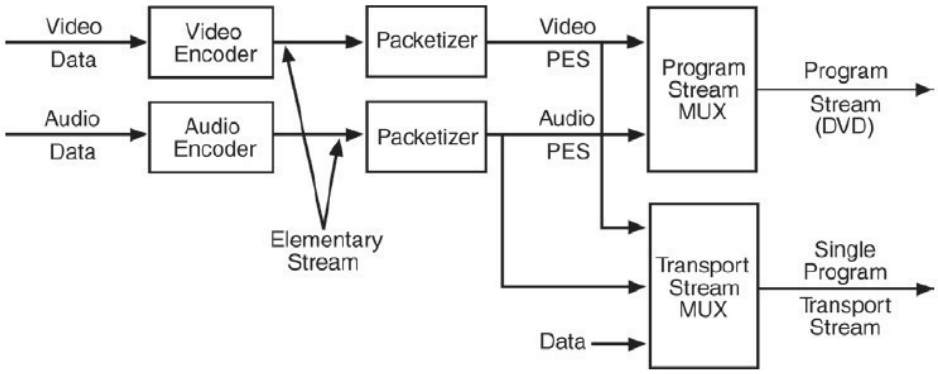

Figura 25: Fluxo da codificação MPEG. Nota-se que os mesmos princípios da codificação de vídeo aplicam-se à codificação de áudio [10].

4. MONITORAMENTO DE VÍDEO

Para que se possa realizar monitoramento de vídeo com o Raspberry Pi, é interessante que se possua um conhecimento das capacidades e funções de sua câmera de vídeo. As tabelas 1 e 2 demonstram as funções de fotografia e vídeo suportadas pela câmera. A câmera é comandada a partir do *prompt* de comando padrão do sistema operacional Linux, popularmente conhecido como *Bash*. O comando usado para captura de fotografia é o comando ***raspistill***. O módulo de câmera captura fotografias em uma resolução padrão de 2592 x 1944 *pixels* e as salva em formato padrão JPEG. O vídeo padrão é gravado em resolução *Full HD* de 1920 x 1080 *pixels* no formato H.264, por meio do comando ***raspivid***.

Tabela 1: Configurações básicas de fotografia da câmera.

PARÂMETRO	**DESCRIÇÃO**
-w, --*width*	Configura largura da imagem em *pixels*
-h, --*height*	Configura altura da imagem em *pixels*
-q, --*quality*	Define a qualidade do codificador JPEG, variando de 1 a 100. O valor padrão de qualidade das fotos é 85
-r, --*raw*	Gera a imagem em formato *raw*. Este é o formato de imagem produzido pela câmera com o menor processamento possível. Uma imagem *raw* é frequentemente comparada aos filmes negativos de fotografia, pois, embora não seja usada diretamente como imagem, contém todas as informações relativas à imagem em si. A imagem em formato *raw* é gerada em um *colorspace* com mais opções de cores que o *colorspace* de imagens em formatos convencionais. Esta opção é aconselhável, portanto, quando se deseja uma imagem com a maior quantidade possível de informações relativas a luminância e crominância.
-o, --*output*	Define o nome do arquivo de imagem a ser salvo.
-l, --*latest*	Ligar a última imagem capturada ao nome definido por este parâmetro
-t, --*timeout*	Tempo em *ms* entre a execução do comando e a captura da foto. Padrão de 5000 *ms*.
-th, --*thumbnail*	Configurar parâmetros de *thumbnail,* a imagem em miniatura gerada para identificar o vídeo. Podem ser configurados: largura, altura, e fator de qualidade.
-d, --*demo*	Modo de demonstração das diversas opções de câmera.
-e, --*encoding*	Codificação a ser usada (JPG, GIF, BMP, PNG)
-x, --*exif*	Acrescentar EXIF *tags* à imagem; metadados da imagem. Por meio dessas informações pode-se visualizar dados referentes à resolução, data, padrão de compressão, e outros aspectos
-tl, --*timelapse*	Modo *timelapse*. Captura uma fotografia a cada ciclo de milissegundos definido pelo usuário
-pv, --*fullpreview*	Executar o *preview* com a resolução de captura de *still photographs*.
-k, --*keypress*	Espera o usuário pressionar a tecla ENTER para capturar a fotografia, e X + ENTER para sair do modo *keypress*.

-p, --*preview*	Configura altura e largura da janela de *preview*
-f, --*fullscreen*	Habilita *preview* em tela cheia.
-op, --*opacity*	Opacidade da janela de *preview* (0 a 255)
-n, --*nonpreview*	Não exibir janela de preview
-sh, --*sharpness*	Configurar nitidez da imagem (-100 a 100)
-co, --*contrast*	Configurar contraste da imagem (-100 a 100)
-br, --*brightness*	Configurar brilho da imagem (0 a 100)
-sa, --*saturation*	Configurar saturação da imagem (-100 a 100)
-ev	Configura compensação EV, ou *exposure value*. Desta forma, pode-se ajustar à exposição da abertura da lente à luz para aumentar ou diminuir o brilho da imagem.
-hf, -hflip	Configurar *flip* horizontal
-vf, -vlfip	Configurar *flip* vertical
-roi	Definir região de interesse na imagem

Tabela 2: Configurações básicas de gravação de vídeo da câmera.

PARÂMETRO	**DESCRIÇÃO**
-w, --*width*	Configura largura da imagem em *pixels*
-h, --*height*	Configura altura da imagem em *pixels*
-b, -bitrate	Definir taxa de transmissão de *bits* por segundo
-fps, --*framerate*	Definir o número de quadros por segundo (*frames per second*) a serem gravados
-o, --*output*	Define o nome do arquivo de vídeo a ser salvo.
-e, --*penc*	Exibe imagem de *preview* após a codificação do vídeo.
-t, --*timeout*	Tempo de gravação em milissegundos. Para gravar indefinidamente, configurar para 0.
-td, --*timed*	Define um período entre gravação e pausa, em milissegundos.
-d, --*demo*	Modo de demonstração das diversas opções de câmera.
-k, --*keypress*	Alterna entre gravação e pausa com o pressionamento da tecla *Enter*
-i, --*initial*	Define estado inicial da câmera quando da execução do comando *raspivid*: *record* ou *pause*. O padrão é *record*, isto é, a câmera começará a gravação de vídeo quando o comando *raspivid* for executado.
-tl, --*timelapse*	Modo *timelapse*. Captura uma fotografia a cada ciclo de milissegundos definido pelo usuário
-pv, --*fullpreview*	Executar o *preview* com a resolução de captura de

	still photographs.
-k, --*keypress*	Espera o usuário pressionar a tecla ENTER para capturar a fotografia, e X + ENTER para sair do modo *keypress*.
-p, --*preview*	Configura altura e largura da janela de *preview*
-f, --*fullscreen*	Habilita *preview* em tela cheia.
-op, --*opacity*	Opacidade da janela de *preview* (0 a 255)
-n, --*nonpreview*	Não exibir janela de preview
-sh, --*sharpness*	Configurar nitidez da imagem (-100 a 100)
-co, --*contrast*	Configurar contraste da imagem (-100 a 100)
-br, --*brightness*	Configurar brilho da imagem (0 a 100)
-sa, --*saturation*	Configurar saturação da imagem (-100 a 100)
-ev	Configura compensação EV, ou *exposure value*. Desta forma, pode-se ajustar à exposição da abertura da lente à luz para aumentar ou diminuir o brilho da imagem.
-hf, -hflip	Configurar *flip* horizontal
-vf, -vlfip	Configurar *flip* vertical
-roi	Definir região de interesse na imagem

4.1 STREAMING DE VÍDEO VIA CONEXÃO DIRETA

A maneira mais simples de realizar *streaming* de vídeo com o **Rpi** é a transferência de vídeo via rede *Ethernet* ou *Wi-Fi* diretamente para o cliente, o computador ou dispositivo que visualizará o vídeo. O vídeo em formato H.264 é enviado ao cliente cujo sistema operacional pode ser Windows, Linux, ou iOS. Independentemente da escolha de sistema operacional, é necessária a instalação de um programa reprodutor de vídeo no cliente que seja compatível com o formato H.264. Conforme foi visto anteriormente, a câmera do *Raspberry Pi* grava vídeos apenas na extensão H.264, extensão esta de escassa compatibilidade com os *media players* disponíveis no mercado. Sugere-se a escolha do *Mplayer,* um reprodutor de mídias de código livre, ou de qualquer outro *media player* compatível com o formato H.264.

O exemplo abaixo descreve o procedimento necessário para visualizar o *streaming* de vídeo em uma máquina com sistema operacional Windows. O endereço IP do servidor (Raspberry Pi) foi definido como **192.168.1.155**, enquanto o endereço IP do cliente foi definido como **192.168.1.108**. O sistema operacional Debian possui a ferramenta *Netcat,* responsável por gerenciar a transferência de dados via TCP/IP, apresentando ampla compatibilidade com diversos sistemas operacionais. A conexão com o servidor é feita de acordo com os procedimentos a seguir:

► Abrir o *prompt* de comando *cmd.exe* ou o *Microsoft Powershell*

► Executar o seguinte comando:

[Diretório do Netcat]\nc.exe -L -p 5001 | [Diretório do Mplayer]\mplayer.exe -fps 31 -cache 1024 -

Figura 26: Após a execução do programa, o Netcat monitora a porta 5001. São vistas as diversas opções de controle do Mplayer.

O primeiro comando habilita o *Netcat* com as configurações -L e -p. O parâmetro **L** habilita o *listening mode*, ou modo de escuta. Nesta configuração, o *Netcat* monitora a porta determinada pelo usuário (no exemplo, a porta 5001), e, quando detecta uma conexão, realiza uma ação escolhida pelo usuário. Quando a conexão é finalizada, o *Netcat* volta a monitorar a porta especificada em busca de uma nova conexão. Vale ressaltar que o operador *pipe*, representado pelo símbolo | tem a função de ligar a saída do primeiro comando à entrada do segundo comando. No exemplo acima, quando ocorrer a detecção da conexão, o sinal de vídeo será aberto pelo *Mplayer,* mostrado na Figura 26.

O parâmetro **fps** força a taxa de quadros do vídeo ao valor determinado pelo usuário (no exemplo, 31 quadros por segundo ou *frames per second*). Esta opção é utilizada para proporcionar uma sensação de suavidade do vídeo ao usuário, visto que a taxa de quadros ou *framerate* costuma variar em *streams* de vídeo. O parâmetro **cache** determina a memória *cache* que será usada para realizar o *buffer* do vídeo, sendo o valor determinado em *kilobytes* (**KB**). A memória *cache* é a memória alocada pelo Raspberry para armazenar temporariamente o vídeo gravado, o *buffer*, antes que este seja transmitido.

A configuração do Raspberry Pi é ainda mais simples que a configuração do cliente. É preciso apenas executar, no terminal do *Debian*, o comando de gravação de vídeo, o comando **raspivid**, e ligar sua saída à entrada do comando do ***nc***, responsável por transmitir dados via protocolo TCP/IP. A Figura 27 ilustra a execução do comando no terminal.

```
raspivid -t 999999 -o - | nc [ENDEREÇO DE IP DO CLIENTE] 5001
```

O comando de gravação é configurado para 999999 milissegundos ou aproximadamente 17 minutos, o que equivale na prática à gravação contínua para este simples exemplo ilustrativo. Novamente é utilizado o operador *pipe* para conectar a saída do comando *raspivid* à entrada do comando *nc*, que ativa o *Netcat.* Neste segundo comando, basta apenas definir o endereço IP do cliente e a porta desejada, e o vídeo começará a ser transmitido. Efeitos e *timecode* podem ser acrescentados ao vídeo transmitido. As figuras 28 e 29 ilustram o vídeo transmitido e o vídeo capturado, respectivamente. Notou-se que o desempenho do *streaming* direto foi excelente, com uma latência de apenas 4 segundos para um *stream* de alta definição.

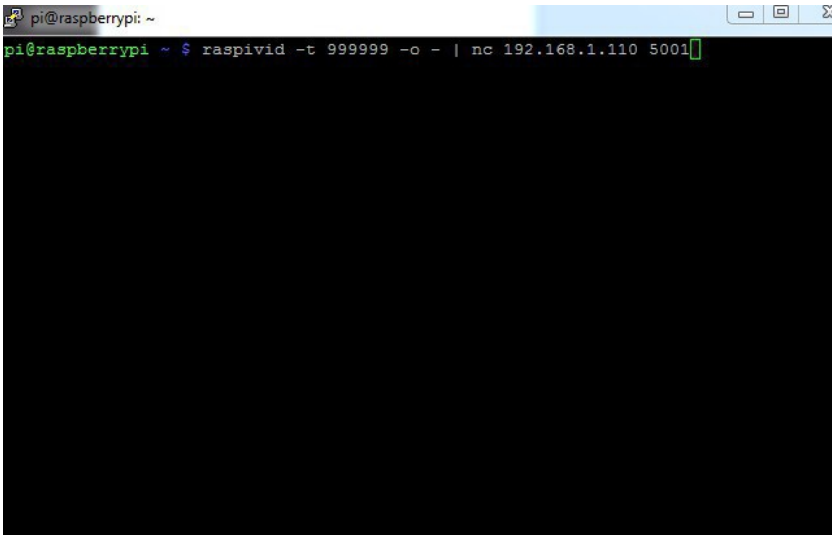

Figura 27: Comando de configuração do servidor de vídeo.

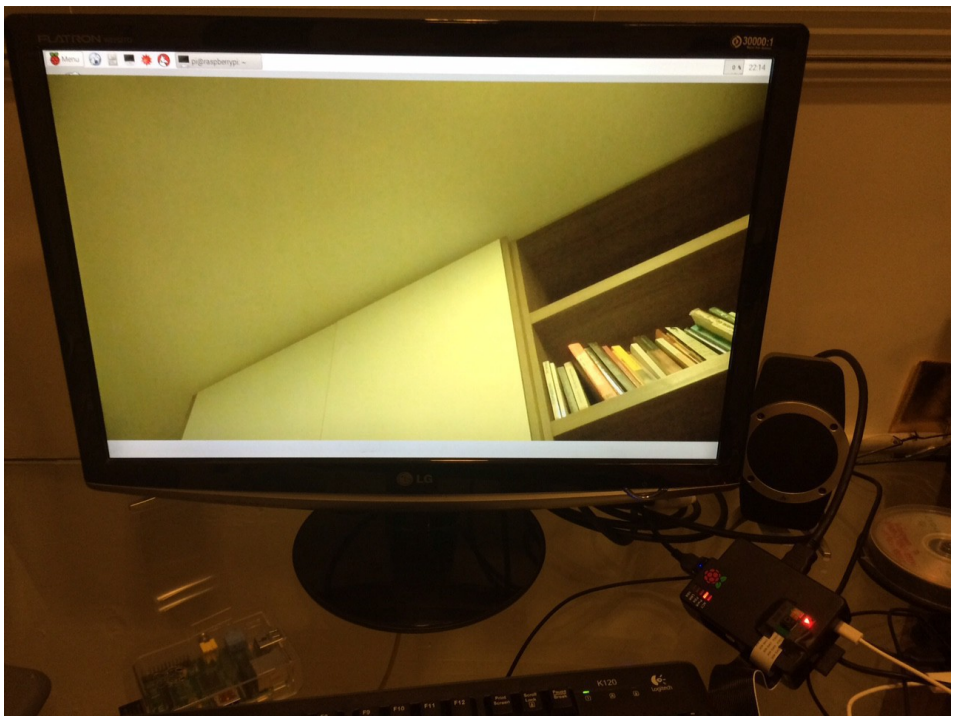

Figura 28: Imagem capturada no servidor

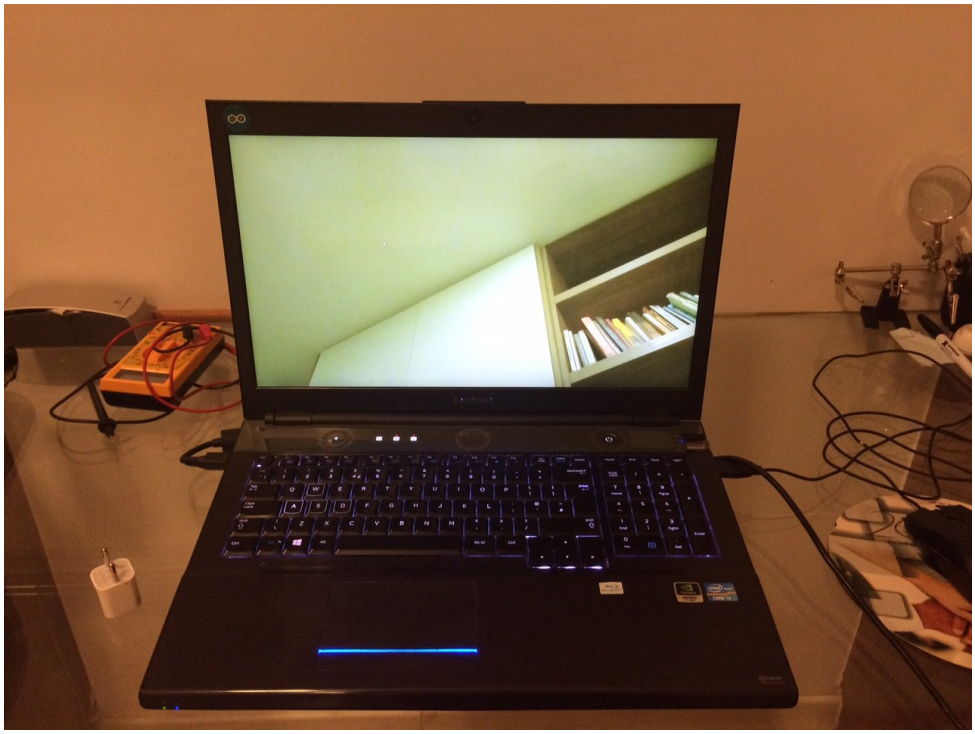

Figura 29: Vídeo reproduzido no cliente.

4.2 STREAMING DE VÍDEO VIA REDE

O método de *streaming* de vídeo via conexão direta apresentado anteriormente, embora funcional, apresenta diversas limitações. Em primeiro lugar, apenas um usuário é capaz de visualizar o vídeo transmitido. Ademais, quando o cliente encerra a visualização de vídeo, a transmissão é interrompida. Pode-se citar também a complicação da baixa compatibilidade do formato de vídeo H.264 com *media players* populares. Este último fator, por sinal, é um grande obstáculo para a visualização de vídeos por meio de aparelhos celulares. Tendo em vista estas limitações práticas, é interessante desenvolver um sistema de transmissão de vídeo que transmita o vídeo em formato acessível para um endereço na *web*. Desta forma, qualquer aparelho capaz de executar um *media player* será capaz de reproduzir o vídeo, e vários aparelhos poderão fazê-lo simultaneamente.

Existem inúmeros protocolos de *streaming* utilizados atualmente. O protocolo *HLS*, da empresa norte-americana *Apple*, *Inc.,* é adequado para dispositivos produzidos pela própria empresa, mas não é compatível com produtos *Microsoft* ou sistemas operacionais Linux. O MP4 fragmentado, ou *fragmented MP4,* funciona em produtos *Microsoft* e em produtos *Apple*, mas requer a instalação de um *plugin* no *browser*, o que compromete a visualização em aparelhos celulares. Um protocolo bastante interessante para solucionar o primeiro problema é o protocolo usado no *streaming* de vídeo de *webcams*: o protocolo MJPEG.

4.2.1 O PROTOCOLO MJPEG

O próprio nome MJPEG pode trazer à cabeça do leitor a associação do protocolo com o formato de imagens JPEG, que, assim como o padrão de compressão MPEG, baseia-se na DCT para eliminação de redundância espacial da imagem. Essa associação é bastante pertinente, pois o princípio básico do *streaming* MJPEG é a transmissão de imagens JPEG sequenciais. Esse simples princípio de compressão de vídeos, a transformação de um vídeo em uma sequência de imagens, tem seu custo: perde-se bastante eficiência de compressão em relação ao padrão H2.64, pois cada *frame* transmitido é uma imagem JPEG completa. No entanto, o protocolo MJPEG é compatível com *browsers* modernos, e portanto pode transmitir vídeo a diversos dispositivos.

É necessário também encontrar uma plataforma de *streaming:* uma plataforma capaz de converter imagens JPEG sequenciais em um *streaming* MJPEG. A câmera do Raspberry Pi, felizmente, possui a capacidade de capturar fotos em formato JPEG. A questão, portanto, é reduzida a buscar uma plataforma que seja capaz de realizar o *streaming* de vídeo via *web* a partir das imagens JPEG capturadas pela câmera. Embora não existam plataformas projetadas para reproduzir o *streaming* de vídeo MJPEG a partir da câmera do Raspberry Pi, é possível adaptar um servidor *open source* de vídeo para que este transmita para a rede as imagens capturadas pela câmera.

4.2.1.1 A PLATAFORMA DE *STREAMING*

O *MJPEG-Streamer* é um servidor *open source* de *online streaming*, capaz de transmitir, a partir de uma sequência de imagens, um vídeo ao vivo para um endereço IP determinado. O *MJPEG-Streamer* é escrito na linguagem de programação C, e como tal, é facilmente compilável em qualquer sistema operacional Linux. Reitera-se que os autores da linguagem C foram fundamentais no processo de desenvolvimento dos sistemas operacionais UNIX. Posteriormente, os sistemas operacionais Linux e OSX seriam baseados na arquitetura UNIX, nascendo daí a razão de tais sistemas serem comumente referidos como *UNIX-like systems*. Embora o *MJPEG-Streamer* não tenha sido projetado para este fim, pode-se usar fotografias capturadas continuamente pelo Raspberry Pi para a geração de um *stream* de vídeo MJPEG.

- **O PROCESSO DE COMPILAÇÃO E INSTALAÇÃO**

Para que o servidor de *streaming* MJPEG torne-se compatível com o Raspberry Pi, é preciso que sejam instaladas bibliotecas de edição de vídeo do sistema operacional Linux.

1. *libjpeg8-dev*: Biblioteca necessária para o gerenciamento de imagens JPEG.
2. *imagemagick:* Biblioteca de conversão e edição de imagens.
3. *libv41dev:* Coleção de bibliotecas de geração de vídeo para Linux.

O comando usado para instalação dessas bibliotecas é o seguinte:

```
$ sudo apt-get install libjpeg8-dev imagemagick libv4l-dev
```

Em sequência, pode-se realizar a compilação do código-fonte. Para tanto, basta, a partir do diretório no qual o código-fonte foi salvo, executar o útil comando *make*. Este comando determina automaticamente quais partes do programa necessitam ser compiladas, a partir de um arquivo de diretrizes denominado *makefile*, e realiza a compilação automaticamente. Como o *streaming* de vídeo almejado requer apenas um arquivo de entrada (*input file*) e um endereço IP destino (*output http*), não é necessária a compilação de todo o código-fonte. O comando abaixo é utilizado para compilar apenas os módulos desejados:

```
$ make mjpg_streamer input_file.so output_http.so
```

Após a compilação, pode-se finalmente ligar a câmera e assim realizar a transmissão de vídeo via protocolo MPEG.

- ***STREAMING* MJPEG**

É necessário, em primeiro lugar, configurar a câmera para capturar fotos em formato JPEG e salvá-las em determinado diretório. Neste caso, será criado um diretório próprio para o *streaming*, que será posteriormente usado pelo *MJPG-Streamer*.

```
$ mkdir /tmp/stream
$ raspistill -w 640 -h 480 -q 5 -o /tmp/stream/pic.jpg -tl 100 -t 9999999
-th 0:0:0 &
```

O primeiro comando acima cria o diretório desejado: o diretório **/tmp/stream**. O segundo comando ativa a câmera. Os parâmetros **-w** e **-h** definem a resolução da imagem, que no exemplo é de 640 x 480 *pixels*. O parâmetro **-o** determina o diretório de armazenamento das fotos. O parâmetro **-t** determina a duração de operação da câmera, enquanto o parâmetro **-tl** regula o intervalo de tempo entre as fotos, ou a *frame rate*, sendo um intervalo de 100 milissegundos equivalente a uma *frame rate* de 10 frames por segundo, ou 10 fps. Assim como no caso de transmissão de vídeo via IP, foi escolhido um tempo de operação de 999999 milissegundos, ou 17 horas, o que equivale praticamente a um processo de captura contínua. O parâmetro **-th** em sua configuração **0:0:0** simplesmente desabilita os *thumbnails* das imagens JPEG, para maior leveza de armazenamento. O operador **&** determina que o comando seja executado em *background*, ou plano de fundo. Caso o comando seja executado sem esse operador, não será possível executar outros comandos até que o comando seja finalizado ao término do período especificado, a não ser que se abra outro terminal. É interessante, por motivos de simplicidade, todavia, executar o comando com o operador **&**.

Conforme foi visto anteriormente, o parâmetro **-q** define a qualidade da imagem; maior qualidade resulta em maior queda na *frame rate*. Cabe ao usuário decidir o melhor curso de ação frente ao compromisso entre qualidade e desempenho. O exemplo aqui descrito demonstra as latências obtidas para *streamings* de diferentes níveis de qualidade.

Para o exemplo atual, o vídeo será transmitido a partir do próprio endereço IP do Raspberry Pi, por meio do comando Bash abaixo.

```
$ LD_LIBRARY_PATH=./ ./mjpg_streamer -i "input_file.so -f /tmp/stream -n
pic.jpg" -o "output_http.so -w ./www"
```

O comando LD_LIBRARY_PATH define o diretório dos *plugins* utilizados pelo *MJPG-Streamer*. Por padrão, utiliza-se o próprio diretório de instalação do programa. O marcador **-i** define o *plugin* de entrada, que neste caso trata-se do *plugin* de arquivo de entrada, o *plugin* **input_file.so**. Este

plugin monitora o diretório **/tmp/stream** e, cada vez que uma nova imagem JPEG lá é salva, realiza o *stream* dessa imagem. Os parâmetros **-f** e **-n** determinam o diretório e a imagem a serem monitorados, respectivamente. O marcador **-o** determina o *output plugin*, ou *plugin* de saída, que neste caso é o *plugin* **output_http.so**, que inicia o servidor de vídeo a partir do diretório definido pelo marcador **-w.** Neste exemplo, utiliza-se o diretório **/www**, como diretório de saída. Após a execução dos comandos acima, o servidor será iniciado e bastará acessá-lo com qualquer *browser* para assistir ao *streaming*, por meio do endereço IP do Raspberry e porta 8080: **http://192.168.1.155:8080/?action=stream**. As figuras abaixo ilustram o vídeo em qualidade original capturado pelo servidor e o vídeo comprimido reproduzido no popular *VLC Player* de um *notebook* de sistema operacional Windows 7. O valor de latência, o tempo entre a gravação da imagem e sua reprodução no cliente, também é determinado em cada um dos casos. Observa-se portanto, que há um grande atraso entre a imagem capturada no servidor e a imagem reproduzida no cliente, em decorrência da baixíssima eficiência de compressão do protocolo MJPEG. A Figura 30 ilustra o vídeo original gravado pelo servidor, enquanto as figuras 31, 32, e 33 demonstram o vídeo exibido no cliente em diversas resoluções.

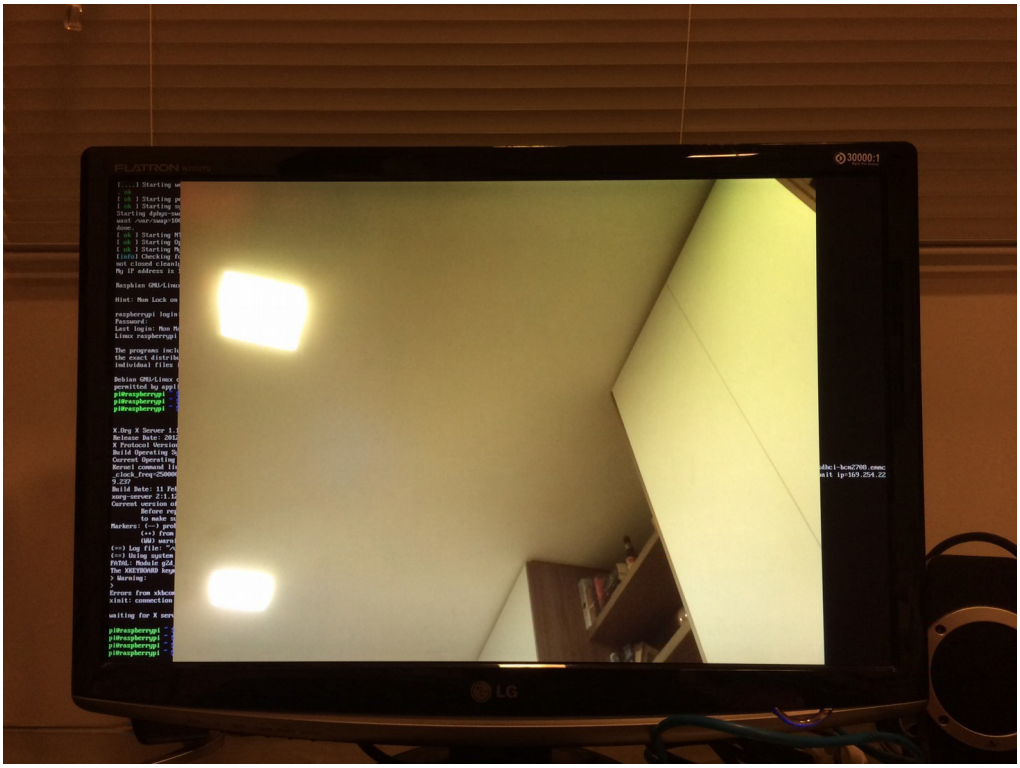

Figura 30: Imagem em qualidade original capturada pela câmera.

Figura 31: Imagem reproduzida no VLC Player do cliente. Resolução de 640 x 480 pixels e parâmetro de qualidade q = 5. Latência de 5 segundos.

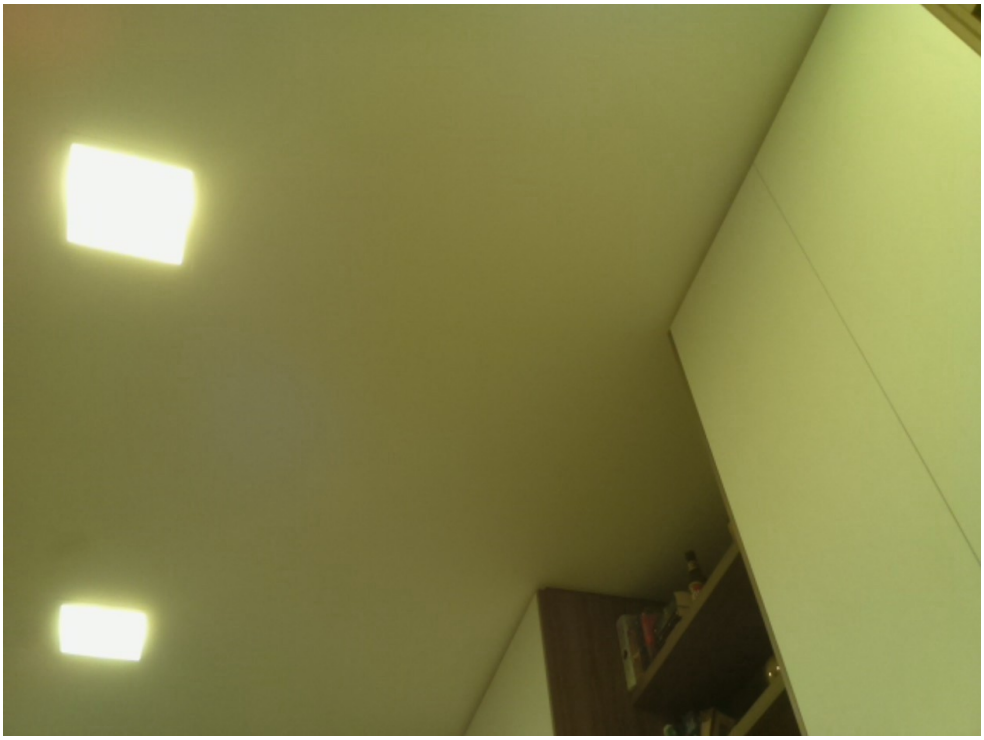

Figura 32: Imagem reproduzida no VLC Player do cliente. Resolução de 640 x 480 pixels e parâmetro de qualidade q = 50. Latência de 10 segundos.

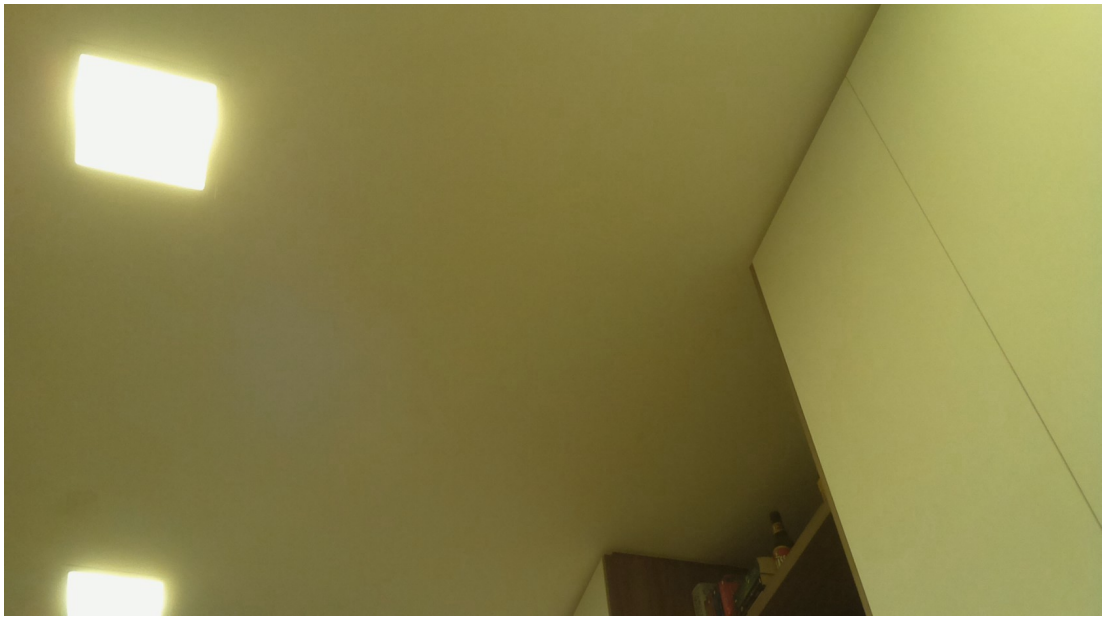

Figura 33: Imagem reproduzida no VLC Player do cliente. Resolução de 1920 x 1080 pixels e parâmetro de qualidade q = 100. Latência de 27 segundos.

4.2.2 STREAMING DE VÍDEO RTSP

O protocolo de *streaming MJPEG*, visto acima, possui as características bastante interessantes de apresentar ampla compatibilidade e permitir o acesso simultâneo de diversos usuários. No entanto, o valor de latência (*lag*) entre o vídeo capturado e o vídeo reproduzido pode ser demasiado alto para aplicações de monitoração de vídeo em tempo real. Naturalmente, esta latência decorre da baixa eficiência de compressão do padrão MJPEG. Seria mais adequado escolher um protocolo que pudesse transmitir o vídeo comprimido em padrão H.264 para múltiplos clientes, visto que este padrão de compressão é extremamente eficiente. Obviamente, esta opção possuiria a limitação de baixa compatibilidade com aparelhos celulares. Sob esta ótica, um protocolo bastante interessante para *streaming* de vídeo é o **RTSP**, ou *Real Time Streaming Protocol*.

O RTSP é um simples, porém amplamente utilizado, protocolo de transmissão de vídeo via rede. Este protocolo permite aos clientes a utilização de simples comandos como *play* e *pause*, para comandar o vídeo do servidor. O vídeo é transmitido via TCP/IP por meio das diretivas Bash descritas abaixo.

```
raspivid -o - -t 0  -w 640 -h 480 -fps 25 | cvlc -vvv stream:///dev/stdin
--sout '#rtp{sdp=rtsp://:8554/}' :demux=h264
```

Onde, conforme visto anteriormente, o parâmetro **-o**, na configuração acima, define a saída de vídeo no diretório */dev/stdout*. O parâmetro **-t** define a gravação contínua de vídeo, os parâmetros **-w** e **-h** definem a resolução do vídeo transmitido, e o parâmetro **fps** define a taxa de quadros por segundo. O operador *pipe* direciona a saída do sinal de vídeo da câmera para o comando **cvlc**, que realizará o *streaming* RTSP, enquanto o parâmetro **-vvv** define a localização do diretório da gravação gerada pela câmera. O parâmetro **sout** define a localização da saída de vídeo do *streaming*. Como estamos interessados no protocolo RTSP, definimos a saída conforme no comando acima e especificamos, arbitrariamente, a porta 8554. O parâmetro **demux=h264** é de fundamental importância, pois configura o cliente para decodificar corretamente o *transport stream* enviado pelo Raspberry Pi.

O vídeo pode ser visualizado por um reprodutor de vídeo *VLC Player* instalado no cliente, ou qualquer outro *player* de vídeo capaz de reproduzir *streaming* RTSP. A figura abaixo ilustra a reprodução de um vídeo transmitido via RTSP. A resolução do vídeo é de 640 x 480 *pixels,* e a taxa de quadros por segundo, ou *fps*, é de 25. A latência neste caso, é de apenas 1,5 segundo, cerca de 3 vezes menor que a latência do protocolo MJPEG. A fluidez da imagem é notadamente superior, pois conta com o padrão de compressão H.264, que é muito mais eficiente que o padrão JPEG. A latência para resoluções maiores cresce lentamente, atingindo 2 segundos a uma resolução de 800 x 600 *pixels*, e 2,4 segundos a uma resolução de 1280 x 720 *pixels*. A transmissão a uma resolução *Full-HD* de 1920 x 1080 *pixels* mostrou-se demasiadamente instável, com diversos erros de compressão em decorrência da alta taxa de transmissão a uma altíssima resolução. Tais erros se dão em razão da limitação do vetor de movimento do padrão de compressão MPEG, e ocorrem proporcionalmente à quantidade de movimento do vídeo. O *streaming* em resolução *Full HD* requer uma altíssima *bitrate*, e a introdução de movimentos na imagem gera uma entropia grande o suficiente para que o vetor de movimento perca a capacidade de emular a diferença de informação entre duas imagens sucessivas. Percebe-se, no entanto, pelas imagens abaixo, que um *streaming* de resolução 1280 x 720 produz uma qualidade perfeitamente aceitável para a maioria das aplicações de monitoração.

Figura 34: Imagem transmitida via RTSP. A resolução é de 640 x 480 pixels a uma taxa de 25 quadros por segundo. A latência é de 1,5 segundo.

Figura 35: Transmissão a uma resolução de 800 x 600 pixels a 25 quadros por segundo. A latência é de 2 segundos.

Figura 36: Transmissão a uma resolução de 1280 x 720 pixels a 25 quadros por segundo. A latência é de apenas 2,4 segundos.

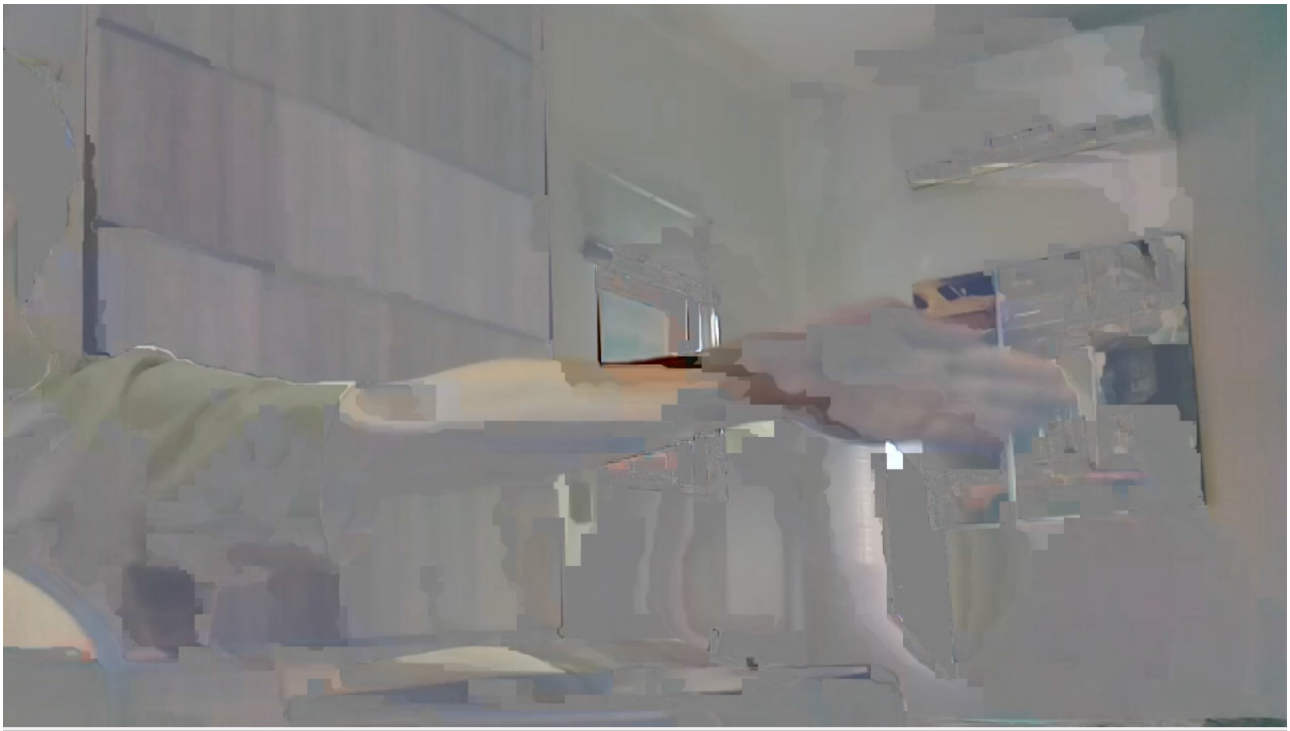

Figura 37: Erro de compressão a uma resolução de 1920 x 1080 pixels. Conforme visto anteriormente, o vetor de movimento do padrão MPEG não é capaz de emular perfeitamente a diferença de informação entre duas imagens sucessivas.

A Figura 38 relaciona o desempenho de cada um dos protocolos adotados. Para o protocolo MJPEG, foram consideradas fotografias com fator de qualidade q = 50. Observa-se que o protocolo RTSP possui desempenho bastante próximo ao do *stream* direto, e que o protocolo MJPEG torna-se praticamente inviável em alta definição, devido à alta latência.

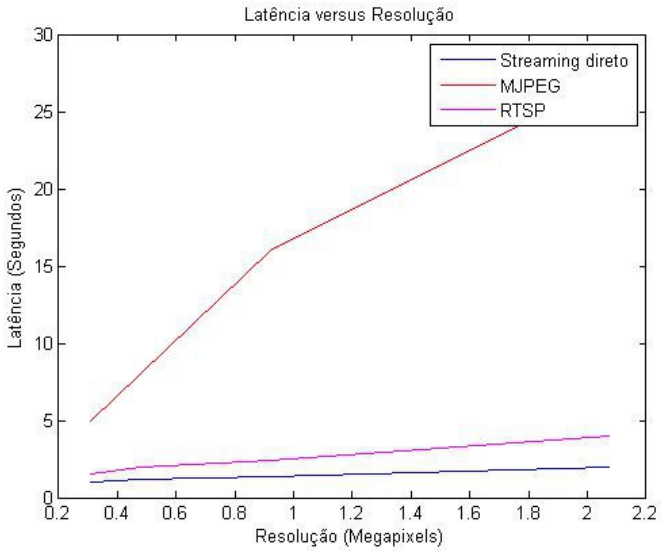

Figura 38: Desempenho dos três protocolos adotados. Nota-se que a latência do protocolo MJPEG aumenta substancialmente em resoluções maiores.

5. INTERFACEAMENTO

5.1 ACIONAMENTO DA CÂMERA VIA RADIOFREQUÊNCIA

Após a configuração da central de monitoração de vídeo, torna-se bastante interessante encontrar uma maneira de acionar e parar a gravação sem a necessidade de digitar os comandos diretamente no Raspberry Pi. Um dispositivo bastante adequado para sanar este problema é o módulo *transceiver* de radiofrequência da empresa norte-americana *Addicore,* mostrado na Figura 39.

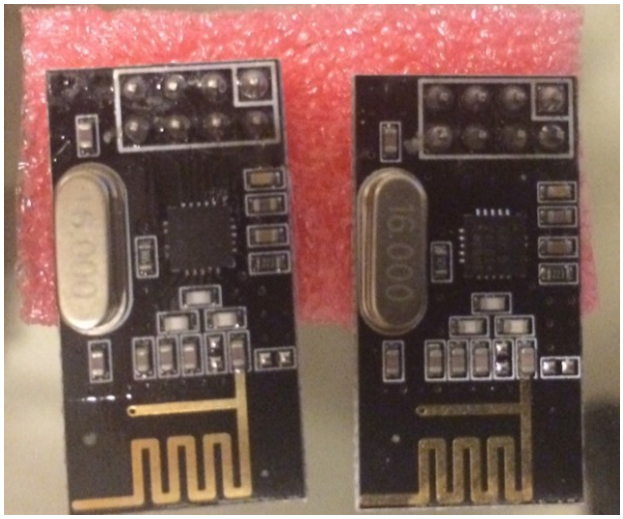

Figura 39: Transceivers Addicore.

Estes pequenos dispositivos possuem a capacidade de transmitir sinais a uma frequência de 2,4 GHz por mais de 80 metros. Os módulos são configuráveis através da interface de quatro pinos SPI, tendo como parâmetros configuráveis: canal de frequência (125 canais disponíveis), potência de transmissão, e *bitrate,* este com três opções possíveis: 250 *kbps*, 1 *Mbps* e 2 *Mbps* [25]. Felizmente, o Raspberry oferece duas portas SPI, como foi visto acima. Consequentemente, pode-se conectar um *transceiver* aos pinos de interface GPI do Raspberry para que este atue como um receptor do sinal de RF que ativará a gravação de vídeo.

Naturalmente, será necessário um microcontrolador adicional para controlar o módulo transmissor. Nesta aplicação, será usado o microcontrolador *Arduino UNO*, que, assim como o Raspberry Pi, é compatível com o protocolo de comunicação SPI. A partir de um circuito básico consistindo apenas em um botão (*push button*) e o módulo transmissor, o *Arduino* poderá atuar como o transmissor responsável por iniciar a gravação de vídeo. A Figura 40 ilustra o microcontrolador *Arduino UNO*.

Figura 40: Microcontrolador Arduino UNO.

Para que os módulos possam ser corretamente configurados, é necessário primeiramente explicitar as funções de cada um dos pinos de conexão do *transceiver,* mostrados na Figura 41. As funções desses pinos são descritas na Tabela 3.

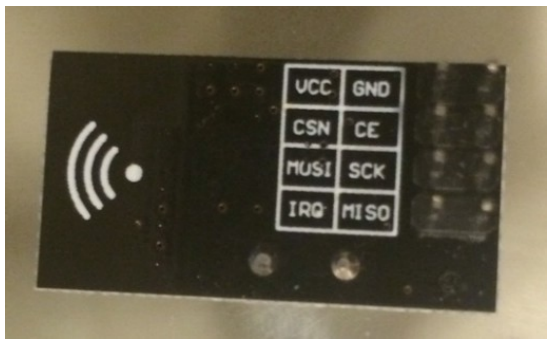

Figura 41: Pinos de conexão do módulo transceiver.

Tabela 3: Funções dos pinos dos módulos transceiver.

PINO	**FUNÇÃO**
VCC	Alimentação: 1,9 a 3,6 Volts
GND	Alimentação: GND
CSN	*Chip Select:* corresponde ao sinal *Slave Select* do protocolo SPI
CE	*Chip Enable*: Ativa o modo TX (transmissor) ou RX (receptor)
MOSI	*Slave data input*

SCK	*Clock* SPI
IRQ	*Interrupt Request*
MISO	*Slave data output*

Para que se possa compreender mais a fundo as funções descritas na Tabela 3, é necessário o conhecimento básico do protocolo de comunicação SPI. É dada, portanto, a seguir, uma breve revisão dos conceitos fundamentais do protocolo SPI.

5.1.1 O PROTOCOLO SPI

O protocolo SPI é um protocolo de comunicação serial síncrono desenvolvido para comunicações de curta distância entre sistemas embarcados. A comunicação ocorre entre o dispositivo mestre, ou *master*, e o dispositivo escravo, ou *slave*. O SPI permite a conexão de múltiplos *slaves* ao barramento, mas apenas um *master* pode existir na comunicação. O mestre é responsável por: gerar requisição de escrita ou leitura, selecionar o *slave* adequado por meio da linha *slave select* ou *chip select*, e gerar o sinal de *clock* para que a comunicação síncrona seja possível. O protocolo SPI permite a comunicação *full duplex* entre *slaves* e *master*. A Figura 42 esquematiza as ligações do protocolo SPI.

Os 4 sinais de lógica utilizados no protocolo SPI são:

► ***SCLK***: *Serial Clock,* correspondente ao sinal de *clock* gerado pelo mestre

► ***MOSI:*** *Master Output Slave Input,* correspondente ao sinal de escrita enviado pelo mestre

► ***MISO*****:** *Master Input Slave Output*, correspondente ao sinal de leitura enviado pelo mestre

► ***SS:*** *Slave Select*, correspondente ao sinal de seleção de *slave* enviado pelo mestre

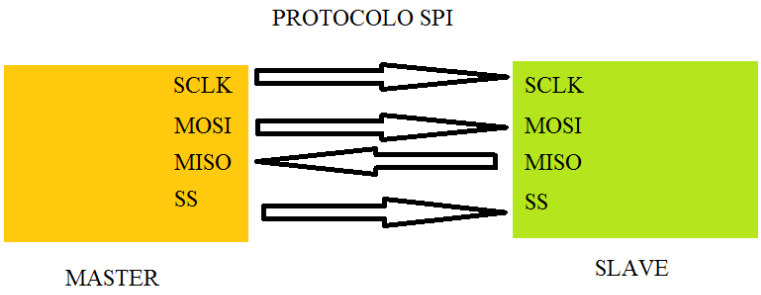

Figura 42: Protocolo SPI

5.1.2 CIRCUITO DE COMUNICAÇÃO VIA RADIOFREQUÊNCIA

O microcontrolador *Arduino Uno* possui os quatro pinos digitais necessários para o protocolo SPI, sendo eles:

- PINO 10: **SS**
- PINO 11: **MOSI**
- PINO 12: **MISO**
- PINO 13: **SCLK**

O *Arduino* também possui os pinos de alimentação necessários para o correto funcionamento do *transceiver*. É necessário, no entanto, atentar-se ao fato de que o *transceiver* deve ser alimentado pelo pino de tensão de 3,3 Volts. A conexão do módulo ao pino de 5 Volts poderá ocasionar defeito do dispositivo.

Deverá ser escolhido um pino digital adicional arbitrário para realizar a função de *Chip Enable*, que elegerá transmissor o módulo conectado ao *Arduino.* Para tanto, deve-se gerar uma tensão lógica HIGH no pino conectado ao *Chip Enable* do *transceiver*. A relação entre os pinos do *transceiver* e os pinos do *Arduino* é descrita na Tabela 4.

Tabela 4: Relação entre pinos do microcontrolador e pinos do transceiver.

TRANSCEIVER	**ARDUINO**
VCC	3,3V
GND	GND
CE	DIGITAL 9 (pino digital arbitrário)
CSN	DIGITAL 10
MOSI	DIGITAL 11
MISO	DIGITAL 12
SCK	DIGITAL 13

O *Arduino* possui uma interface gráfica de programação, cuja linguagem é bastante semelhante à linguagem C. Por meio de uma conexão USB com o computador, pode-se facilmente programar o *Arduino*. O fabricante do microcontrolador disponibiliza uma biblioteca de comando do módulo de radiofrequência, para que este possa ser programado por meio da linguagem do *Arduino* [11]. A partir de um simples botão conectado a um resistor *pull-up* e a qualquer pino digital do microcontrolador, pode-se escrever um código para que o transmissor seja acionado sempre que o botão for pressionado.

Uma vez que o módulo transmissor esteja corretamente configurado, é necessário configurar o módulo receptor, conectado ao Raspberry Pi. A Tabela 5 indica a relação entre os pinos do módulo receptor e os pinos GPI do Raspberry Pi. A Figura 44 ilustra a diagramação das conexões entre o *transceiver* e o Raspberry Pi.

Tabela 5: Relação entre pinos do módulo receptor e pinos GPI do Raspberry Pi.

TRANSCEIVER	**GPI RASPBERRY**
VCC	3V3
GND	GND
CE	BCM 25 (pino digital arbitrário)
CSN	CE0 (BCM 8)
MOSI	MOSI (BCM 10)
MISO	MISO (BCM 9)
SCLK	SCLK (BCM 11)

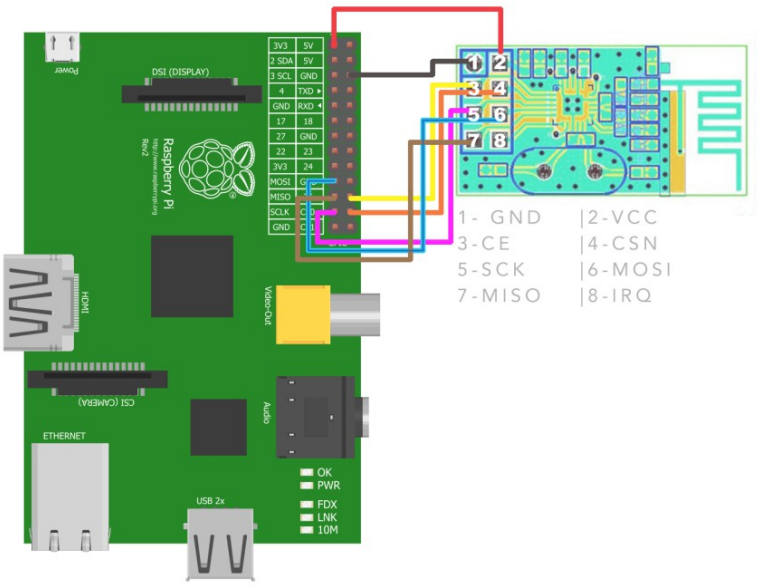

Figura 43: Diagramação das conexões entre o módulo receptor e o Raspberry Pi [12].

Uma vez completas as conexões, pode-se elaborar um *script* na linguagem de programação *Python*, para que o Raspberry monitore o sinal recebido pelo módulo receptor, e, a partir da recepção do sinal, iniciar o protocolo de *streaming* desejado. É pertinente ressaltar que o exemplo acima descrito é uma simples aplicação do conjunto de módulos *transceivers*. É possível, por meio de um código mais complexo, adicionar outras funções ao sistema, como, por exemplo: reiniciar o Raspberry Pi, ou manter uma comunicação constante entre *Arduino* e Raspberry Pi, para que o transmissor possa tomar ações adicionais a partir do estado do receptor.

5.2 MONITORAÇÃO DO ESTADO DA CÂMERA

É bastante interessante possuir a opção de monitoração do estado da câmera sem a necessidade de conectar o Raspberry Pi a um monitor de vídeo. Pode-se usar um pequeno *display* de cristal líquido (LCD) para exibir o estado da câmera. O *display* escolhido para este exemplo é popularmente conhecido como *display* 16 x 2, e é mostrado na Figura 45. Observa-se na diagramação que o *display* conta com 16 pinos, cujas funções são detalhadas na Tabela 6. A conexão do *display* com o Raspberry Pi é esquematizada na Figura 46.

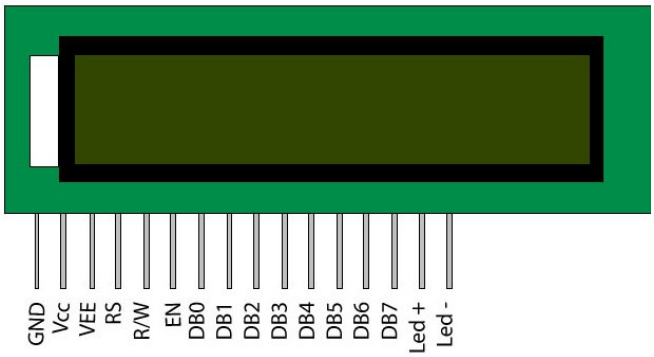

Figura 44: Display LCD 16 x 2 e denominação de seus pinos [13].

Tabela 6: Pinos do display 16 x 2.

PINO	**FUNÇÃO**
GND	Terra
VCC	Tensão de alimentação (5 Volts)
VEE	Ajuste de contraste
RS	*Register select*: selecionar registro de comando ou registro de dados
R/W	*Read/Write*: Ler ou escrever dados no *display*
EN	*Enable*: Envia dados aos pinos quando há transição de tensão HIGH para LOW.

DB[0..7]	Pinos de dados: o *display* pode receber dados paralelamente em formato 4-bit (apenas 4 pinos de dados são usados) ou 8-bit (todos os pinos de dados são usados)
LED +	Tensão de alimentação do LED de iluminação do *display*
LED -	Terra do LED de iluminação do display

O *display* 16x2 comumente possui um controlador guiado pelo *driver* Hitachi HD 44780, para que o *display* possa ser programado por intermédio de comandos de acionamento de funções. Neste exemplo, será usada uma biblioteca na linguagem de programação *Python*, que possibilitará ao usuário o controle das funções descritas na Tabela 7. O *display* possui duas memórias de acesso aleatório (RAM) internas:

- CGRAM: *Character Generation RAM*, responsável por armazenar o padrão dos caracteres exibidos no *display*.
- DDRAM: *Display RAM*, responsável por armazenar em cada posição os caracteres exibidos no *display*.

A partir das instruções descritas na Tabela 7, pode-se programar uma biblioteca na linguagem *Python,* ou em qualquer outra linguagem de alto nível, para que o *display* possa ser controlado a partir de simples comandos. Neste exemplo, foi elaborada uma biblioteca de controle do LCD em *Python.* Subsequentemente, foi programado um *script*, também em *Python*, para que o *display* comporte-se da seguinte maneira:

- Exibir a palavra OCIOSO, quando a câmera estiver desligada.
- Exibir a palavra GRAVANDO, quando a câmera estiver ativa.

Naturalmente, este exemplo ilustra uma simples implementação do *display* 16x2. Poder-se-ia, por exemplo, configurar o *display* para exibir hora e data, endereço IP do servidor, ou total de horas de gravação. A partir da biblioteca de controle do *display*, sua funcionalidade fica inteiramente a critério do usuário. As Figuras 47 e 48 exibem o *display* monitorando o estado da câmera.

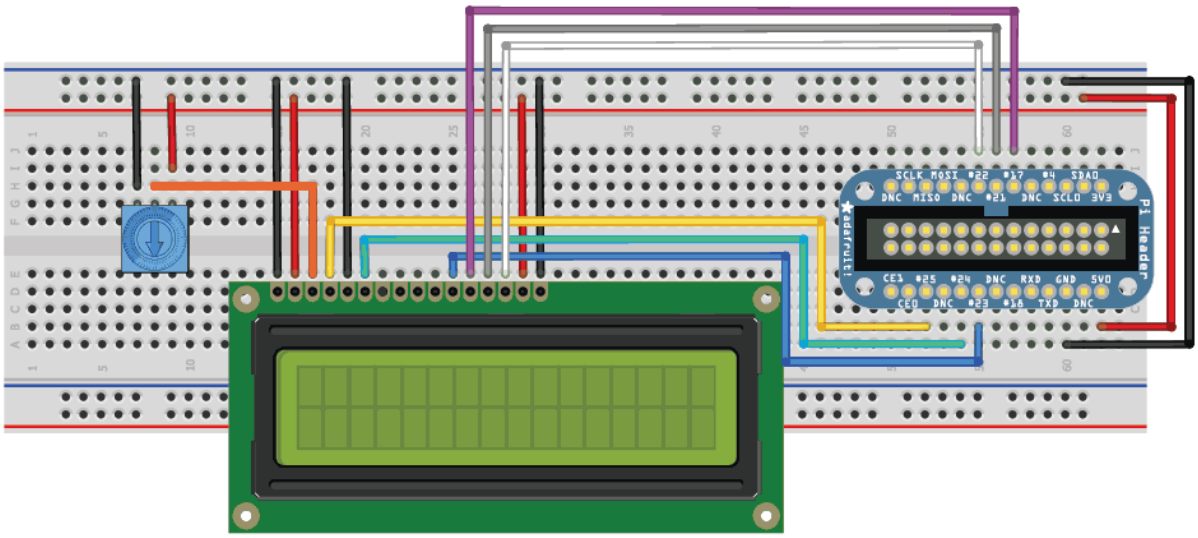

Figura 45: Diagramação da conexão do display 16x2 com o Raspberry Pi. O potenciômetro é utilizado para ajuste de contraste. A base de pinos azul à direita é um extensor da interface GPI do Raspberry Pi, que possibilita a conexão via protoboard [14].

Tabela 7: Instruções do display 16 x 2.

INSTRUÇÃO	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Limpar *display*	0	0	0	0	0	0	0	0	0	1
Retornar cursor à posição inicial	0	0	0	0	0	0	0	0	1	X
Determinar direção de movimento do cursor. Direção determinada pelo bit R/L. O bit S habilita o *shift* dos caracteres no *display*	0	0	0	0	0	0	0	0	R/L	S
Controle *on/off*. O bit D liga ou desliga o *display*, o bit C liga ou desliga o cursor, e o bit B habilita o cursor piscante	0	0	0	0	0	0	1	D	C	B
Cursor/Display Shift: bit S/C determina movimento do cursor ou do *display*, e bit R/L determina direção do movimento	0	0	0	0	0	1	S/C	R/L	X	X
Function set: bit DL determina o comprimento de dados (4-bit ou 8-bit), bit N determina o número da linha, e bit F determina a fonte dos caracteres	0	0	0	0	1	DL	N	F	X	X
Configurar endereço da memória CGRAM, para envio de dados ao *display*.	0	0	0	1	A5	A4	A3	A2	A1	A0
Configurar endereço da memória DDRAM, para envio de dados ao *display*.	0	0	1	A6	A5	A4	A3	A2	A1	A0

Ler *busy-flag*, indicativo de que o *display* está efetuando operações, e conteúdo do endereço da memória DDRAM.	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Salvar dados na memória DDRAM ou na memória CGRAM, de acordo com o endereço configurado anteriormente	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Ler dados da memória DDRAM ou da memória CGRAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

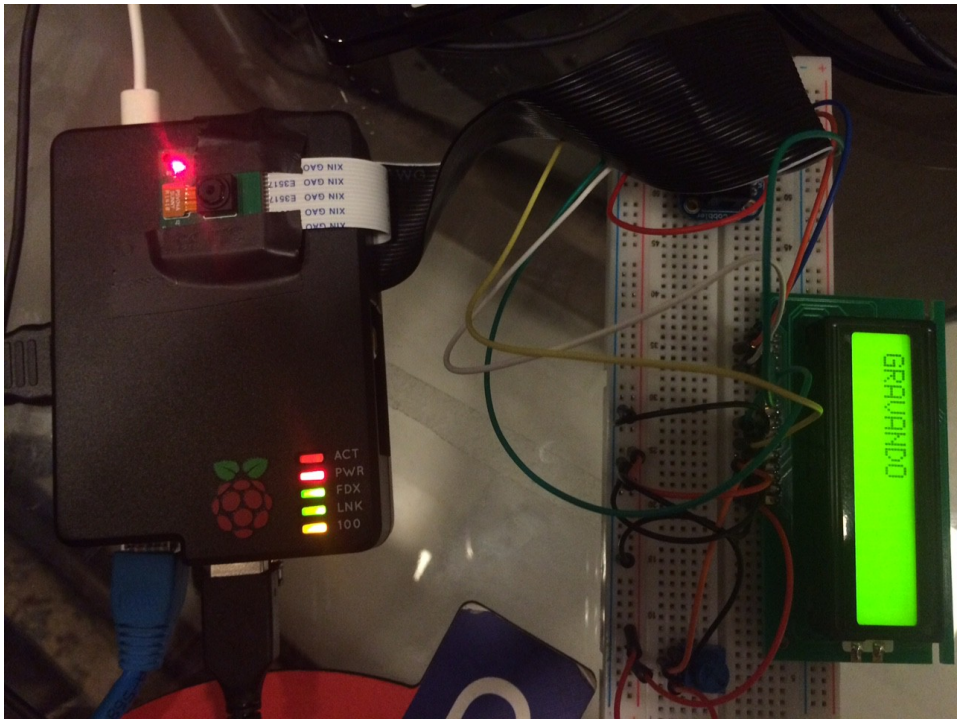

Figura 46: Display LCD exibindo o estado "GRAVANDO" durante a gravação de vídeo.

Figura 47: Display exibe o estado "OCIOSO" quando não há gravação de vídeo.

6. CONCLUSÕES

Observou-se que uma central de monitoração pôde ser implementada com êxito por meio do Raspberry Pi. Este pequeno microcontrolador tem grande flexibilidade em aplicações de sistemas embarcados, e uma central de monitoração de vídeo representa um único exemplo dessas aplicações. O desempenho obtido variou de acordo com o protocolo obtido, sendo relatados os seguintes resultados.

O *streaming* de vídeo direto, entre Raspberry Pi e cliente, apresenta baixa latência mas esbarra em limitações práticas. Apenas um cliente pode visualizar o vídeo, e o *streaming* é finalizado sempre que o cliente encerra a visualização do vídeo. Esta aplicação, por conseguinte, é indicada apenas para casos bastantes específicos, onde há a necessidade ocasional de monitoração por curto período de tempo.

O *streaming* de vídeo via protocolo MJPEG é indicado quando o usuário necessita de ampla compatibilidade, no caso de o vídeo ser visualizado simultaneamente por diversos dispositivos de diferentes fabricantes. A alta latência encontrada neste protocolo decorre de sua baixa eficiência de transmissão; cada *frame* de vídeo consiste em uma imagem JPEG completa. Este protocolo é inadequado, portanto, em aplicações onde haja a exigência de monitoração de vídeo muito próxima ao tempo real.

O *streaming* de vídeo via protocolo RTSP é a opção mais versátil, sendo indicada para a maioria de aplicações de monitoração de vídeo que não requeiram a visualização por aparelhos celulares. Apresenta baixa latência e permite a visualização de vídeo por múltiplos clientes simultaneamente. Ademais, existe a possibilidade de ajuste de qualidade de acordo com o desejo do usuário. O protocolo RTSP apresentou desempenho bastante satisfatório na maioria das resoluções. As limitações do protocolo surgiram em transmissões de alta definição com grande quantidade de movimento no vídeo. Indica-se uma resolução de vídeo de até 800 x 600 *pixels* em ambientes de grande movimento. Caso o ambiente monitorado seja relativamente estático, é possível configurar a transmissão para uma resolução de alta definição de 1280 x 720 *pixels*. A transmissão em 1920 x 1080 *pixels*, chamada transmissão *Full HD*, mostrou-se demasiadamente instável para monitoração de vídeo.

O interfaceamento com os módulos *transceivers* foi implementado com sucesso. Naturalmente, o exemplo aqui descrito demonstrou apenas uma simples aplicação de comunicação entre os dois módulos para ativação da câmera. No entanto, os *transceivers*, por intermédio do protocolo SPI, podem comunicar-se de maneira muito mais complexa e rápida. Poder-se-ia, por exemplo, monitorar o estado da câmera a partir do módulo transmissor, com um *display* 16x2 conectado ao *Arduino UNO*.

A mesma ressalva se aplica ao exemplo de monitoração do estado da câmera por meio de um *display* conectado ao Raspberry Pi. Adaptando-se o *script* de controle do *display*, com a simples e didática linguagem de programação *Python*, é possível configurar o *display* para exibir diversas

informações, relativas ou não à câmera de vídeo. Seria possível, por exemplo, configurar o *display* para que este exibisse a carga remanescente da bateria, caso o Raspberry Pi estivesse monitorando algum ambiente remoto sem a existência de tomadas.

Como ideia de projeto futuro, sugere-se primeiramente uma central de monitoramento de vídeo com múltiplos microcontroladores Raspberry Pi, de forma que se tenha acesso simultâneo a diversas câmeras. Seria interessante o desenvolvimento de uma interface *web* para que o intercâmbio entre as câmeras fosse facilitado. O Raspberry Pi pode também ser usado como servidor de vídeo, tendo a capacidade de armazenar o vídeo gravado em seu próprio cartão de memória. Assim sendo, é possível configurar um *cluster* de armazenamento composto por várias unidades do Raspberry Pi, e acessar o material gravado nos cartões via rede. As diversas opções de interfaceamento dão margem a inúmeras possibilidades de melhoria da central. Pode-se sugerir, por exemplo, a implementação de um sistema de detecção de movimento, onde sensores conectados à interface GPI do Raspberry ativam a gravação sempre que detectam qualquer tipo de movimento. É fácil perceber que a interface GPI permite ao projetista um grande grau de criatividade.

Reitera-se, ao fim deste trabalho, que o objetivo do autor não é implementar um produto comercialmente viável, embora o custo da central aqui demonstrada seja menor que as correntes opções de mercado. O motivo ulterior deste projeto é atiçar a curiosidade de engenheiros e entusiastas da engenharia para que novas criações práticas surjam a partir de microcontroladores programáveis de baixo custo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]. *Raspberry Pi Documentation*., https://www.raspberrypi.org/documentation/, Acesso 14/04/2015

[2]. *Master Thesis in Computer Engineering.,* http://hh.diva-portal.org/smash/get/diva2:737364/FULLTEXT01.pdf – Acesso 07/06/2015

[3]. *Compute Module.*, https://www.raspberrypi.org/products/compute-module-development-kit/ - Acesso 01/06/2015

[4]. *ARM Processors.*, http://www.arm.com/products/processors/classic/arm11/arm1176.php Acesso 28/05/2015

[5]. *Camera Module*., https://www.raspberrypi.org/products/camera-module/ - Acesso 02/06/2015

[6]. *Raspberry Pi Camera Board.*, https://www.raspberrypi.org/camera-board-available-for-sale/ - Acesso 05/05/2015

[7]. *Raspberry Pi GPIO.*, https://www.raspberrypi.org/documentation/usage/gpio/ - Acesso 02/06/2015

[8]. *Raspberry Pi Pinou*t., http://pi.gadgetoid.com/pinout – Acesso 06/06/2015

[9]. TEKTRONIX ARCHIVES., *A Guide to Standard and High-Definition Video Measurements.*

[10]. TEKTRONIX ARCHIVES., *A Guide to MPEG Fundamentals and Protocol Analysis.*

[11]. *Arduino nRF24L01 Libraries.,* http://playground.arduino.cc/InterfacingWithHardware/Nrf24L01 – Acesso 06/06/2015

[12]. *Arduino Switching Circuit with nRF24l01.,* http://hack.lenotta.com/arduino-raspberry-pi-switching-light-with-nrf24l01/ - Acesso 06/06/2015

[13]. *16X2 LCD Module Datasheet.,* http://www.engineersgarage.com/electronic-components/16x2-lcd-module-datasheet – Acesso 06/06/2015

[14]. *Wiring the Cobbler to the LCD.,* https://learn.adafruit.com/assets/1729 – Acesso 07/06/2015

[15]. *MJPEG Streamer.*, http://sourceforge.net/projects/mjpg-streamer/ - Acesso 11/05/2015

[16]. *Debian Video Libraries.*, https://packages.debian.org/sid/libjpeg8-dev – Acesso 11/05/2015

[17]. *Image Magick.,* http://www.imagemagick.org/script/index.php – Acesso 11/05/2015

[18]. *Libv41 Video Library.*, http://packages.ubuntu.com/lucid/libv4l-dev - Acesso 11/05/2015

[19]. *Embedded Linux Wiki.*, http://elinux.org/, Acesso 21/04/2015

[20]. *VideoCore GPU.*, http://www.broadcom.com/docs/support/videocore/VideoCoreIV-AG100-R.pdf Acesso 28/05/2015

[21]. *ARM Processors documentation.*, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/index.html Acesso 01/06/2015

[22]. *MPEG Codec.*, http://www.itu.int/rec/T-REC-H.262 – Acesso 01/06/2015

[23]. *RTSP.*, https://www.ietf.org/rfc/rfc2326.txt – Acesso 01/06/2015

[24]. WIDLAR, Robert., *Introduction to Semiconductor Devices, 1960.*

[25]. *MJPEG Protocol and Raspberry Pi*., http://ozzmaker.com/2012/12/03/send-email-from-the-raspberry-pi-or-linux-command-line-with-attachments/ - Acesso 24/04/2015

[26]. JAIN, Anil K.*, Fundamentals of Digital Image Processing, Englewood Cliffs, NJ, Prentice Hall, 1989, pp. 150-153.*

[27]. *Spatial Frequency of Images.*, http://www.haberdar.org/Spatial-Frequency-of-an-Image-Tutorial.htm – Acesso 02/06/2015

[28]. *Addicore Transceiver.*, http://www.addicore.com/2pcs-Addicore-nRF24L01-Wireless-Transceiver-p/112.htm – Acesso 06/06/2015

[29]. TEKTRONIX ARCHIVES., *Understanding Colors and Gamut.*

[30]. HOROWITZ, Paul. HILL, Winfield., *The Art of Electronics, Cambridge University Press, 2nd edition.*

[31]. *Mplayer.*, http://www.mplayerhq.hu/design7/info.html – Acesso 07/05/2015

[32]. *Netcat.*, http://netcat.sourceforge.net/ - Acesso 07/05/2015

[33]. *Introduction to BASH.*, http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-4.html – Acesso 07/05/2015

[34]. ROSCH, L. Winn., *The Winn L. Rosch Hardware Bible, QUE Publishing, 6th edition.*