UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RAFFAELLO SALVETTI SANTOS

CONTROLE DO MOVIMENTO DE CÂMERA COM BASE EM SENSORES DE POSIÇÃO DE UM SMARTFONE

SALVADOR/BA
2019

RAFFAELLO SALVETTI SANTOS

CONTROLE DO MOVIMENTO DE CÂMERA COM BASE EM SENSORES DE POSIÇÃO DE UM SMARTFONE

Trabalho de Conclusão de Curso, apresentado como parte dos requisito para a obtenção de grau de Bacharel em Engenharia de Computação, pela Universidade Federal da Bahia.

Orientador: Prof. Paulo César Machado de Abreu Farias
Universidade Federal da Bahia

SALVADOR/BA
2019

RAFFAELLO SALVETTI SANTOS

CONTROLE DO MOVIMENTO DE CÂMERA COM BASE EM SENSORES DE POSIÇÃO DE UM SMARTFONE

Trabalho de Conclusão de Curso, apresentado como parte dos requisito para a obtenção de grau de Bacharel em Engenharia de Computação, pela Universidade Federal da Bahia.

DATA DE APROVAÇÃO:

CONCEITO:

Prof. Paulo César Machado de Abreu Farias
Orientador - UFBA

Prof. 1
Membro - UFBA

Prof. 2
Membro - UFBA

SALVADOR/BA
2019

Dedico a...

AGRADECIMENTOS

Agradeço a...

"Se o conhecimento traz problemas, não é a ignorância que os resolve."(Isaac Asimov)

RESUMO

Sistemas robóticos fazem parte da história humana. Seja na vida pratica, nos dias atuais, ou na peça teatral de ficção cientifica *Rossumovi Univerzální Roboti*, criada em 1920, onde o termo "ROBÔ"foi utilizado pela primeira vez, antes mesmo do primeiro computador eletrônico ser construído. Atualmente, a robótica é usada nos mais variados setores. Nos domicílios é usado como auxiliar de limpeza doméstica (na forma de aspiradores de pó), em alguns países são utilizados como seguranças patrimoniais, na medicina auxiliam em cirurgias e no espaço exploram mundos desconhecidos. Usando elementos de robótica, este trabalho pretende usar dados de sensores de posição de um smartphone, para criar um sistema de controle de movimento para câmera que pode ser embarcada num robô, fornecendo para o operador uma interface mais intuitiva. São apresentados as características do projeto, montagem do protótipo, programação dos softwares de controle de câmera e envio de dados dos sensores.

Palavras-chave: Robótica, Movimento, Controle, Automação, Câmera.

ABSTRACT

Write your abstract here!!!

Keywords: Keywords 1. Keywords 2. Keywords 3. Keywords 4.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BET	Teoria do Elemento de Pá (*Blade Element Theory*)
THEV	Turbina Hidrocinética de Eixo Vertical
THEH	Turbina Hidrocinética de Eixo Horizontal ...

LISTA DE SÍMBOLOS

Γ	Letra grega Gama
λ	Comprimento de onda
$\in$	Pertence ...

Sumário

1 INTRODUÇÃO

Dutos de ventilação usados nos sistemas de ar-condicionado estão sujeitos a diversos tipos de danos, dentre os quais pode-se listar os entupimentos progressivos devido ao acúmulo de poeira e de pequenos animais mortos. (CARMO; PRADO, 1999) Por normalmente ser locais de difícil acesso, apresentam dificuldades em sua manutenção, favorecendo a proliferação de bactérias e transmissão de vírus (BORTOLETTO et al., 2002).

Subestações de energia elétrica, em grande parte das vezes, ficam expostas a intempéries, que causam oxidações em suportes, equipamentos e cabos. Sua inspeção oferece riscos a vida por expor o corpo humano a uma quantidade enorme de energia. Apesar de existir uma norma rigorosa para a realização de inspeções preventivas, acidentes com vítima ainda acontecem (SANTOS et al., 2012).

A inspeção de reservatórios de produtos químicos requer uma minuciosa análise estrutural, uma busca por áreas oxidadas e falhas em pontos de solda, tarefa que demanda muitas horas de trabalho humano. A exposição a gases e vapores tóxicos, por menor que seja a quantidade, causam riscos a saúde do inspetor (MOLINA et al., 2008).

Os casos citados, são apenas algumas das atividades extremamente necessárias no ambiente industrial, que expõem a saúde das pessoas a riscos que poderiam ser evitados, através do uso de dispositivos especializados.

Robôs equipados com ferramentas adequadas para cada tarefa, dotados de um sistema de navegação autônomo ou por controle remoto, podem ser usados para evitar ou minimizar os riscos a saúde dos inspetores. Como exemplo, o robô *SENSABOT*, usado pela Shell, para monitorar e inspecionar plantas de óleo e gás, mostrado na Figura 1.1a, e o robô *ABB*, usado em inspeções de transformadores, sem a necessidade de drenar o óleo, visto na Figura 1.1b. Contudo o uso de robôs não se restringe apenas ao ambiente industrial. No ano de 2011, robôs submarinos foram protagonistas na localização e resgate das peças do avião da Air France 447, que caiu no Oceano Atlântico em 2009. Foram usados robôs de inspeção, para verificar as condições estruturais da usina nuclear de Fukushima Daiichi no Japão, que foi afetada por um tsunami.

(a) SENSABOT

Fonte: (TRACTICA.COM, 2016).

(b) ABB

Fonte: (PROCESSONLINE.COM.AU, 2019).

Figura 1.1 – Robôs de inspeção

Independente de um robô ser controlado diretamente por uma pessoa, ou operar em modo completamente autônomo, em algum momento, a observação, supervisão e o julgamento humano ainda é um elemento crítico da atividade robótica.

A maior ligação perceptual entre um ambiente remoto e o operador de um robô, acontece através do vídeo enviado por uma (ou mais) câmera montada no robô. Existe uma relação forte entre problemas de localização da câmera e sua montagem, bem como angulo de visão e outros fatores, que podem degradar essa ligação e deixar o operador vulnerável a uma serie de erros como: desorientação, falha ao reconhecer danos ou simplesmente não notar um ponto importante durante uma inspeção.

Estudos sugerem que disponibilizar uma câmera controlada independentemente da orientação de um robô, pode facilitar tarefas de localização num ambiente. (HUGHES; LEWIS, 2004)

A complexidade de operação de um robô é proporcional ao seu DOF (*Degrees of Freedom*), isto é, quanto maior a variedade de movimentos o robô pode executar (considerando seu deslocamento, movimento de braços mecânicos e câmera), mais difícil é o controle manual para o operador. Pensando nisso, o desenvolvimento de controles mais intuitivos para a câmera embarcada, diminui a complexidade geral do controle de movimentos globais do robô.

Com o objetivo de aprofundar os conhecimentos adquiridos ao longo do curso, este trabalho visa criar um controle de câmera do tipo *Pan* e *Tilt*, baseando-se nos dados de movimento, coletados e enviados, através de rede sem fio, por um *smartphone* acoplado a cabeça do operador, por um óculos VR (*Virtual Reality*). Dessa forma, os movimentos de rotação da cabeça do operador (*yaw* e *pitch*) são traduzidos em movimentos da câmera (*Pan* e *Tilt*), montada no robô.

No projeto, pretende-se utilizar um *Raspberry Pi*, como unidade computacional e de controle dos servo motores da câmera, e um celular *smartphone Android*, responsável por coletar e enviar informações referentes a posição espacial do aparelho.

Com o desenvolvimento do projeto procura-se um melhor entendimento em aspetos relacionados a rede de computadores, sistemas operacionais, interfaces de hardware, modulação por largura de pulso, eletrônica geral e desenvolvimento de sistemas para plataformas móveis.

Competências ligadas a construção de software, utilizando múltiplas linguagens de programação, devem ser evoluídas, já que o módulo de controle, embarcado no *Raspberry Pi*, deve ser construído em linguagem C e o módulo de coleta de dados desenvolvido em Java, usando a API (*Application Programming Interface*) fornecida pelo sistema operacional *Android*.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo descreve as tecnologias e conceitos centrais, utilizados durante a concepção do projeto. As definições apresentadas, são embasadas no material bibliográfico revisado, que serviu de apoio no desenvolvimento de um trabalho fundamentado nas teorias existentes.

2.1 *Raspberry Pi*

O *Raspberry Pi*, é uma família de computadores, com arquitetura de processador ARM (*Advanced RISC Machine*), desenvolvido em um único chip de silício, chamados de SoC (*System on Chip*), com o tamanho de um cartão de crédito. Inicialmente, seu objetivo era promover o ensino de computação (programação) básica em escolas, principalmente públicas, de todo o mundo. Entretanto, por possuir um poder computacional razoável, uma boa quantidade de memória RAM (*Random Access memory*) (a partir do modelo B, ilustrado na Figura 2.1) e um preço relativamente baixo, passou a ser usado para outros objetivos como: console de videogame clássico (emulação de jogos), gerencia de mídia (vídeos, fotos e musicas), estudos em eletrônica, domótica (automação residencial), internet das coisas e robótica. (JUCá; PEREIRA, 2018)

Uma versão do sistema operacional *Debian Linux*, chamada *Raspbian*, foi portada para o *Raspberry Pi*, levando também uma serie de aplicativos e ferramentas de desenvolvimento, já existentes para computadores com arquitetura x86. Dessa modo, o desenvolvimento de programas se torna uma tarefa extremamente simples, já que o sistema operacional fornece a HAL (*Hardware Abstraction Layer*), não sendo necessário o conhecimento especifico do *hardware* do *Raspberry Pi*.

Existem várias linguagens de programação portadas para o *Raspberry Pi*, as mais utilizadas para desenvolvimento de software, com bibliotecas disponíveis para interação com o *hardware*, são o C/C++ e Python porém, é possível desenvolver em outras linguagens de programação como o PHP e Java. (JUCá; PEREIRA, 2018)

Figura 2.1 – Raspberry Pi (Modelo B).

Fonte: (SPARKFUN, 2011)

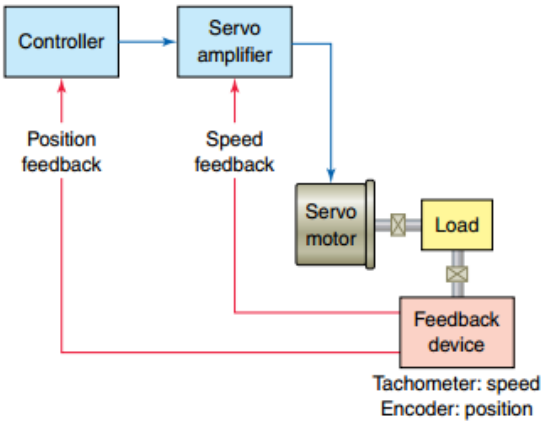
2.2 Servo Motor

Um servo motor, visto na Figura 2.2, é um atuador rotatório, ou atuador linear, que permite um controle preciso da posição linear ou angular, velocidade e aceleração de uma carga ligada ao seu eixo. Consiste basicamente em um motor, normalmente de corrente continua, acoplado a um sensor (potenciômetro), para ler sua posição durante o movimento, conforme ilustrado na Figura 2.3b. Normalmente, os motores servos são controlador por um sinal PWM (*Pulse Width Modulation*). (PETRUZELLA, 2009)

Figura 2.2 – Servo Micro TG9.

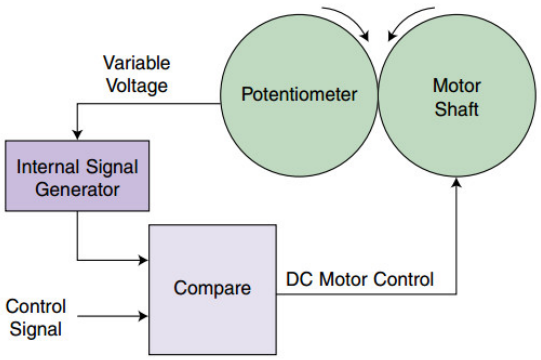
Fonte: (HOBBYKING.COM, 2019)

O servo motor opera em malha fechada, isto é, seu controlador compara a velocidade e sua posição para gerar o próximo comando de movimento, minimizando o erro. (PETRUZELLA, 2009). O esquema de funcionamento do servo motor é mostrado na figura Figura 2.3a.



(a) Sistema em malha fechada.

Fonte: (PETRUZELLA, 2009)



(b) Componentes internos.

Fonte: (PINCKNEY, 2006).

Figura 2.3 – Sistema de controle de um servo motor.

2.3 Modulação por Largura de Pulso - PWM

A modulação por largura de pulso é uma técnica empregada em diversas áreas da eletrônica. Sendo utilizada para controlar fontes chaveadas, velocidade de motores, luminosidade, servo motores e diversas outras aplicações. Consiste em variar a o tempo, em que um pulso de tensão oscila entre os níveis alto e baixo, numa taxa rápida o suficiente, para que a média dos pulsos crie um valor médio de tensão efetivo, ilustrado na Figura 2.4. Ao valor definido pela divisão entre a largura de pulso, com a tensão em nível alto, e o período do sinal, é dado o nome **ciclo de trabalho** ou *dutty cycle*. Variar o *dutty cycle* significa variar a tensão média gerada pelo sinal, isto é, a potência é proporcional a tensão média resultante. (PINCKNEY, 2006).

A largura de pulso pode ser calculada usando a equação Equação (2.1).

$$DuttyCycle = 100 \times \frac{LarguraPulso}{Período} \tag{2.1}$$

Onde: ***DutyCycle*** é um valor dado em porcento, **LarguraPulso** e **Período** são valores dados em segundos.

Uma vez calculado o ciclo de trabalho, é possível calcular o valor da tensão média gerada pelo sinal através da Equação (2.2).

$$TensãoMédia = TensãoPulso \times DutyCycle \tag{2.2}$$

Onde: **TensãoMédia** é a tensão média resultante, dado em volts, **TensãoPulso** é a tensão usada na geração do pulso PWM, também dado em volts, e ***DutyCycle*** é o valor encontrado com a Equação (2.1), dado em porcentagem.

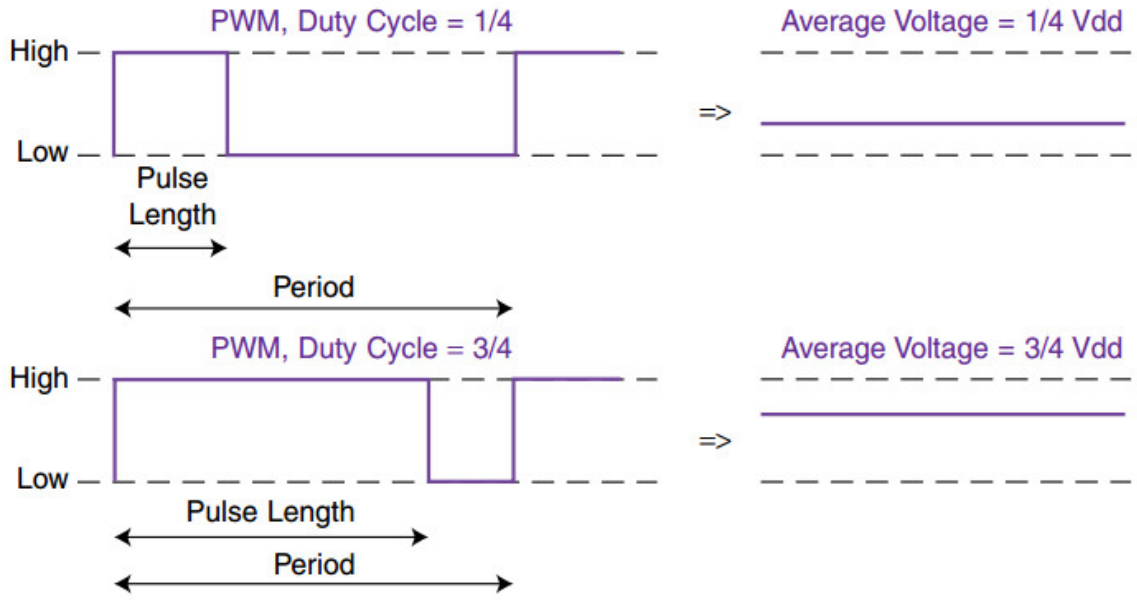


Figura 2.4 – Modulação por largura de pulso e tensão média resultante

Fonte: (PINCKNEY, 2006)

2.4 Smartfone Android

O *smartphone* é um dispositivo móvel, que mescla recursos de um telefone celular (receber e efetuar chamadas, mensagens de texto curto e etc), e recursos de um computador pessoal, garantindo a possibilidade de instalar novos aplicativos, agregando novas funcionalidades ao aparelho.

O *Android* é um sistema operacional de código livre, baseado em núcleo *Linux* (responsável por gerenciar dispositivos de entrada e saída, memória, processos e etc), marcado em vermelho na Figura 2.5, e desenvolvido por uma das maiores empresas de tecnologia da atualidade, a Google (AMADEO, 2018). Sua interface com o usuário é baseada na manipulação direta, isto é, apresenta continuamente o objeto de interesse, permitindo sua manipulação, usando recursos que correspondem proximamente ao mundo físico (BARBOSA; SILVA, 2010). Atualmente, o sistema operacional *Android* ,pode ser encontrado em outros aparelhos como relógios, televisores e dispositivos gerenciadores de mídia (ANDROID.COM, 2019).

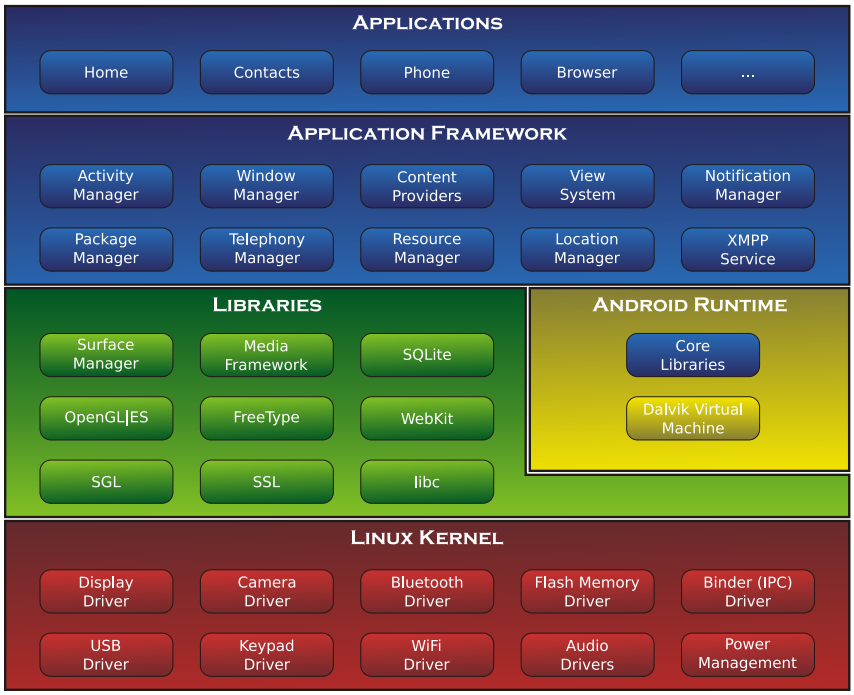


Figura 2.5 – Arquitetura do sistema operacional Android.

Fonte: (BRADY, 2008).

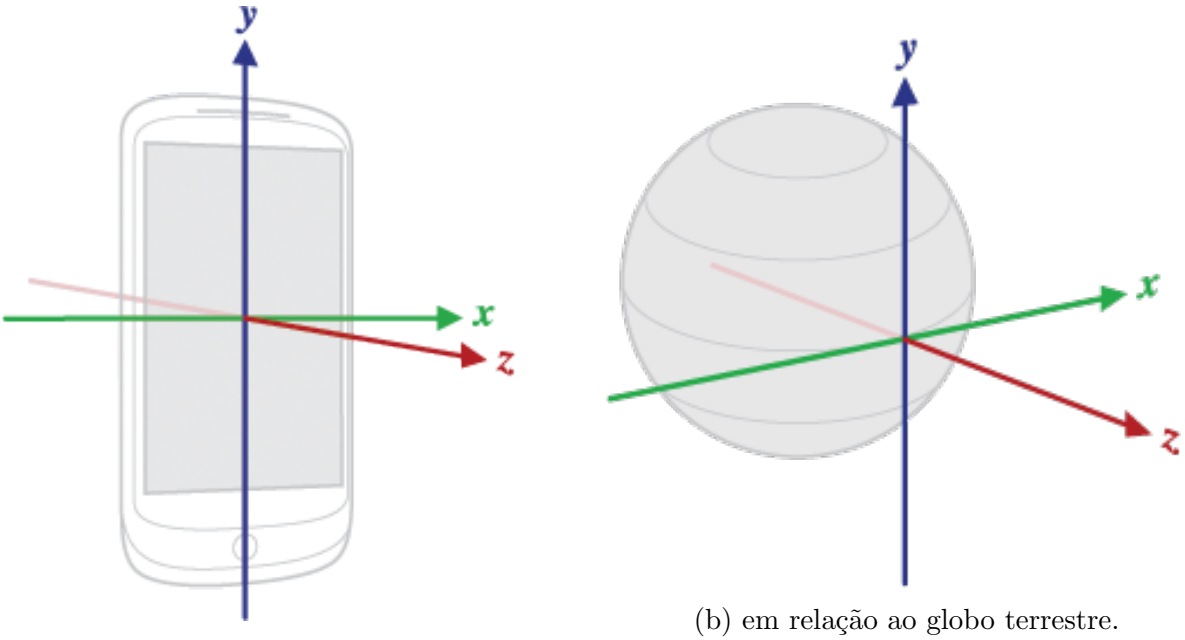
Boa parte dos smartfones, vendidos atualmente com o sistema operacional *Android*, possuem uma gama de sensores embutidos, capazes de fornecer dados, com um grau de precisão aceitável, relativos a orientação e movimento do aparelho, e até condições do ambiente como: iluminação, pressão atmosférica e umidade do ar (ANDROID.COM, 2019). Esses sensores encontram-se divididos em três grupos básicos, na API de desenvolvimento fornecida pelo *Android*: Sensores **Ambientais**, que podem ser utilizados em aplicações simples, como coletar o nível de iluminação do ambiente, a umidade e temperatura, para calcular o ponto de orvalho (condição em que a água em vapor presente num ambiente se condensa tornando-se o orvalho), e os sensores de **Posição** e **Movimento**, que podem ser usados em aplicações como jogos que, por exemplo, utilizam a aceleração da gravidade, para inferir movimentos complexos do usuário como rotações, sacudidas e inclinações do aparelho. Este último grupo sendo o ponto de interesse para o trabalho, e portanto detalhado adiante.

2.4.1 Sensores de Movimento

A API do *Android* oferece vários sensores que permitem sentir o movimento de um dispositivo. Dentre eles, os mais utilizados são os sensores de gravidade, aceleração linear e vetor de rotação, que podem ser implementados em *hardware* ou *software* (baseando-se em dados fornecidos por outros sensores, implementados em *hardware*), e os sensores **acelerômetro** e **giroscópio**, que sempre são implementados em *hardware*. Todos eles,

descrevem o movimento do dispositivo, através de uma estrutura de dados em formato matricial, *array* com mais de uma dimensão.

De forma geral, os sensores usam um sistema de coordenadas com três eixos para expressas os dados. Para a maioria dos sensores da API, o sistema de coordenadas é definido relativo a tela do dispositivo, quando segurado na orientação padrão (porta retrato para celulares e paisagem para *tablets*), como mostrado na Figura 2.6a. Dessa forma, o eixo X é horizontal e aponta para a direita, o eixo Y é vertical apontando para cima e o eixo Z, perpendicular a tela do dispositivo, estando seus valores positivos sobre a tela, e os negativos sob a tela. Um ponto importante a ser notado, é que a orientação do sistema de eixos não muda, quando o dispositivo é segurado de maneira diferente da sua orientação padrão.



(a) em relação ao aparelho.

(b) em relação ao globo terrestre.

Figura 2.6 – Sistema de coordenadas.

Fonte: (DEVELOPERS, 2019).

O sensor **vetor de rotação**, representa a orientação do aparelho como uma combinação de um angulo e um eixo, no qual o dispositivo rotacionou com um ângulo θ em torno de um eixo (x, y ou z) (DEVELOPERS, 2019).

Os três componentes do vetor re rotação são expressos como:

$$\Delta_x = x \times sen(\frac{\theta}{2}) \tag{2.3}$$

$$\Delta_y = y \times sen(\frac{\theta}{2}) \tag{2.4}$$

$$\Delta_z = z \times sen(\frac{\theta}{2}) \tag{2.5}$$

Onde, a magnitude do vetor de rotação é expressa por $sen(\frac{\theta}{2})$, e sua direção é a mesma do eixo de rotação ao qual está multiplicando.

O sistema de coordenada de referencia, é definido como uma base ortogonal direta, isto é, os seus vetores são mutuamente ortogonais, como mostrado na Figura 2.6b, e possui as seguintes características:

- O eixo X é definido como o produto vetorial dos eixos Y e Z, é tangente a superfície terrestre, no local onde o dispositivo se encontra ,e aponta aproximadamente para o leste.
- O eixo Y é tangente a superfície terrestre, no local onde o dispositivo se encontra, e aponta para o norte magnético.
- O eixo Z aponta para o céu e é perpendicular ao solo.

3 Implementação do Protótipo

Este capitulo descreve a montagem do protótipo para testes. Será feita uma introdução do funcionamento geral, e suas principais características. Posteriormente, será mostrado o funcionamento da base de testes da câmera, conexões entre os servo motores e os pinos GPIOs (*General-Purpose Input/Output*) do *Raspberry Pi*, desenvolvimento do programa de controle dos servo motores, desenvolvimento do aplicativo *Android*, responsável por capturar dados dos sensores de posição e envio pela rede sem fio.

3.1 Especificação do Projeto

O projeto pode ser dividido conceitualmente em duas partes, denominados por módulo de coleta de dados, ou celular Android, e módulo de controle de câmera, ou Raspberry Pi. O módulo de controle de câmera é responsável por receber os dados de posição através de uma conexão socket, converter o sistema de coordenadas, aplicar um filtro para evitar o acionamento desnecessário dos motores e acionar os servos, quando necessário. O módulo de coleta de dados é responsável por configurar os sensores de localização disponíveis no smartfone, aplicar os filtros necessários para minimizar ruídos na coleta de dados, encontrar o módulo de controle de câmeras através de uma busca na rede e enviar para ele, os dados das coordenadas através de uma conexão socket.

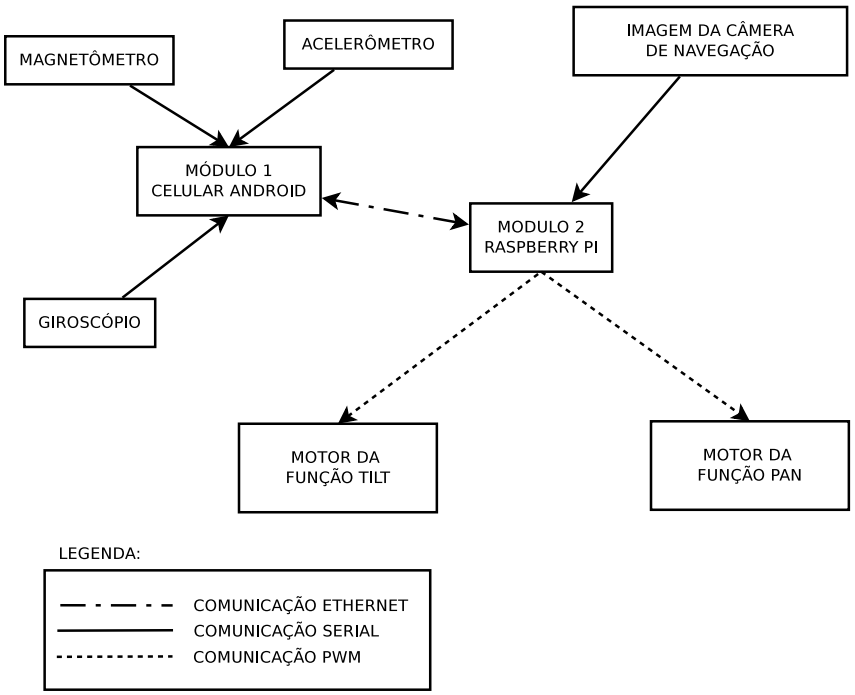


Figura 3.1 – Diagrama de blocos da comunicação entre os módulos

A Figura 3.1 mostra como cada módulo se comunica com seus sensores, atuadores e

entre sí. Os sensores estão embarcados no smartfone e não serão controlados separadamente. O sistema operacional Android fornece uma abstração para o hardware, e portanto não será necessário configurar os sensores manualmente.

3.1.1 Módulo de Controle de Câmeras

O módulo de controle de câmeras consiste em um SoC Raspberry Pi 1 modelo B, que possui um processador ARM de 700MHz, 512Mb de memória RAM, 26 pinos de proposito geral (GPIO), dua portas USB e uma porta Ethernet. Que tem as seguintes responsabilidades:

- Responder requisições broadcast particular que identifica o módulo
- Receber dados de posição de sensores
- Filtrar os dados recebidos para evitar acionamento desnecessário dos motores
- Traduzir coordenadas dos sensores para coordenadas das câmeras
- Acionar dois motores servos de acordo coordenadas recebidas

3.1.2 Módulo de Captura de Movimento

O módulo de envio de coordenadas é um smartfone Android Motorola G6 com processador ARM Qualcomm Snapdragon 450 1,8 GHz Octa-Core, 3GB de memoria RAM, conectividade Wi-Fi, que possui os seguinte sensores: Acelerômetro, Magnetômetro, Giroscópio, Proximidade, Luz Ambiente e leitor de Impressão Digital. Que é responsável por:

- Enviar requisição broadcast especial para detectar o módulo de controle de câmeras
- Conectar no módulo de controle de câmera
- Configurar sensores específicos para detecção de movimento
- Coletar dados dos sensores de movimento
- Aplicar filtro para evitar ruido na coleta dos dados de movimento
- Enviar coordenadas para o módulo de controle de câmera

3.2 Montagem do Protótipo

O protótipo foi construído usando uma fonte de alimentação de computador de mesa, padrão ATX, como fonte de alimentação e suporte para os outros componentes com exceção do smartfone Android.

Cola quente foi usada para fixar o Raspberry Pi, o corpo de montagem da câmera e motores servos (suporte articulado) e um ponto de acesso sem fio, que fornece a infra-estrutura de rede Wi-Fi para o projeto, à fonte e base.

O suporte dos motores servo e da câmera é uma estrutura moldada em plástico, observado na Figura 3.2a, que precisa ser montada com parafusos e fixa os motores em sua posição de descanso (centro de curso), não sendo possível modificar está posição facilmente depois de montado, ilustrado na Figura 3.2b.

(a) Componentes do suporte.

(b) Montagem com motores.

Figura 3.2 – Base de suporta para motores e câmera.

Com a intensão de testar os motores, e preservar o Raspberry Pi (um dispositivo menos tolerante a falhas quando se trata de níveis de tensão nos pinos de propósito geral), foi desenvolvido um circuito gerador de PWM (drive PWM), ilustrado na Figura 3.3, onde era possível ajustar tanto a frequência do sinal quanto a largura dos pulsos, através de ajustes no potenciômetro POT3 e de um resistor que substituiu os potenciômetros POT1 e POT2 no circuito real, implementado na *breadboard*. O propósito do circuito foi descobrir a posição central dos motores e verificar sua estabilidade enquanto se variava a frequência do sinal. O drive foi montado usando o circuito integrado NE555, um componente eletrônico bastante usado em aplicações que envolve temporização, geração de pulso e PWM.

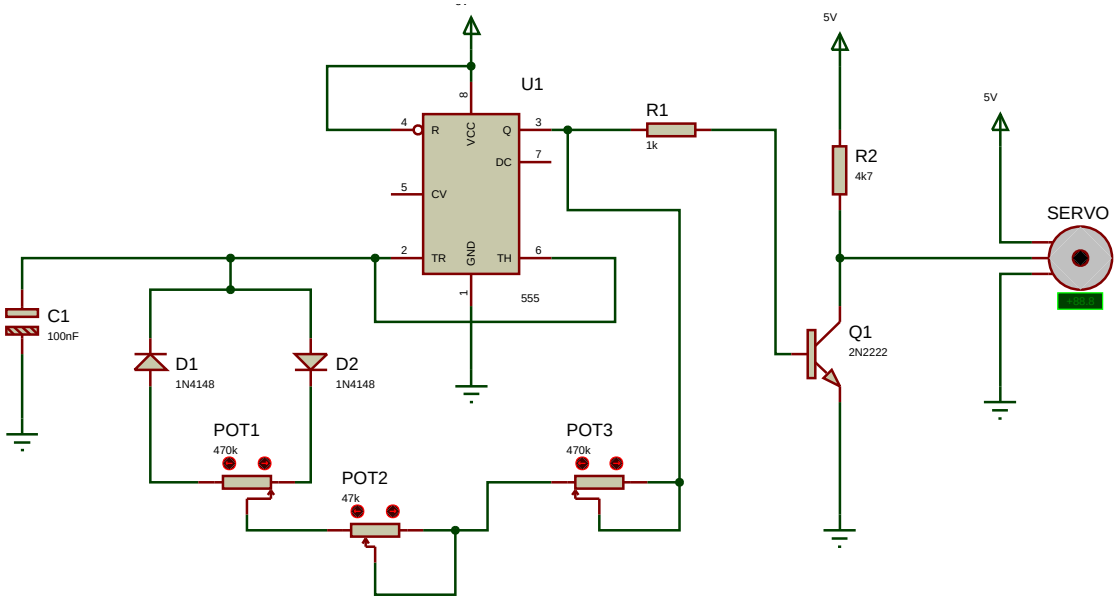


Figura 3.3 – Driver PWM de teste.

Ativando individualmente os motores com o drive construído e com o auxilio de um osciloscópio, foi possível notar que os motores operam de forma estável com pulsos gerados numa frequência de 50Hz, e que é possível controlar a sua posição variando a largura do pulso entre valores próximos a 1ms, posição de angulo 180° (ângulo máximo), e valores próximos 2ms, posição de angulo 0° (ângulo mínimo), como ilustrado na Figura 3.4. Foi possível observar também que existe uma correspondência linear entre a largura do pulso e a posição angular do eixo do motor, sendo assim, a posição de repouso ou centro, pode ser alcançada com uma largura de pulso próximo a 1,5 milissegundos.

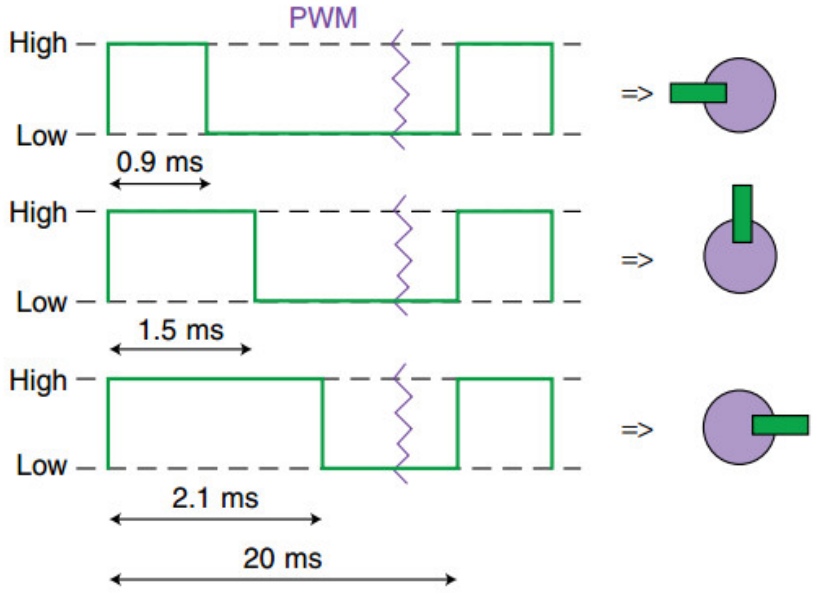


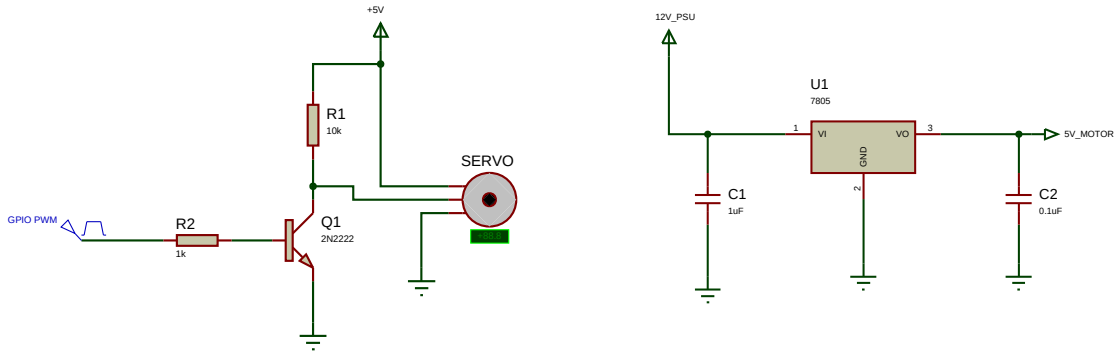
Figura 3.4 – Posição do eixo do motor em função da largura de pulso PWM.

Fonte: (PINCKNEY, 2006)

Para conectar os motores ao módulo de câmera (Raspberry Pi), foi necessário construir um circuito de acionamento dos motores, que consiste em dois transistores bipolares de junção (BJT 2N2222), um para cada motor, configurados no modo emissor-comum, funcionando como um interruptor acionado pelo sinal PWM gerado pelos pinos GPIO do Raspberry Pi. A fonte do sinal PWM foi ligada ao terminal base do transistor através de um resistor de 1Kohm, com a função de limitar a corrente I_b e a via de sinal do motor foi conectado ao terminal coletor do transistor e também a uma fonte de 5V através de um resistor de 10Kohm, responsável por limitar a corrente I_c, máxima quando o transistor está ativado, como ilustrado no circuito da Figura 3.5a.

Esse circuito foi construído numa mini *breadboard* e tem o objetivo proteger o Raspberry Pi, evitando o consumo excessivo de corrente (mais que 15mA) ou uma possível corrente de retorno para um dos pinos GPIO numa eventual falha do circuito interno de controle dos motores. Como o circuito de proteção inverte o sinal PWM, foi necessário gerar um sinal invertido no módulo de controle de câmera, para que esse fosse invertido novamente pelo circuito de proteção e assim chegar como devido ao circuito de controle interno do motor.

A alimentação dos motores é fornecida por um outro circuito composto por um regulador linear de tensão, o LM7805, e dois capacitores, ilustrado na Figura 3.5b, que reduz a tensão fornecida pela PSU de de 12V pra 5V. A linha de 12V foi usada para isolar as fontes de alimentação dos motores e da unidade computacional, o Raspberry Pi, para evitar ruídos causados pelo acionamento dos motores.



(a) Acionamento de motores.

(b) Regulador de tensão.

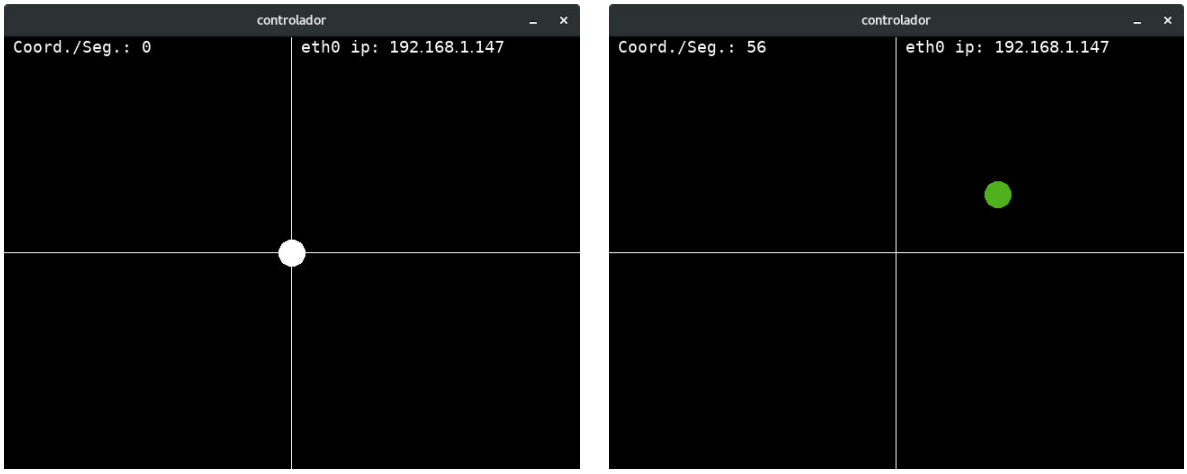
Figura 3.5 – Circuitos usados pelos motores.

3.2.1 Construção do Módulo de Controle de Câmera

O módulo de controle de câmera é responsável por acionar os motores usados para movimentar a câmera e iniciar um serviço de *stream* de mídia que disponibiliza na rede, usando um protocolo de tempo real, as imagens capturadas pela câmera.

Um software de controle foi construído, usando a linguagem C, para receber valores de coordenadas enviados por um socket e acionar os motores.

O software de controle possui dois modos de operação, um modo de produção, que opera consumindo recursos mínimos de memória e processador, e um modo de depuração/teste, habilitado através da diretiva **#define TEST_DISPLAY**.



(a) iniciado.

(b) gerando posição com o click do mouse.

Figura 3.6 – Captura de tela de depuração e teste dos motores do software de controle.

No modo de depuração é mostrada uma tela com um painel negro e quatro quadrantes indicando a posição da câmera através de um ponto branco (localizado no centro dos quadrantes quando o programa inicia); um indicador contendo a contagem de

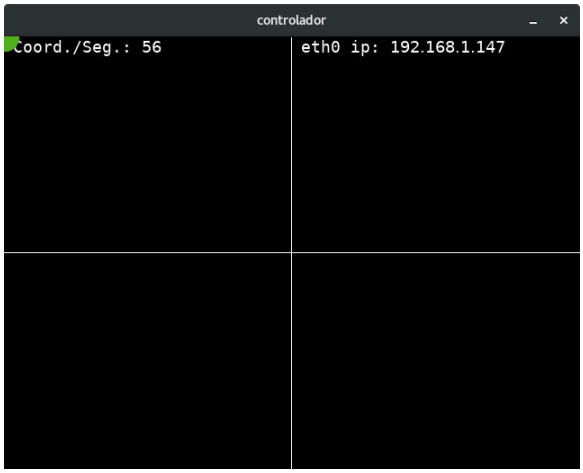
coordenadas recebidas por segundo (no canto superior esquerdo) e o endereço IP que está associado a interface de rede do Raspberry Pi (no canto superior direito), ilustrado na Figura 3.6a e na Figura 3.6b.

Através dessa interface é possível simular posições da câmera usando o mouse. Para isso, o circulo branco deve ser clicado e arrastado para uma posição qualquer dos quadrantes. Ao arrastar e soltar o ponto branco, uma coordenada relativa a posição do ponto na tela é calculada e enviada para os motores, ilustrado na Figura 3.6b.

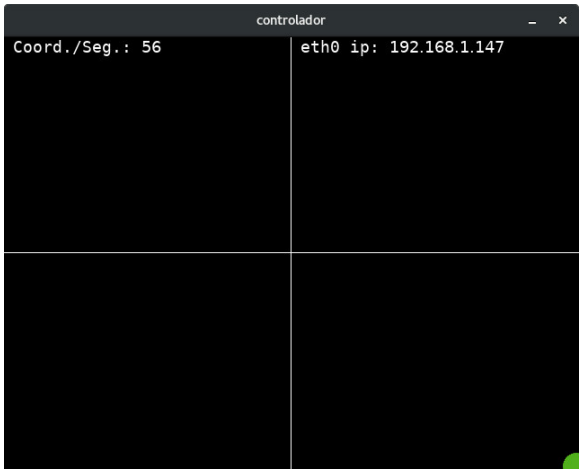
Quando o ponto é movido no sentido horizontal, o motor horizontal é acionado, e de forma semelhante, quando o ponto se move na direção do eixo vertical, o motor vertical é acionado.

As coordenadas do ponto branco são capturadas em relação a sua posição, em *pixels* horizontais e verticais, dentro do painel negro, sendo a coordenada (0;0), a menor possível, localizada no canto superior esquerdo, ilustrado na Figura 3.7a, e a coordenada (640, 480), a maior possível, localizada no canto inferior direito do painel negro, ilustrada na Figura 3.7b.

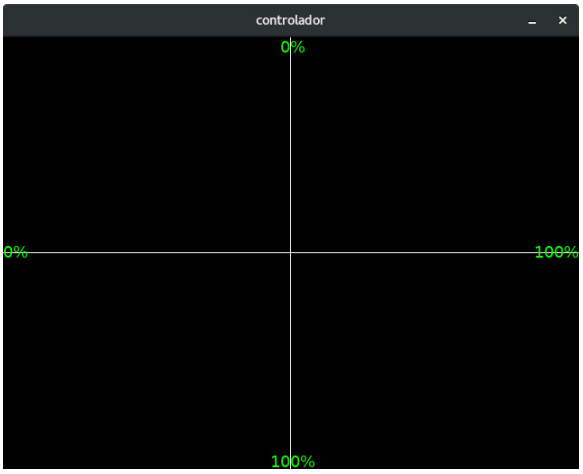
Conforme explicado na Seção 3.2, os motores servos deslocam seu eixo de acordo com a largura do pulso enviado para seu controlador interno. Entretanto, as coordenadas simuladas pela interface de depuração e as que são recebidas do modulo Android não possuem a informação de largura de pulso ou quantidade de ângulos que determinado motor deve ser movimentado. Para resolver esse problema, foi criado um sistema de posição independente da quantidade de *pixels*, largura de pulso ou ângulos.



(a) coordenada mínima



(b) coordenada máxima



(c) orientação dos eixos.

Figura 3.7 – Sistema de coordenadas proporcional.

O sistema de posição proposto é baseado na proporção, ou porcentagem, máxima do movimento possível (do motor, da cabeça do operador e do mouse no painel do programa de teste). Desse modo, o ângulo máximo que o motor horizontal, e a cabeça do operador, podem deslocar é 180°, sendo 90° para a esquerda e 90° para a direita (partindo-se do ângulo 90°, local de descanso do motor e centro de visão do operador), e está relacionado diretamente ao valor máximo de *pixels* que a componente x da coordenada de posição pode assumir, 640 *pixels*, sendo 320 para a esquerda e 320 para a direita (partindo-se do centro da interface). De forma semelhante, movem-se no eixo vertical, o motor e a cabeça do operador com angulo máximo de 180°, mudando apenas a quantidade máxima de *pixels* na interface de teste, que nesse caso é 480, sendo 240 para cima e 240 para baixo.

A relação entre a largura de pulso enviada para os motores horizontal e vertical, em porcento, é dada respectivamente por:

$$PWM_H = \frac{TARGET_X \times 100}{WIDTH} \tag{3.1}$$

$$PWM_V = \frac{TARGET_Y \times 100}{HEIGHT} \tag{3.2}$$

Sendo $WIDTH = 640$ e $HEIGHT = 480$, a largura e altura do painel de teste; $TARGET_X$ e $TARGET_Y$ a localização real em *pixels* horizontais de verticais do circulo indicador.

A Figura 3.7c mostra que o valor dos componentes das coordenada crescem no sentido esquerda-direita horizontalmente e cima-baixo verticalmente. Como o sistema está expresso em porcentagem, os valores das coordenadas são enviados diretamente para os motores, como será visto adiante, no final da Subseção 3.2.2.

Um ajuste foi feito nas coordenadas que são enviadas para o motor servo que movimenta a câmera horizontalmente. Devido a montagem do motor inverter a posição do seu eixo, as coordenadas enviadas pela interface de depuração e pelo módulo de captura de movimento foram invertidas, sendo assim a Equação (3.1) foi ajustada para:

$$PWM_H = 100 - (\frac{TARGET_X \times 100}{WIDTH}) \tag{3.3}$$

3.2.2 Driver PWM

A geração de PWM pode ser realizada através de software ou hardware, usando um dispositivo especializado implementado dentro do processador.

Testes foram feitos no inicio do projeto na expectativa de gerar um sinal PWM adequado via software, usando a biblioteca **RPi.GPIO** escrita em *pyton*, já que a ideia inicial era escrever todos os módulos, inclusive o programa de testes, em *pyton*. Entretanto, os resultados não foram satisfatórios. Os pulsos PWM gerados pela biblioteca não mantinham a frequência e nem a largura do pulso, causando um movimento errático dos motores. A partir desse teste, optou-se por utilizar a linguagem de programação C, nas bibliotecas e ao escrever o código dos softwares utilizados no Raspberry Pi.

Outros testes foram feitos com uma biblioteca chamada **WiringPi**, escrita usando a linguagem C. A biblioteca não oferecia um bom suporte para PWM, sendo mais utilizada para controle geral dos pinos de propósito geral do Raspberry Pi, e portanto foi descartada.

A biblioteca **bcm2835** também foi testada. Foi observado que é uma biblioteca muito extensa e oferece suporte aos diversos dispositivos de hardware como SPI, I2C e PWM, disponíveis no Raspberry Pi. Entretanto, por não possuir uma documentação intuitiva e devido a sua complexidade associada ao curto tempo para a conclusão do projeto, sua utilização foi descartada.

Por fim, o método utilizado para gerar um PWM estável, de forma rápida e confiável foi através do utilitário **ServoBlaster**, um software que provê uma interface de controle do drive PWM via árvore de dispositivos do sistema operacional Linux. Sendo assim, através de um dispositivo localizado em **/dev/servoblaster**, foi possível gerar sinais PWM estáveis para múltiplos pinos do Raspberry Pi.

Para acionar o drive PWM basta abrir o dispositivo **/dev/servoblaster**, como um arquivo, e escrever os comandos de acionamento dos motores.

O utilitário **ServoBlaster** é instalado como um serviço no sistema operacional e fica esperando comandos serem escritos no dispositivo **/dev/servoblaster**, criado durante a instalação. Os comandos de acionamento dos servos são intuitivos e não é necessário conhecimento aprofundado em servo motores ou na eletrônica por trás da geração de PWM. Como exemplo, para mover um motor servo qualquer, conectado a um pino X do Raspberry Pi, para a posição 90°, seu centro ou ponto de descanso, basta usar um emulador de terminal, como o *bash*, e digitar o comando **echo X=50% > /dev/servoblaster**; para mover o mesmo motor para sua posição inicial, ou 0°, basta digitar **echo X=0% > /dev/servoblaster**, e assim sucessivamente para qualquer ângulo θ, sendo $0 \leq \theta \leq 180$.

O módulo de controle das câmeras usa o utilitário *servoblaster* conforme descrito anteriormente, porém acessando o dispositivo **/dev/servoblaster** diretamente, isto é, através das funções de manipulação de arquivo (*open* e *write*), fornecidas pela API C básica do Linux, a *stdlibc*.

3.2.3 Servidor de Imagens

O módulo de câmera também é responsável por enviar as imagens da câmera, em tempo real, para o módulo de captura de movimento. Com esse objetivo, testes foram feitos com a *GStreamer*, uma biblioteca bastante utilizada na criação e manipulação de som e video no ambiente *Linux*. Contudo, devido a uma incompatibilidade do sistema operacional utilizado no *Raspberry Pi* e a biblioteca, a *GStreamer* não foi adotada.

O *VLC*, um *player* bastante difundido e multi-plataforma, com suporte a vários formatos de som e imagem, capaz de realizar *stream*, usando protocolos de tempo real, em modo *headless* (sem carregar uma interface gráfica, via linha de comando) foi testado também. Entretanto, seu uso foi descartado por utilizar demasiadamente o processador (por volta dos 97% de CPU e 71MB da memória RAM), possivelmente causando comportamentos erráticos no controle dos motores.

Testes foram realizados também com o *FFmpeg*, um utilitário multi-plataforma usado em manipulações de som e imagem, bastante versátil, com suporte a um grande número de *codecs* e capacidade para realizar *stream* de áudio e vídeo. O software foi recompilada para o *Rasspberry Pi* com o objetivo de ativar seu suporte ao *codec* de vídeo **h264_omx**, implementado em hardware, no *ASIC* (chip de silicio que contem o processador e todos os periféricos do *Raspberry Pi*), visando reduzir o uso do processador para tarefas de decodificação do vídeo. Por apresentar melhor desempenho (em média 70% do uso do CPU e 46MB de memória RAM), esse utilitário foi usado para enviar o vídeo para o módulo de captura de movimento.

3.2.4 Construção do Módulo de Captura de Movimento

O módulo de captura de movimento é responsável por obter, filtrar e enviar coordenadas relativas a posição da cabeça do operador para o módulo de controle de

câmera. É formado por um software, programado em Java e roda num smartfone Android, que possui uma interface intuitiva de operação, ilustrado na Figura 3.8, e um conjunto de sensores embutidos no celular, que são capazes de fornecer dados relativos a orientação com uma boa acuracidade.

O programa de captura de coordenadas pode ser resumido basicamente em quatro partes principais: **coleta de dados** de posição da cabeça do operador; **envio de coordenadas**; **interação com o operador** (interface do usuário); e, **exibição do vídeo** capturado pela câmera. Que serão detalhadas adiante, iniciando-se pela interface com o usuário.

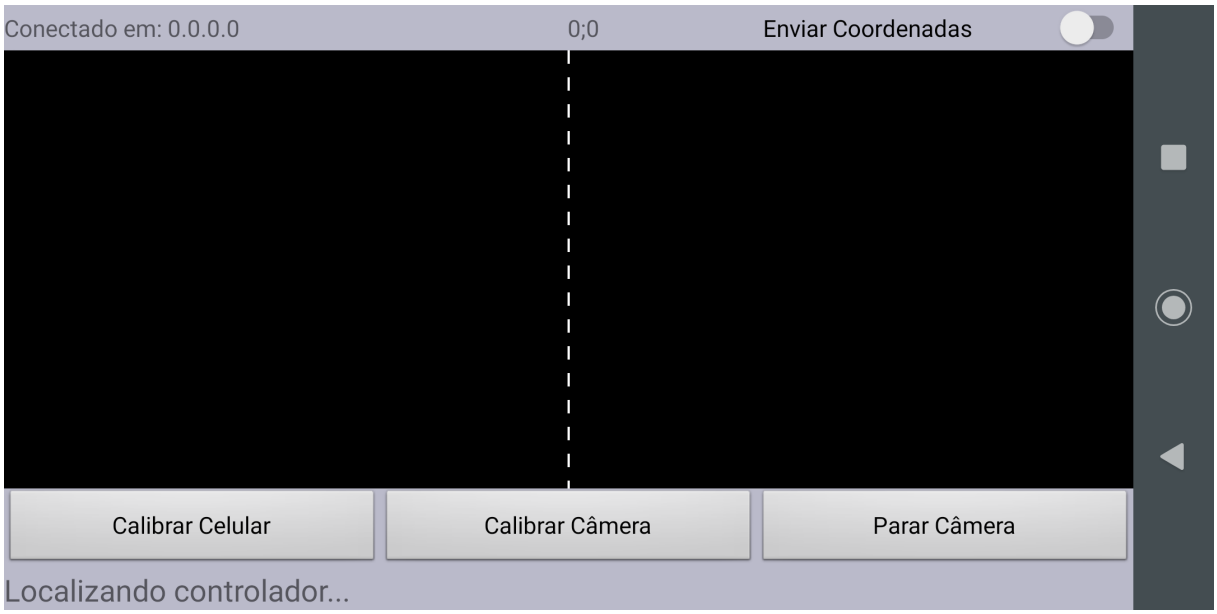


Figura 3.8 – Interface do Módulo de Captura de Movimento.

A captura de tela da interface do Módulo de Captura de Movimento, ilustrada na Figura 3.8, mostra uma serie de componentes visuais, que trazem informações pertinentes ao funcionamento do sistema e permitem interagir remotamente com o Módulo de Controle de Câmera.

O **indicador de conexão**, localizado no canto superior esquerdo da interface, indica a qual endereço IP está associado ao servidor responsável por controlar os motores que movimentam a câmera que ao mesmo tempo disponibilizam as imagens capturadas. Quando o seu valor é **0.0.0.0**, significa que está desconectado de um servidor e não será possível enviar coordenadas ou receber imagens.

O **indicador de coordenadas**, localizado na parte central superior da interface, indica qual coordenada está sendo capturada pelos sensores de movimento do celular em tempo real. Seu formato é semelhante a uma coordenada cartesiana, isto é, a componente da x seguindo da componente y e separada por um "ponto e virgula". Esse indicador mostra pontos do sistema de coordenadas proposto na Subseção 3.2.1, desse modo, a coordenada **50;50** é equivalente ao centro da interface de depuração.

No canto superior direito da interface, se encontra o **controle de envio de coordenada**, uma chave do tipo *on-off* que somente é habilitada quando o Módulo de Captura de Movimento está conectado a um servidor de controle de câmera. Quando a chave está na posição *on*, ou ativada, as coordenas são enviadas para o servidor de controle de câmera.

Ao centro da interface, numa área em cor preta, está localizado o **painel de imagens**. Este é o local onde as imagens da câmera são mostradas quando recebidas do módulo de controle de câmera. O painel de imagens é dividido, verticalmente ao centro em duas porções, indicado pela linha tracejada em cor branca.

A divisão é devida ao sistema de vídeo *stereo* simulado, onde uma única imagem enviada pelo módulo de controle de câmera é duplicada e manipulada de forma a criar um efeito *parallax*, isto é, criar a sensação de que existem duas fontes de vídeo (duas câmeras), capturando imagens de um mesmo objeto a partir de ângulos de visão distintos, trazendo a sensação 3D para o operador.

A interface possui uma **barra de *status***, localizada na parte inferior, que indica qual tarefa está sendo executada no momento. A barra de *status* mostrada na Figura 3.8 indica que quando a imagem foi capturada o Módulo de Captura de Movimento estava buscando o Módulo de Controle de Câmera.

O mecanismo de busca do Módulo de Controle de Câmera se dá através do disparo em *broadcast* de pacotes do tipo *User Datagram Protocol*, mais conhecido como pacotes UDP, com uma mensagem especifica, somente respondida pelo Módulo de Controle de Câmera com o seu endereço IP associado. Desse modo, o Módulo de Captura de Movimento configura o endereço IP respondido como um Módulo de Controle de Câmera válido e habilita o controle de envio de coordenadas.

Acima da barra de *status* e abaixo do painel de imagens, está localizado o menu de botões, contendo as funções de **calibração do celular**, que consiste em salvar as coordenadas referentes a posição atual do celular, usadas posteriormente para calcular uma nova coordenada ajustada que será enviada para o Módulo de Controle de Câmera; a função de **calibração de câmera**, que envia uma mensagem de calibração para o Módulo de Controle de Câmera para salvar a coordenada referentes a atual da câmera, usada posteriormente como referencia para movimentar a câmera; e, por fim, a função **Parar/Enviar Câmera**, que controla o envio de imagens do Módulo de Controle de Câmera.

Devido a escassez de recursos de hardware no Raspberry Pi (processador e memória RAM) associados a qualidade inferior da câmera utilizada no protótipo, a imagem capturada e enviada para o *smartphone* tem péssima resolução (320x240 *pixels*). Quando escalonada para preencher a área disponível na interface do celular, resultou numa figura desforme que pouco representa o objeto filmado. Sendo assim, apenas uma imagem está sendo mostrada, em tamanho reduzido e fora de alinhamento com a interface, conforme ilustrado na

Figura 3.9. Também, devido aos mesmos problemas relacionados ao hardware e qualidade da câmera, existe um atraso de aproximadamente 4 segundos entre as imagens que são coletadas na câmera e as imagens que são exibidas no aplicativo do celular.

Figura 3.9 – Imagem da câmera em tamanho original.

Para mostrar as imagens capturadas pela câmera no celular e transmitida via *RTP - Real Time Protocol* (Protocolo de Tempo Real), foi utilizado a biblioteca do *VLC*, a mesma testada na Subseção 3.2.3. Foi recompilada para o sistema operacional Android e integrada ao programa de captura de movimento. Essa biblioteca foi selecionada para ser integrada por oferecer opções de compatibilidade com o sistema operacional Android, ser compatível com diversos formatos de vídeo e ter suporte ao *RTP*.

A coleta de dados de movimento, acontece a cada 30 milissegundos, através dos eventos de posição fornecidos pela API do sistema operacional Android, que simplificam bastante o desenvolvimento de aplicações, abstraindo toda configuração do hardware dos sensores e a densa matemática, necessária para converter leitura de forças exercidas sobre o aparelho e interações dos vetores aceleração, em uma estrutura de dados que representa numericamente a posição espacial em que se encontra o celular. Contudo, é necessário identificar a orientação do aparelho, Figura 2.6a, em relação ao sistema de eixos do planeta, ilustrado na Figura 2.6b para que a coleta de dados funcione adequadamente.

O envio de coordenadas ocorre assim que um evento de posição é disparado. Ou seja, um novo envio de coordenada acontece a cada 30 milissegundos. Supondo que um par de coordenadas em média consuma 5 bytes (2 bytes para a coordenada x, 2 bytes para a coordenada y e 1 byte para um caractere separador), seriam enviados 167 bytes por segundo, desconsiderando o *overhead* do protocolo TCP, usado para esta finalidade. O consumo da banda de dados, gerado pela transmissão, pode ser cm siderado irrisório para uma banda disponível de 100Mbit/s, atualmente comum em redes do tipo Ethernet.

Entretanto, visando reduzir os recursos computacionais do celular, e consequentemente economizar bateria, um filtro verifica se houve mudança nas coordenadas antes de enviá-las, reduzindo o uso da banda de dados para aproximadamente 7 *bytes* por segundo.

4 RESULTADOS

Nesse capitulo serão discutido, os resultados globais obtidos com o desenvolvimento do protótipo e, mostrado um teste comparativo entre dois *codecs* de video testados com a biblioteca *FFmpeg*.

4.0.1 Resultados Globais

De modo geral, o protótipo se comportou conforme o esperado. A captura de movimento da cabeça do operador funcionou de maneira satisfatória e os motores respondem sem atrasos perceptíveis (que representassem desconforto ao operador). Entretanto, em determinados momentos, os motores apresentaram movimentos bruscos, como se tivessem perdido coordenadas. Posteriormente, descobriu-se que o comportamento inesperado dos motores, acontece devido a um mau contato entre os pinos dos *jumpers*, usado para fazer a conexões entre os componentes eletrônicos, os motores e a *breadboard*.

O sistema de envio de imagens precisa ser optimizado. Talvez um dispositivo de captura com mais qualidade, como o módulo de câmera oficial do Raspverry Pi, que possui um drive especifico implementado em hardware, represente uma melhoria no sistema global de captura e enfio de imagens para o celular.

Os eventos de detecção de posição, gerados pela API do Android, ocorrem em um tempo especificado pelo desenvolvedor. Para identificar esse tempo, levou-se em consideração a responsividade necessária para os motores e o uso de banda de dados. Quanto menor o intervalo de tempo entre os eventos, maior a resolução do movimento, consumo de banda de dados e custo computacional. Para chegar-se a um valor adequado, testes práticos foram feitos variando-se o intervalo de eventos entre 10 e 500 milissegundos. Chegou-se a conclusão que valores entre 30 e 100 milissegundos são satisfatórios.

O consumo da banda de dados, durante o envio de coordenadas, foi minimizado pela aplicação de um filtro que compara a última coordenada válida com a que será enviada, conforme explicado no final da Subseção 3.2.4. Desse modo somente as modificações de posição são enviadas ao módulo de controle de câmeras. O resultado da aplicação desse filtro pode ser visualizado comparando-se a Figura 4.1 e a Figura 4.2.

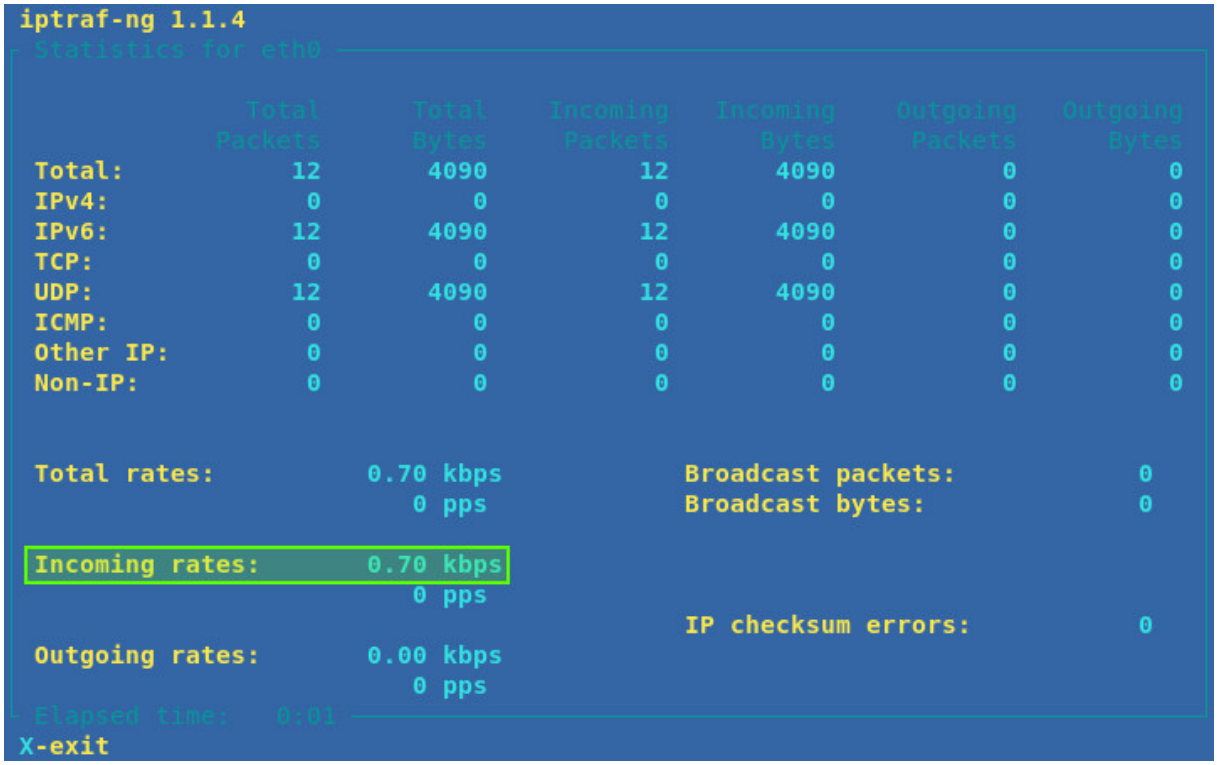


Figura 4.1 – Consumo dos recursos de rede durante o recebimento de coordenadas.

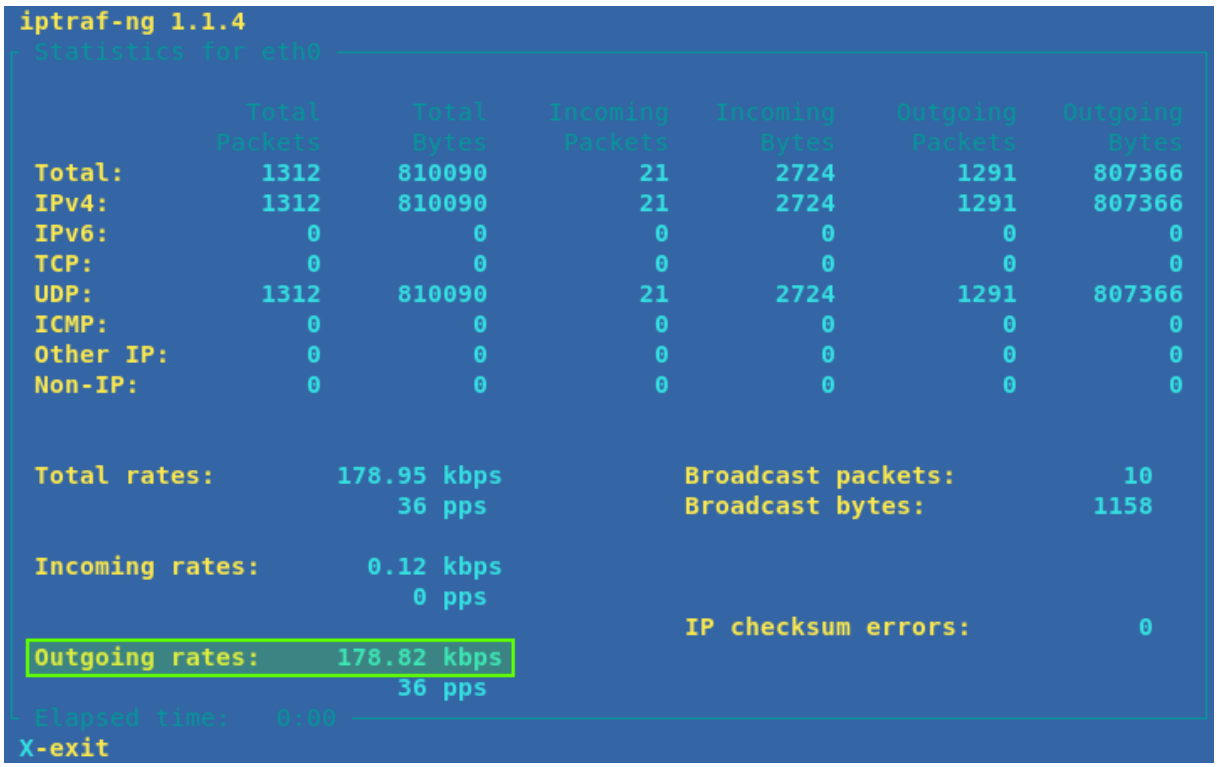


Figura 4.2 – Consumo dos recursos de rede durante o envio de imagem da câmera.

4.0.2 Comparação Entre *codecs*

Dentre os testes realizados com alguns dos *codecs* de vídeo compatíveis com o FFmpeg, os que mais chamaram atenção e que representaram um ganho considerável, em relação ao custo computacional, foram os *codecs* **h264_omx** e **h264**, implementados em hardware e software respectivamente. Comparando-se a Figura 4.4 e a Figura 4.3, obtidas da interface do gerenciador de tarefas *top*, fica claro que a implementação de *codecs* via hardware traz um ganho, em tempo de processador e quantidade de memoria *RAM* alocada, considerável. Existe um ganho de aproximadamente 28% em tempo de processador e 35% em uso de memoria *RAM*. Uma parte do uso de recursos nas duas figuras está relacionada ao processo de *streaming*, implementada via software. Como o *streaming* é o mesmo para as duas comparações, não existe inconsistência na comparação.

```
top - 17:41:39 up 7 min,  3 users,  load average: 1,00, 0,76, 0,39
Tasks:  73 total,   2 running,  44 sleeping,   0 stopped,   0 zombie
%Cpu(s): 60,2 us,  9,5 sy,  0,0 ni, 21,7 id,  0,0 wa,  0,0 hi,  8,6 si,  0,0 st
MiB Mem :  432,699 total,  270,910 free,   57,406 used,  104,383 buff/cache
MiB Swap:   99,996 total,   99,996 free,    0,000 used.  321,781 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  579 pi        20   0  193,3m  46,6m  40,8m R 70,2 10,8   1:04.33 ffmpeg
  536 pi        20   0    7,9m   3,2m   2,7m R  2,5  0,7   0:10.79 top
   39 root       1 -19    0,0m   0,0m   0,0m S  2,2  0,0   0:04.24 vchiq-slot/0
    7 root      20   0    0,0m   0,0m   0,0m S  0,3  0,0   0:00.78 ksoftirqd/0
  219 root      20   0   26,9m   1,3m   1,2m S  0,3  0,3   0:01.07 rngd
  514 pi        20   0   11,2m   3,7m   3,0m S  0,3  0,9   0:02.03 sshd
    1 root      20   0    9,4m   5,9m   4,8m S  0,0  1,4   0:04.69 systemd
    2 root      20   0    0,0m   0,0m   0,0m S  0,0  0,0   0:00.00 kthreadd
    3 root      20   0    0,0m   0,0m   0,0m I  0,0  0,0   0:00.81 kworker/0:0-eve
    6 root       0 -20    0,0m   0,0m   0,0m I  0,0  0,0   0:00.00 mm_percpu_wq
    8 root      20   0    0,0m   0,0m   0,0m S  0,0  0,0   0:00.01 kdevtmpfs
    9 root       0 -20    0,0m   0,0m   0,0m I  0,0  0,0   0:00.00 netns
   10 root      20   0    0,0m   0,0m   0,0m I  0,0  0,0   0:01.61 kworker/0:1-eve
   11 root      20   0    0,0m   0,0m   0,0m S  0,0  0,0   0:00.00 khungtaskd
   12 root      20   0    0,0m   0,0m   0,0m S  0,0  0,0   0:00.00 oom_reaper
   13 root       0 -20    0,0m   0,0m   0,0m I  0,0  0,0   0:00.00 writeback
   14 root      20   0    0,0m   0,0m   0,0m S  0,0  0,0   0:00.00 kcompactd0
   15 root       0 -20    0,0m   0,0m   0,0m I  0,0  0,0   0:00.00 crypto
```

Figura 4.3 – Consumo de recursos de hardware pelo FFmpeg usando o codec h264_omx, implementado em hardware.

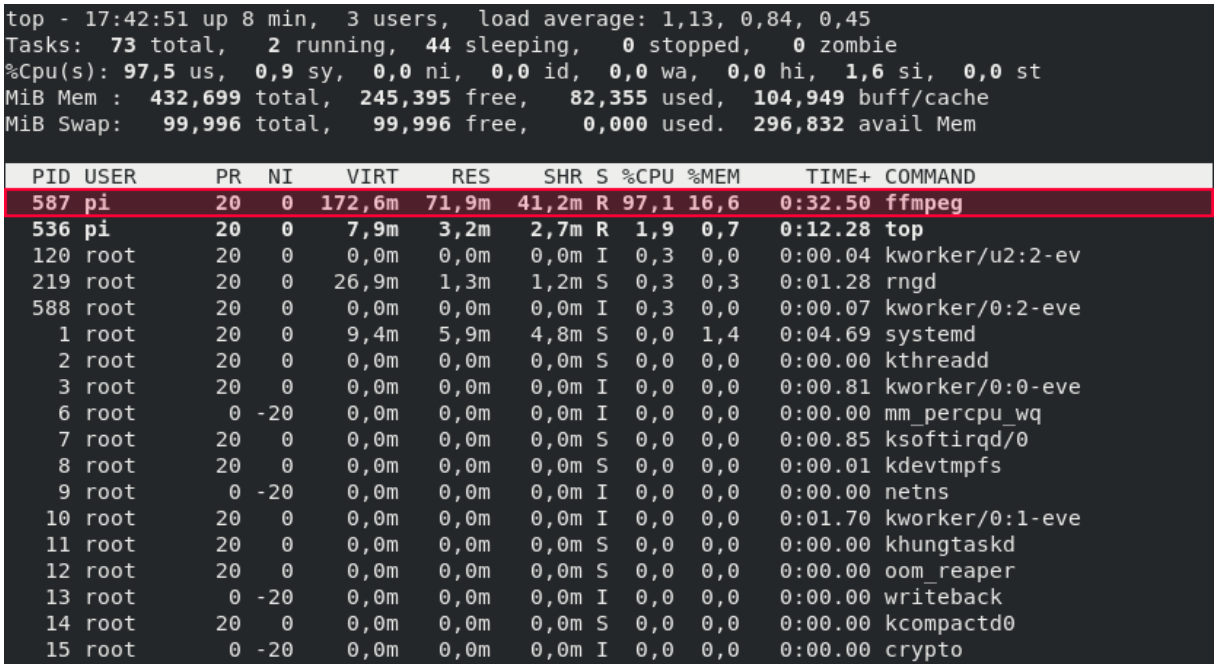


Figura 4.4 – Consumo de recursos de hardware pelo FFmpeg usando o codec h264, implementado em software.

5 CONCLUSÃO

Este trabalho apresenta o desenvolvimento de um sistema do tipo *Pan* e *Tilt*, para controle remoto de câmera, com base no movimento da cabeça do operador, capturado por sensores de um celular, acoplado ao usuário através de um óculos do tipo *VR*.

Com base nos resultados obtidos, e corrigindo-se questões pertinentes ao envio de imagens de video, um sistema semelhante ao construído, pode ser utilizado para facilitar o manuseio, ou operação, de um dispositivo de inspeção. Sendo assim, um operador pode usar os movimentos de sua cabeça para controlar a câmera de um robô, reduzindo o numero de ações vinculada ao controle manual, normalmente necessárias para movimentar o robô, suas ferramentas e a câmera embarcada.

O sistema de imagem *stereo*, aumenta a percepção do local, uma vez que adiciona a sensação de profundidade às imagens, facilita a locomoção, a orientação e, consequentemente, a inspeção de ambientes.

O projeto foi elaborado usando um *Raspberry Pi*, como unidade computacional e de controle dos servo motores da câmera, e um celular *smartphone Android*, responsável por coletar e enviar informações referentes a posição espacial do aparelho. O desenvolvimento do projeto proporcionou um melhor entendimento em aspetos relacionados a rede de computadores, sistemas operacionais, interfaces de hardware, modulação por largura de pulso, eletrônica e desenvolvimento de sistemas para plataformas móveis.

Competências ligadas a construção de software puderam ser evoluídas, já que o módulo de controle, embarcado no Raspberry Pi, foi construído em linguagem C e o módulo de coleta de dados foi desenvolvido em Java, usando a API do sistema operacional Android.

5.1 Trabalhos Futuros

- Corrigir problemas relacionados ao envio de imagens video para o celular.
- Desenvolver um mecanismo automático de calibração do celular (independente da orientação magnética do planeta)
- Criar um modo de tela cheia para melhor visualização das imagens com o óculos de realidade virtual.
- Adicionar suporte a conexão *bluetooth* entre um *joysticks* e o celular, permitindo interação com a interface de teste e enviar comandos para o robô.
- Criar uma API para possibilitar a integração com outras soluções (ex: mostrar dados capturados de sensores embarcados na imagem enviada para o celular).

- Adicionar captura de som *stereo* no módulo de controle dos motores.
- Transmitir o sinal de som capturado junto com o vídeo para o celular, aumentando a sensação de imersão para o operador.
- Melhoria do sistema de alimentação para a solução (tornar portátil).
- Construir uma placa para evitar o mau contato entre componentes eletrônicos no circuito de acionamento dos motores (construção de uma *Raspberry Pi hat*).

REFERÊNCIAS

AMADEO, R. **Google's iron grip on Android: Controlling open source by any means necessary**. 2018. Disponível em: <https://arstechnica.com/gadgets/2018/07/googles-iron-grip-on-android-controlling-open-source-by-any-means-necessary/>. Citado na página 19.

ANDROID.COM. **What is Android**. 2019. Disponível em: <https://www.android.com/what-is-android>. Citado 2 vezes nas páginas 19 e 20.

BARBOSA, S.; SILVA, B. **Interação humano-computador**. [S.l.]: Elsevier Brasil, 2010. Citado na página 19.

BORTOLETTO, M. É.; MACHADO, R. R.; COUTINHO, E. et al. Contaminação fúngica do acervo da biblioteca de manguinhos da fundação oswaldo cruz: Ações desenvolvidas para sua solução. Universidade Federal de Santa Catarina, 2002. Citado na página 13.

BRADY, P. **Anatomy and Physiology of an Android–2008 Google I/O Session Videos and Slides**. [S.l.]: Online: https://sites.google.com/site/io/anatomy–physiology-of-an-android, 2008. Citado na página 20.

CARMO, A. T.; PRADO, R. T. A. **Qualidade do ar interno**. [S.l.]: EPUSP São Paulo, 1999. Citado na página 13.

DEVELOPERS, G. A. **Motion sensors**. 2019. Disponível em: <https://developer.android.com/guide/topics/sensors/sensors_motion>. Citado na página 21.

HOBBYKING.COM. **Servo Motor TG9**. 2019. Online; acessado em 18/08/2019. Disponível em: <https://cdn-global-hk.hobbyking.com/media/catalog/product/cache/9/image/660x415/17f82f742ffe127f42dca9de82fb58b1/legacy/catalog/9549.jpg>. Citado na página 17.

HUGHES, S.; LEWIS, M. Robotic camera control for remote exploration. In: ACM. **Proceedings of the SIGCHI conference on Human factors in computing systems**. [S.l.], 2004. p. 511–517. Citado na página 14.

JUCá, S.; PEREIRA, R. **Aplicações Práticas de sistemas embarcados Linux utilizando Raspberry Pi**. [S.l.]: PoD, 2018. Citado na página 16.

MOLINA, L.; CARVALHO, E.; MOURA, M.; FREIRE, E.; MONTALVÃO, J. Um método de visão robótica para identificação de cordões de solda em tanques de armazenamento visando inspeção automatizada. **XVII CBA**, 2008. Citado na página 13.

PETRUZELLA, F. **Electric motors and control systems**. [S.l.]: McGraw-Hill, Inc., 2009. Citado 2 vezes nas páginas 17 e 18.

PINCKNEY, N. Pulse-width modulation for microcontroller servo control. **IEEE potentials**, IEEE, v. 25, n. 1, p. 27–29, 2006. Citado 3 vezes nas páginas 18, 19 e 27.

PROCESSONLINE.COM.AU. **ABB wins award for submersible inspection robot**. 2019. Online; acessado em 18/08/2019. Disponível em: <https://www.processonline.com.au/content/factory-automation/news/abb-wins-award-for-submersible-inspection-robot-993485352>. Citado na página 14.

SANTOS, E. C. d. S. et al. **Inspeção e adequação das instalações elétricas e procedimentos de trabalho de uma empresa à norma regulamentadora NR-10.** Tese (Doutorado) — UNIVERSIDADE DE SÃO PAULO, 2012. Citado na página 13.

SPARKFUN. **Prduto Raspberry Pi - Model B**. 2011. Online; acessado em 18/08/2019. Disponível em: <https://cdn.sparkfun.com//assets/parts/7/4/9/7/11546-01.jpg>. Citado na página 17.

TRACTICA.COM. **Sensabot Is the First Inspection Robot Approved for Use by Oil and Gas Companies**. 2016. Online; acessado em 18/08/2019. Disponível em: <https://www.tractica.com/robotics/sensabot-is-the-first-inspection-robot-approved-for-use-by-oil-and-gas-companies/>. Citado na página 14.

Apêndices

APÊNDICE A – Nome do apêndice

Lembre-se que a diferença entre apêndice e anexo diz respeito à autoria do texto e/ou material ali colocado.

Caso o material ou texto suplementar ou complementar seja de sua autoria, então ele deverá ser colocado como um apêndice. Porém, caso a autoria seja de terceiros, então o material ou texto deverá ser colocado como anexo.

Caso seja conveniente, podem ser criados outros apêndices para o seu trabalho acadêmico. Basta recortar e colar este trecho neste mesmo documento. Lembre-se de alterar o ”label” do apêndice.

Não é aconselhável colocar tudo que é complementar em um único apêndice. Organize os apêndices de modo que, em cada um deles, haja um único tipo de conteúdo. Isso facilita a leitura e compreensão para o leitor do trabalho.

APÊNDICE B – Instalação e Configuração do Sistema Operacional do Raspberri Pi (Raspbian)

conteúdo do novo apêndice

APÊNDICE C – Fotos do Protótipo

Anexos

ANEXO A – Nome do anexo

Lembre-se que a diferença entre apêndice e anexo diz respeito à autoria do texto e/ou material ali colocado.

Caso o material ou texto suplementar ou complementar seja de sua autoria, então ele deverá ser colocado como um apêndice. Porém, caso a autoria seja de terceiros, então o material ou texto deverá ser colocado como anexo.

Caso seja conveniente, podem ser criados outros anexos para o seu trabalho acadêmico. Basta recortar e colar este trecho neste mesmo documento. Lembre-se de alterar o ”label” do anexo.

Organize seus anexos de modo a que, em cada um deles, haja um único tipo de conteúdo. Isso facilita a leitura e compreensão para o leitor do trabalho. É para ele que você escreve.

ANEXO B – Nome do outro anexo

conteúdo do outro anexo