Universidade de São Paulo

Departamento de Engenharia Elétrica e de Computação

Guilherme Jabur Rossiti

Implementação de um sistema de controle multiplataforma e desenvolvimento de um servidor embarcado centralizador de comandos para automação residencial

Monografia

Implementação de um sistema de controle multiplataforma e desenvolvimento de um servidor embarcado centralizador de comandos para automação residencial

Guilherme Jabur Rossiti

Orientador: Evandro Luís Linhari Rodrigues

Monografia final de conclusão do curso Engenharia de Computação apresentada ao Departamento de Engenharia Elétrica e de Computação - EESC - USP.

São Carlos, Brasil

Junho de 2014

Rossiti, Guilherme Jabur
RR831i Implementação de um sistema de controle multiplataforma e desenvolvimento de um servidor embarcado centralizador de comandos para automação residencial / Guilherme Jabur Rossiti; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2014.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Automação Residencial. 2. Servidor Embarcado. 3. Kinect. 4. Raspberry Pi. 5. Android. 6. iOS. I. Título.

FOLHA DE APROVAÇÃO

Nome: Guilherme Jabur Rossiti

Título: Implementação de um sistema de controle multiplataforma e desenvolvimento de um servidor embarcado centralizador de comandos para automação residencial

Trabalho de Conclusão de Curso defendido em 10/06/2014.

Comissão Julgadora:	**Resultado:**
Prof. Associado Evandro Luís Linhari Rodrigues (Orientador) - SEL/EESC/USP	Aprovado
Prof. Dr. Valdir Grassi Júnior SEL/EESC/USP	Aprovado
Prof. Associado Adenilso da Silva Simão SSC/ICMC/USP	APROVADO

Coordenador do Curso Interunidades - Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

“ O que mais surpreende é o homem, pois perde a saúde para juntar dinheiro, depois perde o dinheiro para recuperar a saúde. Vive pensando ansiosamente no futuro, de tal forma que acaba por não viver nem o presente, nem o futuro. Vive como se nunca fosse morrer e morre como se nunca tivesse vivido. ”

Dalai Lama.

Agradecimentos

Agradeço, em primeiro lugar, a Deus, por ter iluminado o meu caminho durante toda minha trajetória, e por ter inserido pessoas incríveis nele. Pessoas essas que me ensinaram a viver, pessoas essas que me forneceram todo o apoio e conhecimento necessário para que eu me tornasse o que sou hoje. Algumas dessas pessoas são:

Meu orientador, Professor Doutor Evandro Luis Linhari Rodrigues, responsável por me fornecer conselhos e críticas durante alguns dos meus momentos mais difíceis da Universidade. Agradeço pela disponibilidade, confiança e principalmente a paciência despendida a mim.

Os meus pais, Regina Maria Jabur Rossiti e Luís Gonzaga Rossiti, meus maiores exemplos e meus professores eternos, responsáveis pelo meu desenvolvimento e pela formação dos meus valores. Heróis.

Meu irmão, André Jabur Rossiti, a pessoa a qual me espelhei durante grande parte da minha vida. A pessoa a qual foi obrigada, ainda é obrigada, a me aturar nos momentos aos quais eu consigo ser o mais irritante do mundo.

Todos os meus familiares, avôs, avós, tios, tias, primos e primas, apoiadores incondicionais durante toda a minha vida. Em especial aqueles que possuem vogais no nome.

Meus amigos da Escola, da Universidade, da Noruega e da Vida, todos que me forneceram palavras, risadas, gestos e silêncios que serão lembrados eternamente. Todos que compartilharam dos momentos felizes e tristes na minha caminhada.

Todas as pessoas que foram “obrigadas” a conviver comigo em uma mesma casa, membros da República Tira Gosto, do Apartamento 22, do Apartamento 51-A e à Maria Jensen (*Tusen takk!*).

Meus professores, educadores, motivadores, mestres da arte de ensinar.

Todos aqueles que contribuíram de forma direta ou indireta para a realização deste trabalho.

As palavras são poucas para agradecer tamanha gratidão, mas ainda assim…

Obrigado!

Resumo

ROSSITI, G. J. **Implementação de um sistema de controle multiplataforma e desenvolvimento de um servidor embarcado centralizador de comandos para automação residencial.** 2014. 70 f. Trabalho de Conclusão de Curso (Graduação) - Escola de Engenharia de São Carlos, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2014.

A inserção da tecnologia no ambiente residencial está chegando em níveis cada vez mais elevados, com processos de automações e sistemas de gerenciamento centralizados. Seguindo esta evolução estão também outras tecnologias, como a popularização do conceito NUI (*Natural User Interface*) e a ascensão dos *smartphones* no cotidiano das pessoas. Este trabalho visa unir todos esses avanços contemporâneos para a implementação de um servidor embarcado, centralizador de comandos, e seus respectivos controladores. Com isso, deseja-se gerir diferentes dispositivos eletrônicos residenciais a partir de *smartphones*, computadores, ou até mesmo por gestos naturais humanos. No desenvolvimento deste projeto, objetivando atingir suas metas, foram utilizados conceitos de infravermelho para o controle de aparelhos televisivos e possíveis equipamentos de refrigeração, princípios de funcionamento dos relés para o chaveamento de circuitos de tensão elevada, e tópicos relacionados à comunicação *socket* para a implementação de sistemas de controle, utilizando o *Kinect* e os celulares pessoais, contendo os sistemas operacionais *Android* e *iOS*. Além disso, foi utilizada a plataforma *Raspberry Pi* para o desenvolvimento de um servidor central baseado em *linux* embarcado, responsável por receber e processar todos os comandos provenientes dos usuários, desde um comando de alteração do volume da televisão até o acender ou apagar de uma luz. Quanto aos resultados obtidos ao final do projeto, pode-se dizer que foi alcançada a automação residencial proposta, com uma boa distância suportada pelo sensor infravermelho, o adequado chaveamento de relés, o acesso via *HTTP*, *smartphones* e o *Kinect*.

Palavras-chaves: Automação Residencial, Servidor Embarcado, *Kinect*, *Raspberry Pi*, *Android*, *iOS*.

Abstract

ROSSITI, G. J. ***Implementation of a multiplataform control system and development of a command embedded server for home automation.*** 2014. 70 f. Trabalho de Conclusão de Curso (Graduação) - Escola de Engenharia de São Carlos, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2014.

Use of Technology applications on domestic environment have experienced noted growth rates in the last few years. These applications include residential automation process and central management systems. At the same time, we have seen popularization of Natural User Interface (NUI) concept, and smartphones. This project aims to combine all these technologies on a project of an embedded web server, a command server and devices controllers. Using these components, we intend to control different home appliances using commands from smartphones, computers and gestures interpreted by kinetic. This project uses technologies such as: universal remote control protocol over infrared communication, relay to control electric devices, socket communication to transfer data between Kinect subsystems and command server, Android, iOS. So, we developed a linux server on Raspberry Pi responsible to receive all commands from clients (mobile devices, desktop and web), an example of command is a client message to change channels or turn on the room light. As result, the residential automation proposed was reached, with a appropriate switching relays and optimized access via HTTP, smartphones and Kinect.

Keywords: *Residential Automation, Embedded Server, Kinect, Raspberry Pi, Android, iOS.*

Lista de Figuras

Lista de Tabelas

Lista de Gráficos

Lista de Abreviaturas e Siglas

SoC	***System-on-a-Chip***
RAM	***Random-access Memory***
USB	***Universal Serial Bus***
CPU	***Central Processing Unit***
GPU	***Graphics Processing Unit***
SDRAM	***Synchronous Dynamic Random-access Memory***
SD	***Secure Digital***
SDK	***Software Development Kit***
JVM	***Java Virtual Machine***
API	***Application Programming Interface***
HTTP	***Hypertext Transfer Protocol***
NUI	***Natural User Interface***
GUI	***Graphical User Interface***
GPIO	***General Purpose Input/Output***
LED	***Light-emitting Diode***
IV	**Infravermelho**
RC	***Remote Control***
NA	**Normalmente Aberto**
NF	**Normalmente Fechado**
HD	***Hard Disk***

UART ..... ***Universal Asynchronous Receiver/Transmitter***

I2C ..... ***Inter-Integrated Circuit***

SPI ..... ***Serial Peripheral Interface***

GND ..... ***Ground***

SO ..... **Sistema Operacional**

VGA ..... ***Video Graphics Array***

JSP ..... ***JavaServer Pages***

ADT ..... ***Android Development Tools***

HTML ..... ***Hypertext Markup Language***

JDK ..... ***Java Development Kit***

FTP ..... ***File Transfer Protocol***

TCP ..... ***Transmission Control Protocol***

UDP ..... ***User Datagram Protocol***

HDMI ..... ***High-Definition Multimedia Interface***

IP ..... ***Internet Protocol***

DHCP ..... ***Dynamic Host Configuration Protocol***

SSH ..... ***Secure Shell***

CSS ..... ***Cascading Style Sheets***

WAR ..... ***Web Application Archieve***

LIRC ..... ***Linux Infrared Remote Control***

XML ..... ***eXtensible Markup Language***

Sumário

Capítulo 1: Introdução

1.1 Contextualização e Motivação

A inserção da tecnologia no ambiente residencial está chegando em níveis cada vez mais altos, com processos de automações residências e sistemas de gerenciamento centralizados, visando facilitar a vida dos seus moradores, permitindo controlar diferentes ambientes residências a partir de um *website* ou de um celular, tornando acessível ações desde o abrir do portão até o acionamento de sistemas de iluminação, refrigeração e irrigação das casas. Para o acionamento desses sistemas há inúmeras maneiras, no entanto, alguns métodos mais comuns são o fornecimento de uma tensão de alimentação, para o acendimento de uma lâmpada ou de um aquecedor, e a comunicação por infravermelho, responsáveis por controlar uma televisão ou um ar condicionado.

Durante o projeto, um dos fatores que influenciaram para a escolha dos *smartphones* como um dos possíveis controladores de um ambiente se deu devido à sua ascensão no mercado e na vida das pessoas. Atualmente, no Brasil, observamos um número superior à 70 milhões de *smartphones* de acordo com os estudos realizados pela Morgan Stanley [1]. Foi utilizado também um controlador *HTTP*, devido este ser acessível por inúmeros dispositivos eletrônicos com diferentes sistemas operacionais. Outro produto escolhido foi o *Kinect*, em razão da crescente popularização do conceito NUI (*Natural User Interface*), onde utiliza-se os movimentos do corpo para a realização de comandos, sem que haja a necessidade de um dispositivo físico, com botões ou tela sensível a toque, para ser pressionado. Esta interface é considerada pelos especialistas em tecnologia como sendo a próxima evolução do GUI (*Graphical User Interface*) [2].

Outro conceito já adotado pelo mercado é o de sistemas embarcados, dispositivos que possuem como algumas de suas características uma funcionalidade mais específica e restrições de projeto mais rígidas. [3]

Tendo em vista esse ambiente foi proposto um sistema servidor centralizador de comandos, utilizando a plataforma *Raspberry Pi*, semelhante a um computador, mas com dimensões inferiores e de preço mais acessível.

1.2 Objetivos

O objetivo do sistema proposto é apresentar um servidor responsável por receber os comandos enviados pelos usuários e processa-los de acordo com as funcionalidades disponíveis, sendo elas: o controle de duas luminárias, um ventilador e uma televisão. As requisições poderão ser realizadas via comunicação *HTTP* ou por *Sockets*.

Já do lado do usuário, serão fornecidas diferentes formas de controlar o ambiente, um sistema multiplataformas, todas elas devem apresentar interfaces de fácil utilização e intuitivas. Dentre os controles propostos para o envio de requisições estão os s*martphones* com sistema operacional *Android* ou *iOS*, um *website* e o aparelho *Kinect*.

Tendo isso em mente, é proposta uma forma de realizar a automação de um ambiente residencial, sendo possível controla-lo através de simples botões ou gestos. Além disso, o sistema deve acarretar em facilidade e comodidade aos usuários, não necessitando grandes conhecimentos ou esforços para a sua utilização.

Com isso, deseja-se que os diferentes componentes do sistema sejam capazes de:

· Disponibilizar diferentes plataformas de controle ao usuário final;

· Alterar os canais, volume e estado de uma televisão remotamente, utilizando para isso um emissor de infravermelho conectado a *Raspberry Pi*;

· Obter uma distância considerável para o uso do infravermelho;

· Controlar o estado de dispositivos eletrônicos, realizando o chaveamento de relés através do acionamento dos pinos GPIO da *Raspberry Pi*;

· Realizar toda a comunicação com o programa servidor central via *sockets*.

1.3 Organização

Este documento é dividido em cinco capítulos, sendo o primeiro deles a introdução, contendo uma breve descrição sobre a motivação para a realização do projeto e os objetivos desejados durante a sua implementação. No segundo são apresentadas algumas informações

fundamentais para o entendimento do mesmo. Posteriormente, no terceiro capítulo, são descritas todas as etapas realizadas para o desenvolvimento do projeto. Por fim, no quarto e quinto capítulos são apresentados respectivamente os resultados obtidos e as conclusões retiradas com o estudo desses dados.

Capítulo 2: Fundamentação Teórica

2.1 Considerações Iniciais

Neste capítulo serão apresentados os conceitos básicos necessários para o entendimento do projeto, conceitos esses relacionados ao funcionamento de um controle remoto televisivo, aos relés, à plataforma *Raspberry Pi,* ao sensor *Kinect*, ao sistema operacional *Android* e *iOS*, à utilização do servidor *Apache Tomcat* e ao uso de *Sockets* para comunicação. Todos eles são conceitos utilizados durante toda a realização do projeto.

2.2 Controle Remoto

O primeiro controle remoto projetado foi apresentado, em 1898, pelo cientista croata Nikola Tesla, durante uma exibição pública de um barco controlado por rádio, chamado de *Teleautomaton*. Após o seu surgimento foram criadas tecnologias semelhantes nos anos subsequentes, como por exemplo, o *Telekino*, robô produzido em 1903 pelo engenheiro espanhol Leonardo Torres Quevedo, controlado remotamente por comandos transmitidos por ondas eletromagnéticas. Demais projetos surgiram utilizando também aparelhos de rádio frequência, a maioria deles foram desenvolvidos para o uso militar durante a Primeira e Segunda Guerra Mundial [4].

Atualmente, a maior aplicação desses controles está no uso doméstico, no controle de dispositivos de curto alcance, equipamentos de vídeo e áudio [5]. O controle remoto televisivo desenvolvido inicialmente na década de 1950, por Eugene Polley, funcionário da empresa *Zenith Radio Corporation*, apresentava grandes limitações relacionadas a distância, ao ângulo de transmissão e ao número de comandos (apenas um), sendo utilizada a comunicação via feixes luminosos e células fotoelétricas. Já em 1956 surgiu então o controle com ultrassom, perdurando por duas décadas [6]. Com a constante evolução das televisões e a implementação de novas

funcionalidades foi necessário o desenvolvido de uma nova tecnologia, nomeada de Protocolo ITT de Comunicação Infravermelha, produzida pela *ITT Corporation* em 1977 e utilizado até os dias atuais.

2.2.1 Comunicação via Infravermelho

A utilização da comunicação via infravermelho ocorre principalmente devido a simplicidade dos LEDs IV e ao baixo custo deles. Outro ponto positivo é o seu comprimento de onda, estando entre 700 e 1000 nm, pertencendo a um dos espectros de cores não perceptíveis ao olho humano, tendo em vista que não desejamos ver essa luz, apenas queremos usa-la. Embora não possamos enxerga-la normalmente, podemos utilizar dispositivos como câmeras de celulares e *webcans* para visualiza-las durante a realização de testes [5].

Já do lado negativo, podemos destacar a emissão de luzes infravermelhas por inúmeras outras fontes que liberam calor, podendo causar interferências durante a emissão e recepção dos sinais utilizados em uma comunicação IV.

Para realizar a comunicação é necessário a utilização de um emissor e um receptor de sinais infravermelho, ambos devem trabalhar na mesma frequência. O sinal utilizado para a transmissão deve ser modulado, diminuindo a ocorrência de ruídos. Na Figura 2.1 é possível vê-lo atuando nos dois componentes.

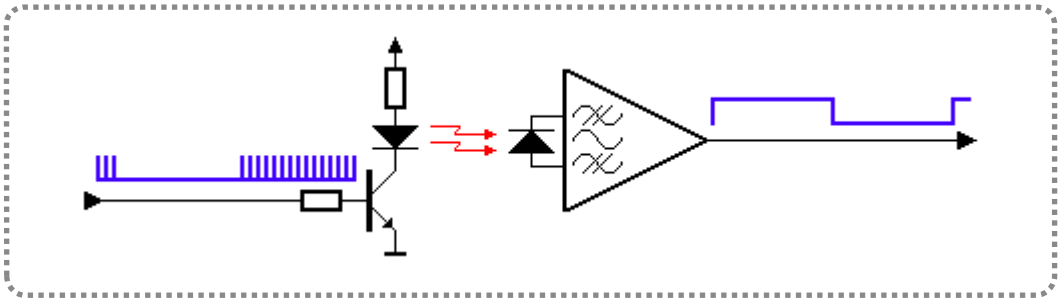

Figura 2.1: Emissor e receptor de sinais infravermelho [5].

Durante a comunicação são utilizados o *"space"* (espaço), estado de repouso do transmissor, e o *"mark"* (marca), estado de pulsação do LED IV. O receptor interpreta os sinais de *"space"* como nível alto, enquanto *"mark"* é considerado nível baixo. No entanto, esses níveis não são necessariamente os bits zeros e uns, essa relação dependerá do protocolo utilizado em diferentes dispositivos [5]. No tópico adiante será descrito um desses protocolos utilizado durante a realização do projeto.

2.2.2 Protocolo RC-5

O protocolo RC-5, utilizado para comunicação via infravermelho, foi desenvolvido pela Philips, em torno de 1980, e apresenta como características principais a codificação de *Manchester* e a frequência da portadora igual a 36kHz [7]. Para transmitir os sinais são utilizadas mensagens com 14 bits cada, conforme apresentado na Figura 2.2 abaixo.

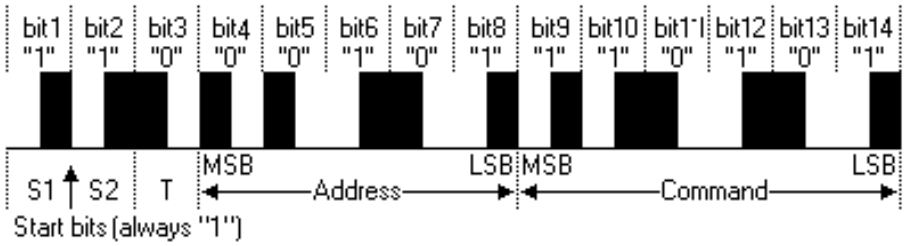

Figura 2.2: Estrutura da mensagem definida pelo Protocolo RC-5 [7].

Os dois primeiros bits utilizados são níveis lógico alto, indicando o início da mensagem. Adiante, o terceiro pulso é chamado de *"toogle bit"*, bit de alternância, utilizado para que o receptor consiga distinguir as ações: manter uma tecla pressionada e pressionar repetidamente a mesma tecla; para isso, cada vez que a tecla é pressionada o bit é invertido. Por fim, são enviados 5 bits de endereçamento do dispositivo seguido então do comando, representado por 6 bits [7]. A lista de alguns dos comandos padrão deste protocolo é apresentado na Tabela 2.1.

Tabela 2.1: Comandos padrões do protocolo RC-5 [7].

COMANDO RC-5	COMANDO TV
$0C - 12	**POWER**
$0D - 13	**MUTE**
$20 - 32	**CHANNEL +**
$21 - 33	**CHANNEL -**
$10 - 16	**VOLUME +**
$11 - 17	**VOLUME -**
$01 - 01	**1**
$02 - 02	**2**
$03 - 03	**3**
$04 - 04	**4**
$05 - 05	**5**
$06 - 06	**6**
$07 - 07	**7**
$08 - 08	**8**
$09 - 09	**9**
$00 - 00	**0**

2.3 Relé

Os relés são dispositivos, operados eletronicamente, que funcionam como interruptores. Há diferentes tipos de relés, no entanto, os mais comuns, utilizados durante o projeto, apresentam comportamento eletromagnético, chaveando mecanicamente os seus contatos [8]. A sua utilização se dá principalmente para o controle de circuitos externos de grandes correntes a partir de pequenas correntes, por exemplo, pode-se controlar um dispositivo que utiliza a rede elétrica convencional, 110 ou 220 volts, utilizando pinos GPIO convencionais das plataformas *Raspberry Pi* ou *Arduino*.

O funcionamento do relé usado durante o projeto é simples, ele apresenta três contatos principais de chaveamento, quando uma corrente circula por sua bobina é criado um campo

magnético que atrai um dos contatos, fechando o circuito do contato “Comum” (C) com o “Normalmente Aberto” (NA). Quando a corrente é cessada, retorna-se ao funcionamento normal do circuito, chaveando para o contato “Normalmente Fechado” (NF). Os dois estados descritos são apresentados nas Figuras 2.3 e 2.4 adiante.

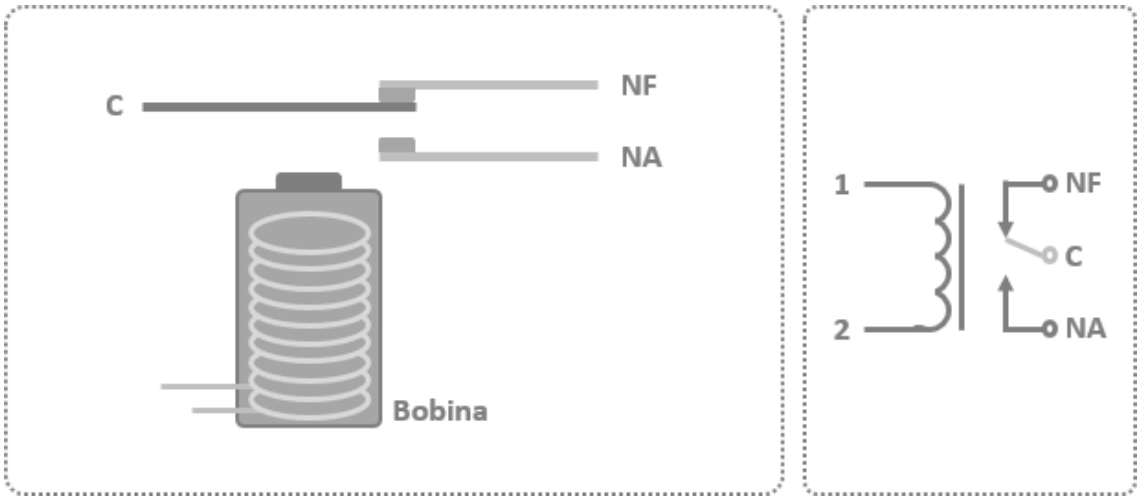

Figura 2.3: Bobina sem tensão aplicada nos seus terminais, chaveando para o contato NF.

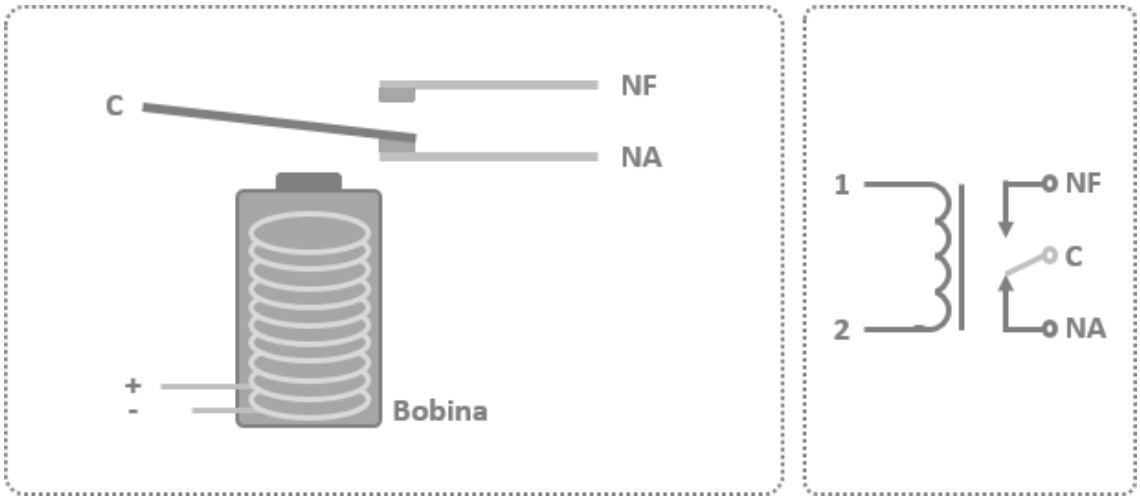

Figura 2.4: Bobina com tensão aplicada nos seus terminais, chaveando para o contato NA.

Na etapa de “desenergização” do relé ocorre o surgimento de uma tensão com polaridade oposta à aquela que criou o campo magnético na bobina, esta tensão pode atingir valores altos e dependendo do circuito implementado pode ocorrer a perda de determinados componentes. Para solucionar este problema é utilizado um diodo, polarizado inversamente em relação a tensão de ativação que ativa o relé, com isso, no momento que surge a tensão oposta nos extremos da bobina o diodo polarizado passa a apresentar baixa resistência, absorvendo a energia gerada [8]. Esta técnica de proteção do circuito de acionamento é apresentada na Figura 2.5.

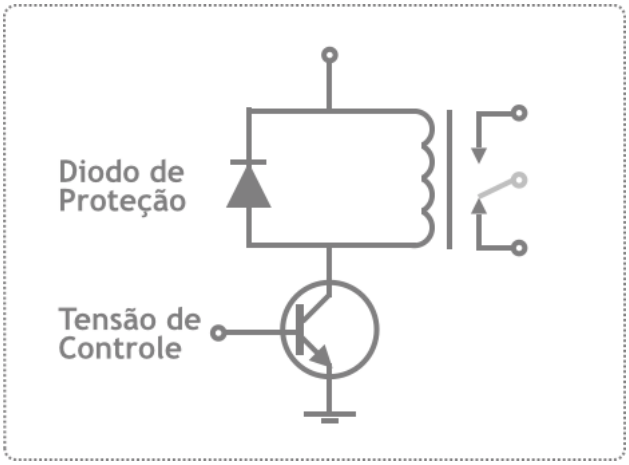

Figura 2.5: Diodo de proteção do circuito de acionamento [8].

As principais vantagens da utilização do relé advém do relativo isolamento do circuito de controle e do circuito chaveado, permitindo trabalhar com tensões completamente diferente. Já dentre as desvantagens, está o seu gradativo desgaste, sendo este um componente mecânico, o que pode tornar a vida útil deste "interruptor" inferior à de outros.

2.4 *Raspberry Pi*

O projeto educacional denominado *Raspberry Pi* possui sua base no Reino Unido e foi desenvolvido inicialmente com o objetivo de disponibilizar um computador simples e de baixo custo para crianças do mundo todo, facilitando assim o ensino da ciência da computação básica em diferentes ambientes escolares. Suas proporções são semelhantes a de um cartão de crédito e podem ser utilizados para projetos de eletrônica e para algumas funcionalidades básicas de um computador pessoal [9].

Atualmente estão disponíveis no mercado dois modelos da plataforma *Raspberry Pi*, o Modelo A e Modelo B, ambos possuem características semelhantes, no entanto, o Modelo B apresenta um desempenho superior em determinados aspectos, por exemplo, a presença de uma porta *Ethernet* e duas portas USB. Algumas dessas características podem ser observadas na Tabela 2.2 e Figura 2.6 adiante.

Tabela 2.2: Especificações dos distintos modelos de *Raspberry Pi* [9].

	MODELO A	MODELO B
Preço	US$25,00	US$35,00
SoC	Broadcom BCM2835 (CPU, GPU, DSP e SDRAM)	
CPU	700 MHz ARM1176JZF-S core (ARM11 family)	
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 decodificador h.264/MPEG-4 AVC	
Memória (SDRAM)	256 MB (compartilhada GPU)	512 MB (compartilhada GPU)
Portas USB 2.0	1	2
Saídas de Vídeo	RCA Composto (PAL & NTSC), HDMI (rev 1.3 & 1.4), Painéis LCD via DSI, 14 resoluções HDMI de 640x350 à 1920x1200 mais diversos padrões PAL & NTSC	
Saídas de Áudio	Conector de 3.5 mm, HDMI	
Armazenamento Onboard	SD, MMC, slot para cartão SDIO	
Rede onboard	Nenhuma	10/100 Ethernet (RJ45)
Periféricos de baixo-nível	8 x GPIO, UART, I2C, SPI com dois seletores de chip, +3,3V, +5V, terra	
Potência	500 mA (2.5W)	700 mA (3,5W)
Fonte de energia	5 volt via MicroUSB ou header GPIO	
Tamanho	85,60 mm x 56,00 mm x 21,00 mm	
Peso	45 g	

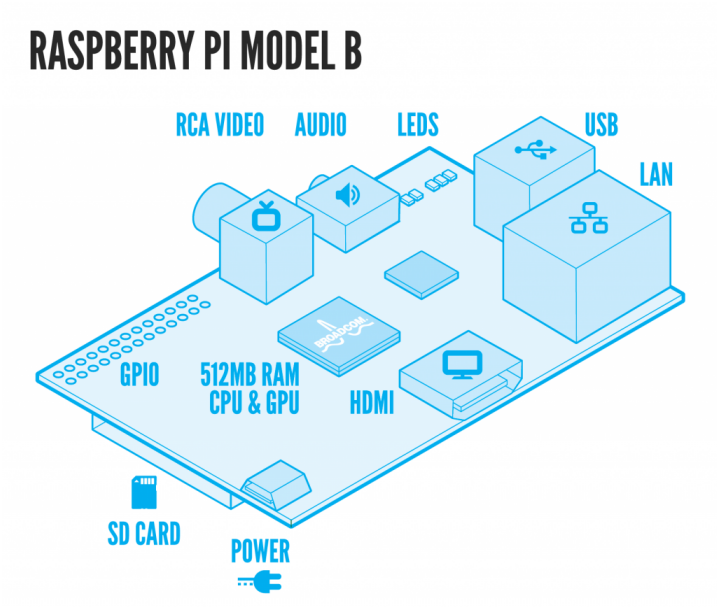

Figura 2.6: *Raspberry Pi*, Modelo B [9].

Os dois modelos apresentam portas USB (*Universal Serial Bus*) limitadas, projetadas para fornecer alimentação à dispositivos de entrada de baixo consumo, próximos de 100mA, por exemplo, teclados e *mouses*. Tendo em vista essa limitação, outros dispositivos como Adaptadores Wi-Fi, Pen-Drives e HDs externos podem não funcionar corretamente devido a necessidade de uma corrente superior [9].

Além disso, na área superior da *Raspberry Pi* há 26 pinos, organizados em duas colunas de 13 pinos cada, responsáveis por fornecer ao usuário 8 GPIO, UART, I2C, SPI, +3.3V, +5V e GND, conforme pode ser observado na Figura 2.7. Em relação aos pinos de propósitos gerais, *General-Purpose Input/Output* (GPIO), pode-se expandir o seu número indefinidamente fazendo uso dos pinos de I2C e SPI [9], já para utilizar esses pinos eles devem ser programados via *softwares* para que sejam tratados como pinos de entrada ou saída, o nível de tensão permitido neles é igual a 3.3V, não sendo tolerados valores de 5V.

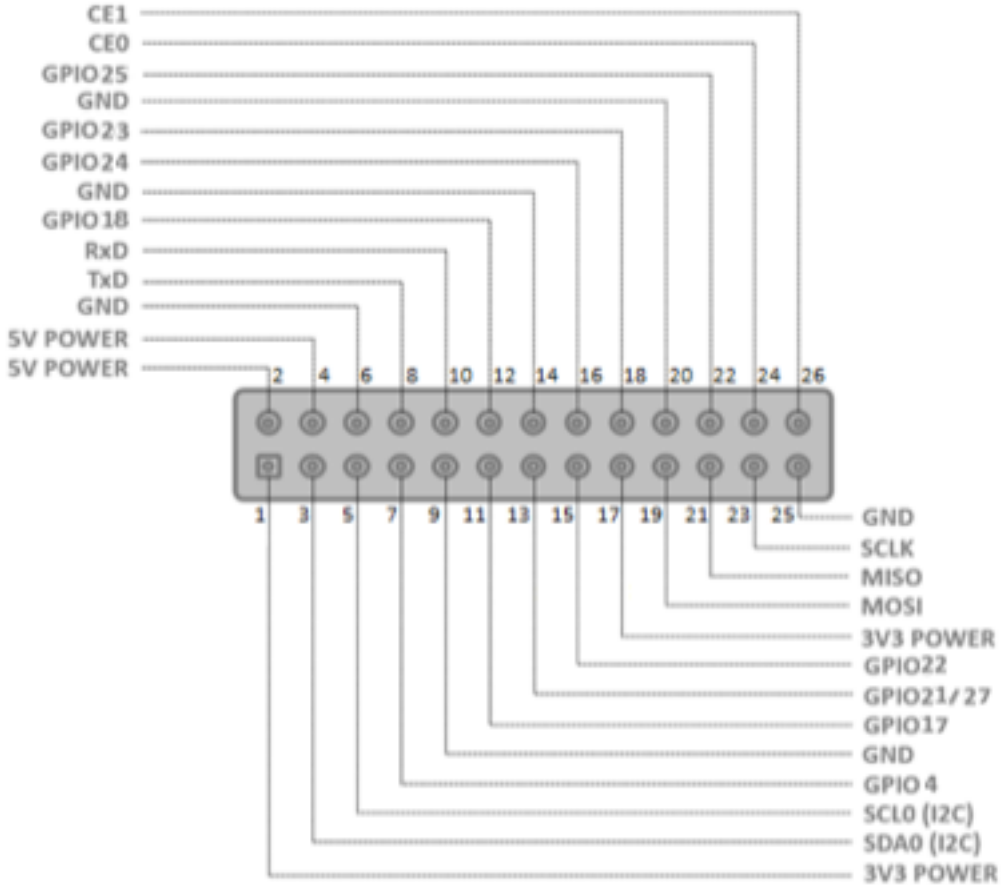

Figura 2.7: Pinos de entrada e saída da *Raspberry Pi*.

Como já dito, a *Raspberry Pi* possui inúmeras semelhanças com um computador pessoal, outro ponto em comum é a necessidade da instalação de um Sistema Operacional para inicializar a sua utilização. Na página oficial do projeto são disponibilizados diferentes sistemas operacionais, por exemplo, *Raspian Wheezy, Pidora, RISC OS, RaspBMC, Arch e OpenELEC*. Para a instalação desses sistemas operacionais são necessárias as realizações de determinadas etapas utilizando o cartão SD (FAT32), todas elas são descritas de forma sucinta nos manuais presentes no site da companhia [10]. Finalizada a instalação do SO desejado a plataforma estará pronta para a implementação de projetos, à exemplo do apresentado neste documento.

2.5 *Kinect*

O *Kinect*, apresentado na Figura 2.8, é um sensor de movimentos da *Microsoft*, desenvolvido juntamente com a *PrimeSense*, lançado no mercado internacional em novembro de 2010 e criado originalmente para ser utilizado junto ao *video-game Xbox 360*. Ele apresenta um conjunto de sensores os quais permitem ao usuário interagir com os jogos eletrônicos apenas realizando movimentos de corpo ou utilizando a própria fala, não sendo necessária a utilização de controles físicos (*joysticks*) [11].

Figura 2.8: Sensor Kinect [11].

Para conseguir desempenhar sua principal função, tornar o humano um controle por completo, o *Kinect* faz uso de diferentes sensores conforme apresentado na Figura 2.9, sendo as principais estruturas dele dividida em 6 componentes:

1. **Microfones Multi-Matriz:** um conjunto de 4 microfones utilizados para identificar sons e vozes no ambiente, contendo também um filtro para tratar os possíveis ruídos;

2. **Emissor Infravermelho:** projeta vários feixes de luz infravermelha no ambiente, ao atingir alguma superfície o padrão torna-se distorcido. Distorção esta que é percebida pela câmera de profundidade;

3. **Câmera de Profundidade:** analisa os padrões de infravermelho emitidos no ambiente para construir um mapa 3-D contendo todos os objetos e pessoas presentes nele;

4. **Motor de Inclinação:** ajusta o campo de visão do sensor, podendo inclinar o sensor para cima ou para baixo;

5. **Cabo USB:** responsável por transmitir os dados para o dispositivo desejado, sendo que a comunicação realizada não é criptografada;

6. **Câmera VGA Colorida:** funciona como uma *webcam*, capturando uma imagem de vídeo colorida. Tornando possível detectar e diferenciar as pessoas presentes no ambiente.

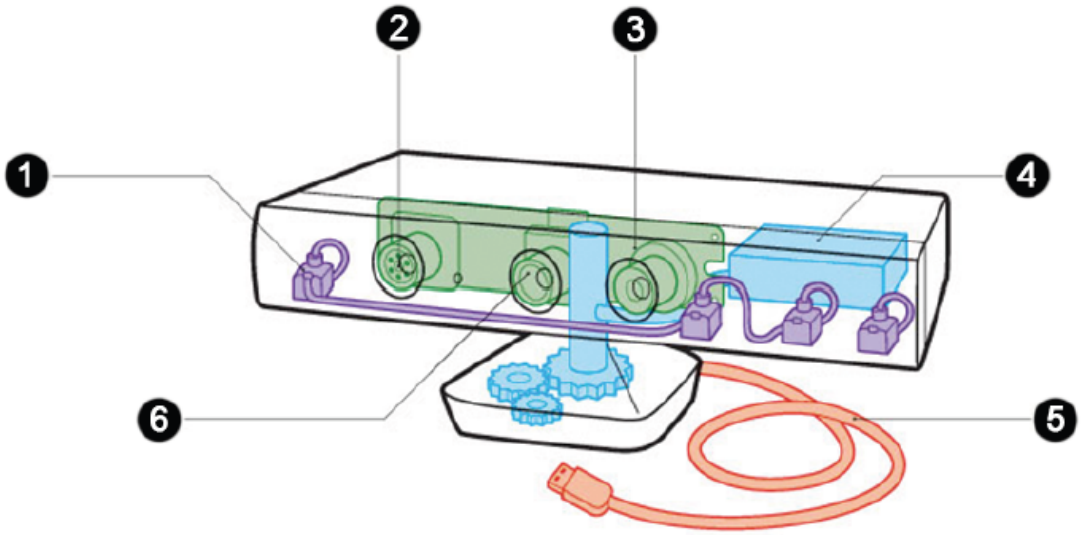

Figura 2.9: Estrutura básica do Kinect [12].

Para produzir um mapa 3-D o aparelho cria inicialmente uma nuvem de pontos a partir da matriz de feixes infravermelhos emitidos no ambiente, transformando esses pontos em uma imagem monocromática com diferentes tonalidades, conforme apresentado nas Figuras 2.10 e 2.11, onde os *pixels* mais escuros apresentam uma distância maior em relação ao sensor, enquanto os *pixels* mais claros se encontram mais próximos a ele [13]. No entanto, o sensor possui uma certa limitação para visualizar objetos muito próximos, sendo necessária uma certa distância para o seu funcionamento correto.

Figura 2.10: Imagem obtida pela câmera VGA colorida [13].

Figura 2.11: Imagem monocromática produzida pela câmera de profundidade [13].

Devido ao seu grande potencial sensorial dentro de um ambiente, o *Kinect* vem ganhando novas utilidades fora da área de entretenimento originária. Algumas das configurações padrões do sensor são apresentadas na Tabela 2.3 adiante. Para o desenvolvimento de novas aplicações, utilizando o dispositivo, há alguns projetos auxiliares, como por exemplo o *OpenNI* descrito no próximo tópico.

Tabela 2.3: Características do sensor *Kinect* do *Xbox* [14].

Campo de Visão	57,5º horizontal, 43,5º vertical
Distância	0,8m -> 4,0m
Câmera VGA	640x480x24 bpp 4:3 RGB @ 30fps 640x480x16 bpp 4:3 YUV @15fps
Câmera de Profundidade	320x240x16 bpp, 13-bit depth
Registro	Cor <-> Profundidade
Captura do Áudio	4 microfones, 48Hz audio
Caminho dos Dados	USB 2.0
Latência	~90ms com processamento
Motor de Inclinação	Vertical apenas

2.5.1 *Open Natural Interection*

A organização nomeada *OpenNI* (*Open Natural Interaction*) foi criada no ano de 2010 e não possui fins lucrativos, disponibilizando códigos abertos para o desenvolvimento de aplicações relacionadas à Interface Natural do Usuário (NUI) para o controle de dispositivos. Um dos principais membros dessa organização é a *PrimeSense*, companhia que desenvolveu juntamente à *Microsoft* o *Kinect* [15].

No *website* da companhia são disponibilizados o *OpenNI Framework*, responsável por prover um conjunto de APIs (*Application Programming Interface*). Todo o conteúdo utilizado é implementado em C++ e faz uso de algumas pendências externas, como por exemplo, *OpenGL* e

libjpeg, para a renderização e outras atividades relacionadas ao processamento de imagens. Há suporte para a sua instalação em diferentes sistemas operacionais. No site são descritas etapas de instalação para o *Windows*, *Linux* e *Mac*.

Uma série de exemplos iniciais são fornecidos com o pacote original, dentre eles destacam-se os códigos que realizam a identificação de diferentes usuários, o rastreamento das mãos e o reconhecimento de gestos.

Atualmente, algumas mudanças estão a ocorrer na organização *OpenNI*, tendo em vista que a organização foi comprada pela *Apple* no final do ano 2013. Após essa aquisição foi anunciado o fechamento do *website* que disponibilizava o seu conteúdo de desenvolvimento, em Abril de 2014 [16].

2.6 *Android*

O *Android* é um sistema operacional baseado no *Linux Kernel* desenvolvido primariamente para dispositivos *touchscreen* como *smartphones* e *tablets*. Trata-se de um sistema *open source* liberado pela *Google* sob a licença *Apache*, permitindo assim que o software seja modificado livremente e distribuído por diversas fábricas de aparelhos eletrônicos, operadoras sem fio e desenvolvedores entusiastas [17].

O desenvolvimento de softwares para o *Android* é facilitado devido a presença de inúmeras bibliotecas em *java* e um bom ambiente de desenvolvimento de software, comumente chamado de SDK. As bibliotecas distribuídas fazem com que a manipulação dos recursos existentes dos sistemas seja realizada de maneira mais simples, permitindo assim controlar ações de botões, toques, escritas e apresentações.

Para o desenvolvimento de aplicativos há inúmeros ambientes de desenvolvimento, um dos mais utilizados é o *plugin* gratuito do *Eclipse IDE* chamado *Android Development Tools* (ADT), nele é possível gerenciar diferentes projetos, visualizar a interface do usuário, depurar os aplicativos e simular um sistema genérico que utiliza o *Android*. Pode-se também instalar os aplicativos desenvolvidos em um dispositivo contendo o sistema operacional, para isso é necessário apenas conectá-lo ao computador via cabo USB.

Em maio de 2013, na conferência da Google I/O, foi anunciado a disponibilidade de mais de 900 mil aplicativos na loja online *Google Play*, além disso, foi revelado um número superior à 48 bilhões de *downloads* desses aplicativos [18]. Atualmente, o número de aplicações disponíveis na loja já é superior a 1,2 milhões [19]. Os valores apresentados refletem o elevado número de dispositivos que fazem uso desse sistema operacional e a sua importância no atual mercado. O seu crescimento nos últimos anos pode ser observado no Gráfico 2.1.

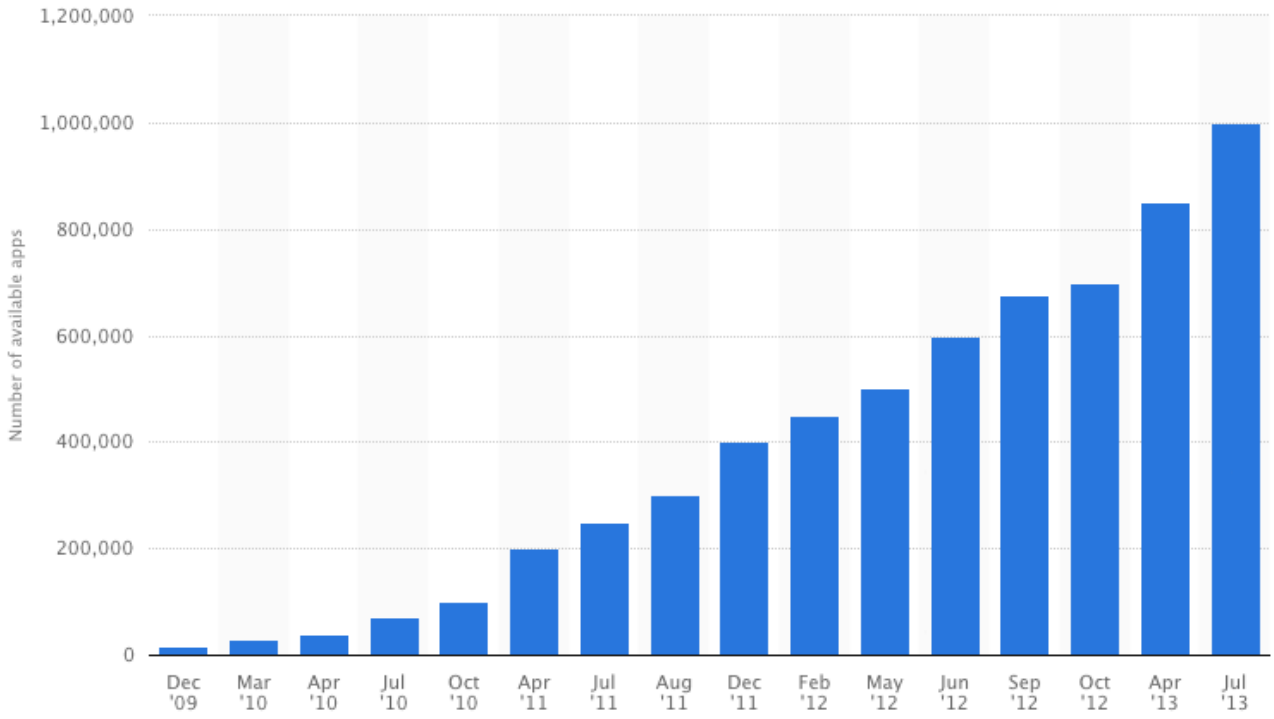

Gráfico 2.1: Número de aplicativos na *Google Play Store* de 12/2009 à 07/2013 [19].

2.7 *iOS*

Originalmente chamado de *iPhone OS*, o sistema operacional exclusivo da *Apple* foi lançado em 2008, disponibilizado inicialmente apenas para o *iPhone*. Com o passar dos anos ele foi estendido para suportar outros dispositivos móveis da empresa, como o *iPod Touch*, *iPad*, *iPad Mini* e *Apple TV*, recebendo assim um nome mais genérico em 2010, *iOS* [20].

O *iOS* é derivado do popular *Mac OS X*, apresentando interfaces semelhantes e o suporte a gestos multi-toque [20].

A linguagem de programação utilizada para o desenvolvimento de aplicativos destinados aos dispositivos *Apple* é o *Objective-C*.

Para criar aplicativos destinados aos dispositivos móveis da *Apple* há algumas barreiras. A primeira delas é a obrigatoriedade de se utilizar um *Mac*, executando o *Mac OS X*. A segunda delas é a necessidade do cadastro de uma conta de desenvolvedor gratuita, realizado no *website* de desenvolvedores da *Apple*. Passada essas duas etapas deve-se instalar a ferramenta *Xcode*, disponibilizada na *Apple Store*, trata-se de um ambiente integrado para o desenvolvimento de softwares. Por fim, para a realização de testes dos aplicativos criados encontra-se a maior das barreiras, a indisponibilidade da realização de testes em dispositivos físicos, sendo necessário para isso o registro no *iOS Developer Program*, um programa pago de US$99, com a duração de um ano. No entanto, há uma solução parcial para a maior das dificuldades, junto ao *Xcode* é disponibilizado um simulador do *iPhone* e *iPad*, permitindo a realização de aplicativos de graça, embora de maneira limitada [21].

O sistema operacional *Android* ainda é o maior representante no mercado de dispositivos móveis, sendo o *iOS* o seu maior concorrente e possuindo números também incríveis, como por exemplo, o número de aplicativos disponibilizados em sua loja, a *App Store*, superior à 1 milhão [22]. Quanto ao número de usuários dos dispositivos *Apple*, há uma diferença maior em relação aos de proprietários de aparelhos contendo o *Android*, este valor é um reflexo das vendas mundiais de *smartphones*, Gráfico 2.2.

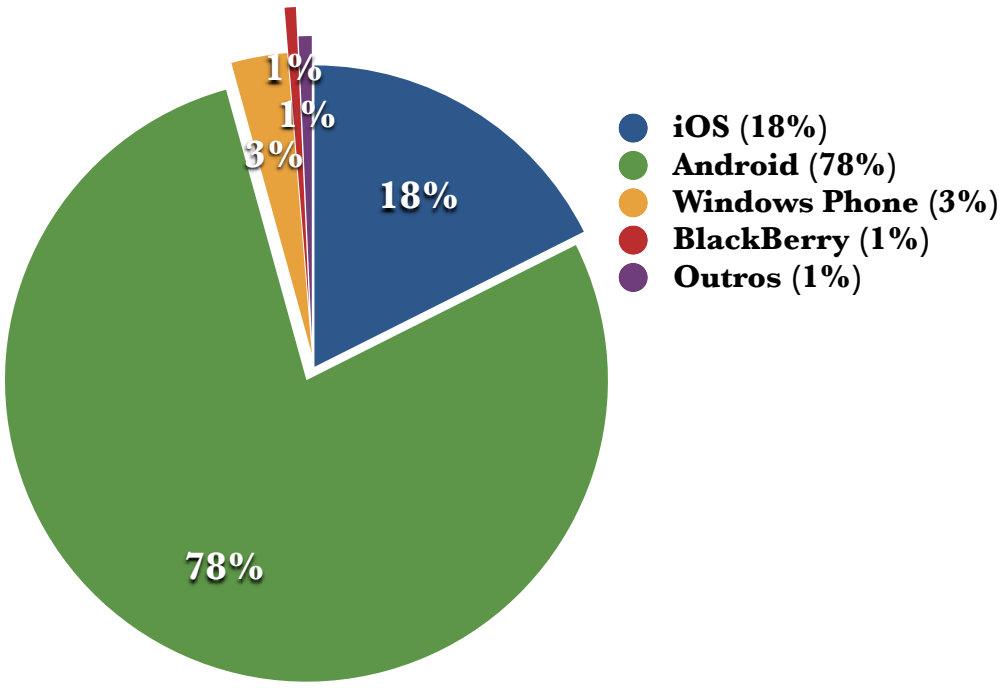

Gráfico 2.2: Vendas de *smartphones* no mundo, 4Q de 2013 [23].

2.8 *Apache Tomcat*

O servidor *Apache Tomcat* é um projeto *Open Source*, baseado em *Java*, criado para executar aplicações *Web* que utilizam as tecnologias *Java Servlet* e *JavaServer Pages* (JSP). A primeira tecnologia trata-se de uma classe empregada para estender as funcionalidades de um servidor, proporcionando a adição de um conteúdo dinâmico em um servidor web, utilizando para isso a plataforma *Java*, já a segunda é uma linguagem de *script* com especificação aberta, tendo também como objetivo a geração de um conteúdo dinâmico para páginas web, substituindo a linguagem HTML (*HyperText Markup Language*). O *Tomcat* apresenta características de um servidor de aplicações, tendo a versatilidade de atuar também como servido web, provendo um servidor HTTP puramente em Java, e a possibilidade de trabalhar integrado a um servidor *web* dedicado como o *Apache* [24].

Antigamente o servidor era um subprojeto da *Apache-Jakarta*, no entanto, devido ao seu aumento de popularidade ele tornou-se um projeto *Apache* separado, apoiado por um grupo de voluntários da comunidade *Java* de código aberto [24].

Há distintas versões do servidor disponíveis para *download* na página do projeto. No entanto, as anteriores à 5.5 já não recebem suporte. Outro fator importante a se observar é a compatibilidade com as versões das APIs e da JDK utilizada, na Tabela 2.4 adiante são apresentados esses dados.

Tabela 2.4: Versões do servidor *Apache Tomcat* e suas compatibilidades [24].

APACHE TOMCAT	SERVLET API	JSP API	JDK
7.0	**3.0**	**2.2**	**1.6**
6.0	**2.5**	**2.1**	**1.5**
5.5	**2.4**	**2.0**	**1.4**
4.1	**2.3**	**1.2**	**1.3**
3.0	**2.2**	**1.1**	**1.1**

Ao instalar o servidor *Tomcat* outros pacotes são fornecidos junto com ele, um deles é o *Tomcat Manager*, acessível em qualquer navegador e responsável por prover funcionalidades

básicas de gerenciamento dos serviços *webs* executados pelo servidor. As principais utilidades são a visualização dos relatórios, a instalação, inicialização, paralisação e remoção das aplicações. Além disso, utilizando o *Manager* pode-se realizar o *deploy* das aplicações *webs* nos servidores, locais e remotos, de maneira simples, tendo em vista que todos os comandos são transmitidos utilizando o protocolo HTTP, sem a necessidade de obter acesso FTP (*File Transfer Protocol*) [24].

Após desenvolver e compilar a aplicação, seguindo as etapas de instalação, os arquivos serão todos armazenados dentro da pasta "*Tomcat/webapps/*", contendo todas as classes java e arquivos de definição de estilos.

2.9 *Sockets*

Socket é uma API (*Application Programming Interface*) para a comunicação entre duas portas. Nele é definido um mecanismo de troca de dados entre dois ou mais processos, processos estes que podem estar em execução na mesma máquina (local) ou em uma rede com máquinas distintas (remoto) [25].

As implementações mais utilizadas de comunicação por *sockets* são:

• *Stream Sockets*: utiliza protocolo TCP com comunicação confiável (ausência de erros) e orientada a conexão;

• *Datagram Sockets*: utiliza protocolo UDP com comunicação não-confiável e não-orientada a conexão.

Durante o projeto foi dado enfoque em *stream sockets* por possuir comunicação confiável.

Para realizar a conexão por *stream socket* é necessário inicialmente que o servidor ligue-se a uma porta usando a função *bind( )*, após essa etapa ele deverá escutar novas requisições de conexão com a função *listen( )*, quando o cliente requisitar uma conexão usando *connect( )* ela deverá ser aceita pelo servidor utilizando a função *accept( )*, estabelecendo assim a conexão entre cliente-servidor. Após isso a comunicação entre eles deverá ser feita com as funções *write( )* e *read( )*, as quais enviam dados e leem dados respectivamente. Para encerrar a comunicação é chamada a função *close( )*. As chamadas de funções descritas podem ser visualizadas no esquemático apresentado na Figura 2.12 a seguir [25].

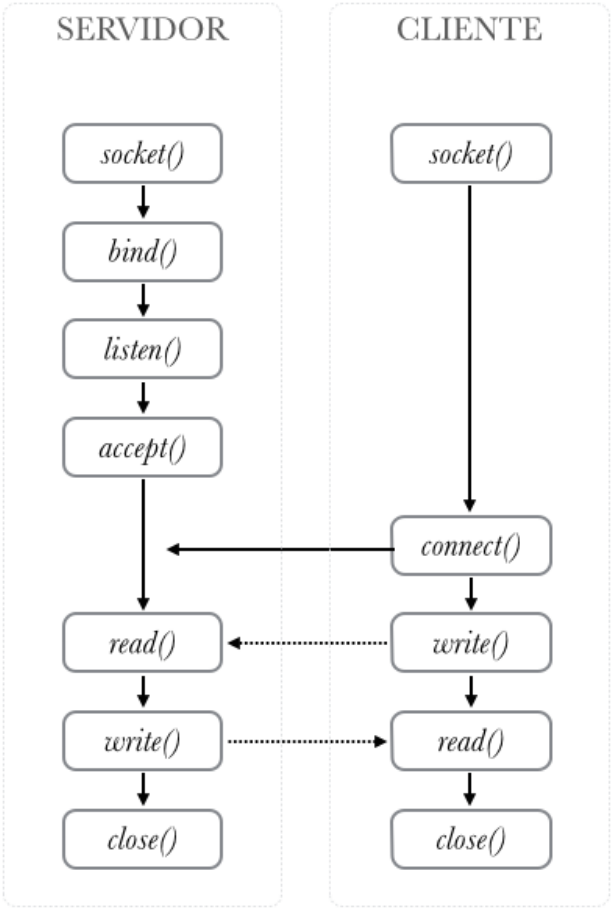

Figura 2.12: Fluxograma da comunicação utilizando *Stream Sockets* [25].

2.10 Considerações Finais

Utilizando os conceitos mencionados nos tópicos anteriores foram definidas algumas das características do projeto, como a utilização da *Raspberry Pi*, Modelo B, usada devido ao desempenho superior em relação ao Modelo A, fato este que permite o funcionamento correto do sistema e a extensão de funcionalidades em trabalhos futuros. Em relação aos ambientes de desenvolvimento dos aplicativos implementados para os *smartphones*, foi utilizado o *Android Development Tools*, para o *Android*, e o *Xcode*, para o *iOS*, ambos de fácil utilização. Demais temas destacados descreveram o funcionamento básico dos demais componentes presentes no projeto, seja apresentando as etapas necessárias para a realização da comunicação por *sockets* e

infravermelho entre os dispositivos, a forma que o *Kinect* trabalha para a aquisição de dados, ou o modo que o relé opera para chavear os circuitos. Adiante serão apresentadas maiores informações sobre o uso de cada componente durante o projeto.

Capítulo 3: Desenvolvimento

3.1 Considerações Iniciais

O projeto consiste principalmente em uma aplicação com arquitetura do tipo Cliente/ Servidor, na qual o servidor deverá ser executado na *Raspberry Pi* e os clientes serão aplicativos desenvolvidos para os *smartphones* com os sistema operacionais *Android* e *iOS,* um programa criado para a utilização da interface natural do *Kinect* e um servidor de acesso *HTTP*. Em todos os casos será utilizada a comunicação via *sockets* para a troca de mensagens.

O servidor é incumbido de atender requisições dos usuários que desejam ligar ou desligar uma luminária, alterar o canal de uma televisão, ajustar o seu volume ou alterar o seu estado de funcionamento. As solicitações podem ser enviadas ao servidor por diferentes dispositivos de controle. No entanto, ambas as formas devem ser de fácil utilização e intuitivas. A Figura 3.1 apresenta um diagrama de blocos simplificado da arquitetura do sistema implementado neste projeto. Já na Figura 3.2 são descritos todos os componentes utilizados, com uma apresentação mais didática.

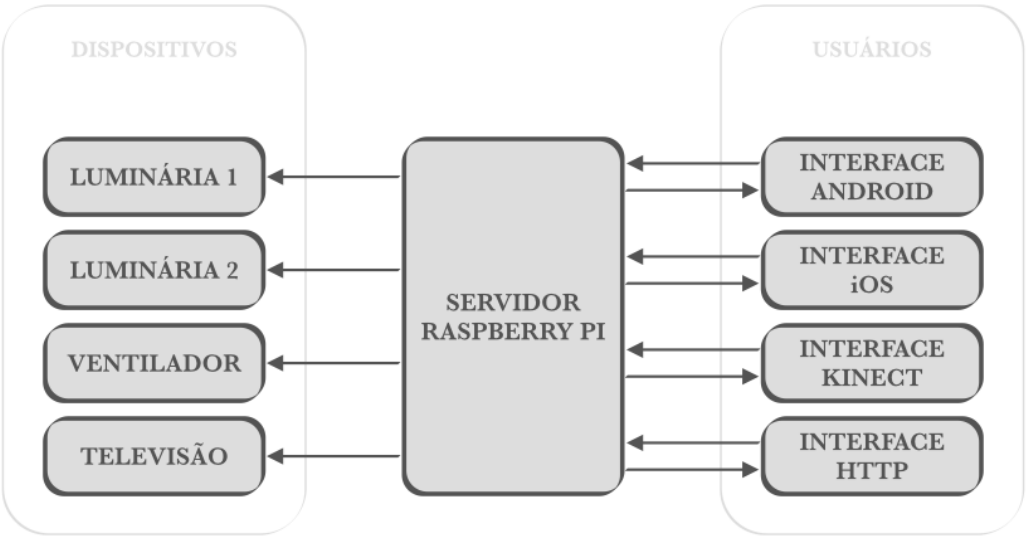

Figura 3.1: Arquitetura simplificada do sistema implementado.

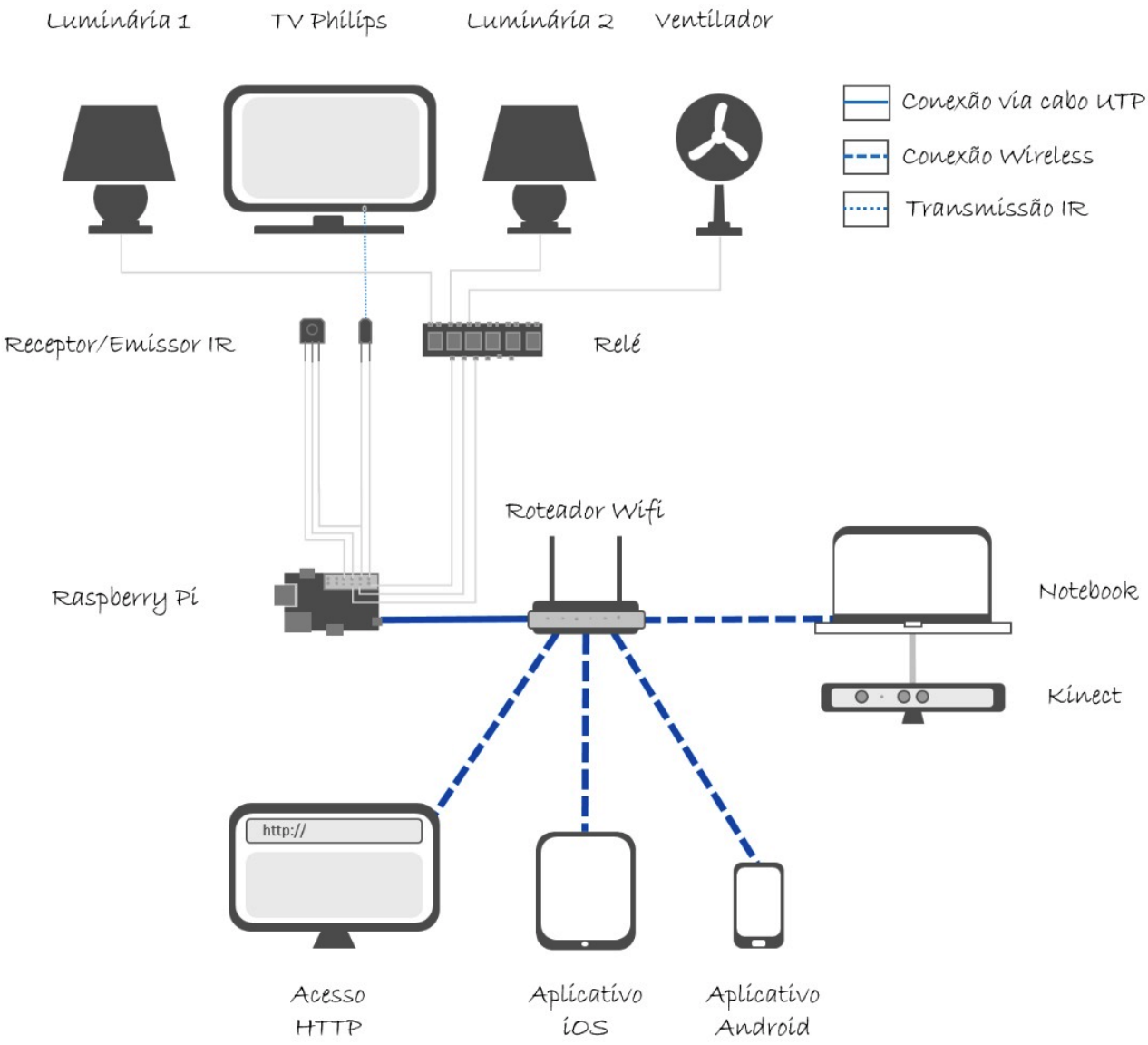

Figura 3.2: Arquitetura do sistema implementado, com todos os componentes.

Os capítulos a seguir descrevem detalhadamente as diferentes etapas de desenvolvimento, de cada um dos componentes do sistema, assim como o processo de integração entre eles.

3.2 Servidor Raspberry Pi

O servidor foi projetado levando em consideração algumas restrições impostas pela plataforma embarcada escolhida, por exemplo o processamento limitado, impossibilitando a utilização do *Kinect* de forma direta, e a baixa corrente fornecida na saída USB.

Quando estabelecida uma conexão com o servidor, todas as requisições serão direcionadas a um mesmo programa centralizador, incumbido de verificar o comando desejado e processá-los, utilizando para isso os pinos de GPIO disponíveis.

3.2.1 Instalação do Sistema Operacional

Na página oficial do projeto são disponibilizados diferentes sistemas operacionais para a instalação na *Raspberry Pi*. O sistema escolhido para o desenvolvimento do projeto foi o *Raspian Wheezy*, baseado em *Debian* e otimizado para o seu *hardware*. A sua utilização ocorreu devido a grande quantidade de pacotes fornecidos com o SO e a interface de fácil utilização.

Para a sua instalação foi necessário seguir uma série de passos, os primeiros deles foram a formatação de uma cartão SD, para o formato FAT32, e o *download* da imagem do *Raspian Wheezy* na página do projeto. Por fim, é necessário inserir o SO no cartão, sendo então utilizado o programa *Win32DiskImager,* executado em uma máquina contendo *Windows*. Com o programa inicializado deve-se selecionar a imagem baixada e o *driver* contendo o cartão SD, para que seja então realizado o processo de transferência ao escolher a opção *write* [10].

Com o sistema operacional instalado, o próximo passo é a primeira inicialização da *Raspberry Pi*, sendo necessário para isso a placa de desenvolvimento, uma fonte de energia, um monitor HDMI, um teclado e o cartão pré-preparado. Com todos componentes em mãos deve-se inserir o cartão SD na *Raspberry Pi*, e ligá-la na fonte de energia, com isso será apresentada uma tela de configuração inicial no monitor, nela serão alterados o tamanho do sistema de arquivos, para que seja utilizado o cartão inteiro, a localidade internacional e o horário.

Finalizada todas essas etapas a plataforma encontra-se pronta para receber a instalação de novos pacotes, utilizados para o desenvolvimento do projeto, conforme descritos nos próximos tópicos.

3.2.2 Interfaces de Rede

Uma rede dedicada foi criada para o desenvolvimento do projeto, para isso foi utilizado um roteador Wi-Fi Jensen Scandinavia, Modelo AL29150 v4, configurado com endereço IP 192.168.0.1 e máscara de rede 255.255.255.0. Obtendo o acesso à essa rede, a plataforma de desenvolvimento *Raspberry Pi* teve as suas interfaces de rede modificadas para um melhor desempenho.

Inicialmente foi configurada a interface *eth0*, para que esta passasse a possuir um endereço IP estático, diferente do padrão, onde é realizado o processo de DHCP. Visando realizar essa mudança foi alterado o arquivo *interfaces*, localizado no diretório *"/etc/network/"*. Dentre as modificações estão o comando *static*, o novo endereço IP (192.168.0.100), a máscara de rede (255.255.255.0), a rede a qual se encontra (192.168.0.0), o endereço de *broadcast* (192.168.0.255) e de *gateway* (192.168.0.1). As adições podem ser observadas na Figura 3.3.

```
iface lo inet loopback

iface eth0 inet static

address 192.168.0.200
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1
```

Figura 3.3: Configurações da interface *eth0*.

Finalizada a configuração da primeira interface, todo o acesso foi realizado de forma remota, via SSH, utilizando o programa *PuTTy*, no Sistema OS X 10.9.2. Para realizar essa comunicação é necessário apenas informar o endereço IP do dispositivo que deseja acessar e a porta, por padrão SSH é realizado na porta 22, conforme apresentado na Figura 3.4.

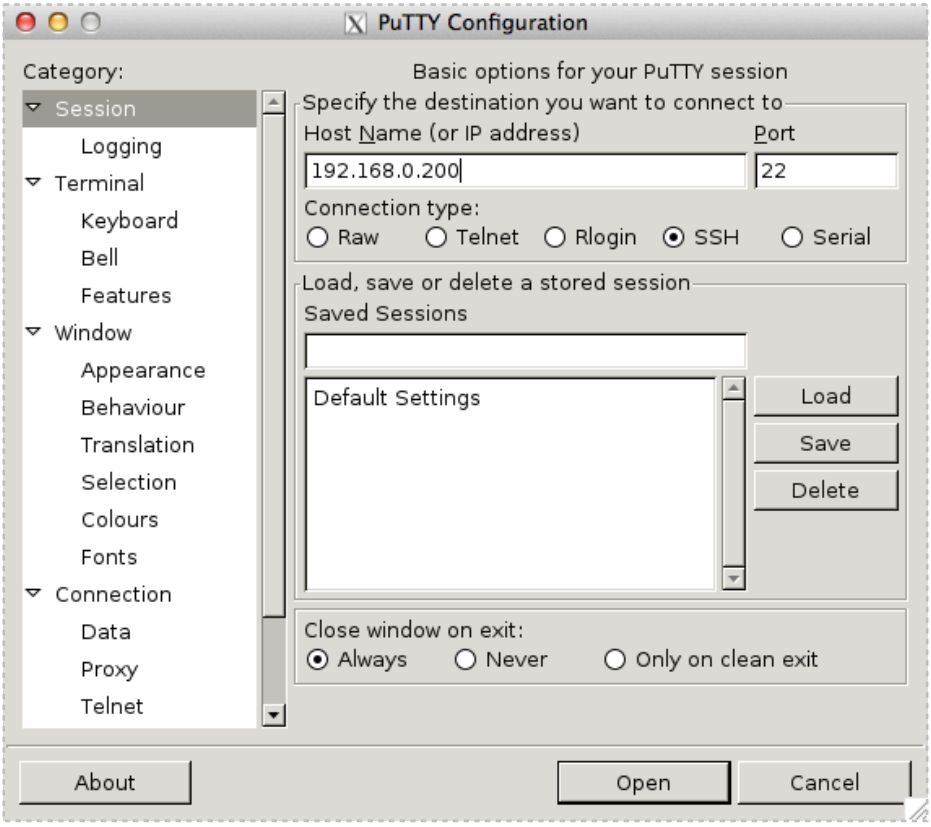

Figura 3.4: Interface do programa PuTTy utilizado para acesso remoto.

O *PuTTy* não é um programa padrão dos sistemas OS X, no entanto, a sua instalação é simples, para quem possui instalado o *MacPorts* [26] basta abrir um terminal e inserir o comando *"sudo port install putty"*. Finalizada a instalação, para executá-lo basta utilizar o comando *"putty"* no terminal.

Outra interface criada foi a *wlan0*, responsável pela comunicação utilizando um Módulo Wi-Fi inserido na entrada USB. Para realizar a sua configuração foi necessária a realização de algumas etapas extras além da simples edição do arquivo *interfaces*. A primeira delas foi a instalação de um pacote de suporte, *wpasupplicant*, instalado a partir do comando *"sudo apt-get install wpasupplicant"*. Seguindo o processo foi criado um arquivo de nome *wpa_supplicant.conf*, localizado no diretório *"/etc/wpa_supplicant/"*, contendo as informações para a conexão Wi-Fi com o roteador, como o nome da rede e a senha, conforme apresentado na Figura 3.5.

Figura 3.5: Configurações do arquivo wpa_supplicant.conf.

Seguindo o modelo de edição da interface *eth0*, também foi necessário alterar o arquivo *interfaces*, permitindo que a interface *wlan0* também apresente um endereço IP estático (192.168.0.100) e carregue as configurações necessárias para a comunicação com o roteador via Wi-Fi. As adições encontram-se na Figura 3.6.

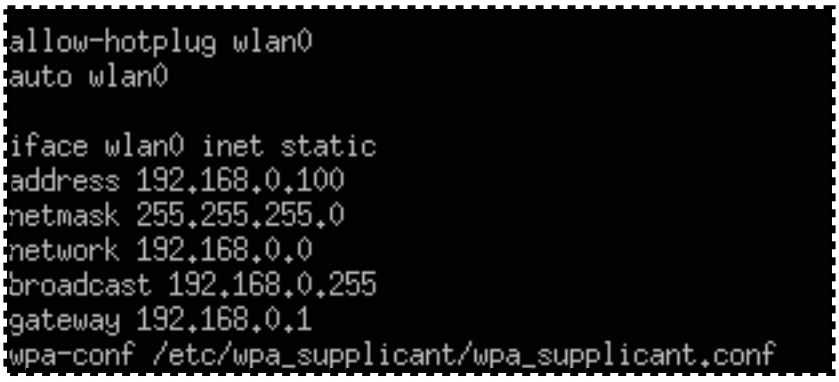

Figura 3.6: Configurações da interface *wlan0*.

Finalizadas todas essas configurações é necessário que as duas interfaces sejam reiniciadas. A maneira mais simples e eficaz de fazer tais alterações é reiniciando a *Raspberry Pi*.

Apesar das duas interfaces terem sido configuradas, apenas uma delas foi utilizada para realizar a comunicação via *sockets* no projeto, a *eth0*. Essa decisão foi realizada devido ao baixo desempenho observado na comunicação Wi-Fi, problema este causado devido a baixa corrente fornecida aos módulos USB.

3.2.3 Requisições *Sockets*

Para que o servidor permanecesse em funcionamento o tempo todo, recebendo requisições dos usuários, foi produzido um código em *Java*, utilizando *API Sockets*. No entanto, foi necessário preparar o ambiente de execução inicialmente, sendo necessária a instalação de uma Máquina Virtual Java (JVM). Foi utilizado o repositório de pacotes *apt-get* para a realização deste procedimento, obtendo assim o pacotes *openjdk-7-jre* para a execução das aplicações e o *openjdk-7-jdk* para a compilação dos códigos.

Com a plataforma *Raspberry Pi* preparada para a execução dos códigos, foi então elaborado um programa com uma série de rotinas para a requisição de comandos e o seu futuro processamento. Nele é definido o procedimento de criação de um *socket* para o recebimento de conexões. Posteriormente ele aceita as comunicações solicitadas pelo cliente, enviando uma mensagem para o mesmo informando que a conexão foi aceita. Por fim o servidor recebe os comandos enviados pelo cliente, processa as requisições, através da função *"recev.command(message)"*, e finaliza a comunicação. O programa deverá manter-se nesse ciclo aguardando conexões e comandos por tempo indeterminado.

Há uma série de comandos interpretados e processados pelo servidor. Parte deles estão relacionados ao controle da televisão, os demais alternam o estado de funcionamento dos dispositivos elétricos controlados por relés. Esses comandos são listados respectivamente nas Tabelas 3.1 e 3.2.

Tabela 3.1: Comandos aceitos pelo servidor para o controle televisivo.

MENSAGEM	NOME	DESCRIÇÃO
pw	***Power***	**Ligar/Desligar TV**
up	***Up***	***(Channel +)* Incrementa o canal da TV**
dw	***Down***	***(Channel -)* Decrementa o canal da TV**
rg	***Right***	***(Volume +)* Incrementa o volume da TV**
lf	***Left***	***(Volume -)* Decrementa o volume da TV**
b0	***Button0***	**Botão zero da TV**
b1	***Button1***	**Botão um da TV**

MENSAGEM	NOME	DESCRIÇÃO
b2	*Button2*	Botão dois da TV
b3	*Button3*	Botão três da TV
b4	*Button4*	Botão quatro da TV
b5	*Button5*	Botão cinco da TV
b6	*Button6*	Botão seis da TV
b7	*Button7*	Botão sete da TV
b8	*Button8*	Botão oito da TV
b9	*Button9*	Botão nove da TV
mt	*Mute*	Ativa/Desativa o som da TV
al	*Alternate*	Alterna entre dois canais da TV

Tabela 3.2: Comandos aceitos pelo servidor para controle de dispositivos.

MENSAGEM	NOME	DESCRIÇÃO
n1	*On1*	Liga Dispositivo 1
f1	*Off1*	Desliga Dispositivo 1
n2	*On2*	Liga Dispositivo 2
f2	*Off2*	Desliga Dispositivo 2
n3	*On3*	Liga Dispositivo 3
f3	*Off3*	Desliga Dispositivo 3
d1	*Device1*	Liga/Desliga Dispositivo 1
d2	*Device2*	Liga/Desliga Dispositivo 2
d3	*Device3*	Liga/Desliga Dispositivo 3

A presença de comandos semelhantes para o controle de dispositivos eletrônicos acionados por relés ocorre devido a diferentes implementações realizadas no lado do cliente, em resumo, as mensagens *d1*, *d2* e *d3* fazem com que o relé alterne seu chaveamento, independente da posição a qual se encontra, diferente dos outros comandos, os quais definem o estado que o relé permanecerá.

3.2.4 *Web-Service*

Para configurar um *web-service* na *Raspberry Pi* foi escolhido o servidor *Apache Tomcat 7*, em razão deste ser um software livre e por suportar aplicações *Java* para manipular uma página *web*.

Este programa também faz uso de uma JVM, previamente descrita e instalada para responder as requisições enviadas por *sockets*. Com isso, tornou-se necessário apenas a instalação do seu pacote, *tomcat7*, através do diretório *apt-get*. Para executar ou interromper os serviços do servidor basta utilizar os comandos *"sudo service tomcat7 start"* e *"sudo service tomcat7 stop"* respectivamente. Enquanto a aplicação estiver executando, todos os códigos armazenados dentro do diretório *"Tomcat/webapps/"* serão disponibilizados para o acesso *HTTP*.

Outro ponto importante de se destacar é a ferramenta *Vaadin*, utilizada para a geração dos códigos com conteúdo *web*, sendo ela um *framework* de código aberto disponível na *Marketplace* do *Eclipse IDE*. Utilizando-o é possível fazer toda a codificação de uma página *web* utilizando apenas *Java*, por exemplo, realizar a formatação do layout da página e a adição de botões com funcionalidades, como pode ser observado na Figura 3.7. Outro componente criado ao fazer uso deste *plugin* é um arquivo CSS (*Cascading Style Sheets*), responsável por definir os estilos utilizados na apresentação do conteúdo.

Todos os botões implementados no servidor *HTTP* realizam a comunicação por *sockets* com um programa centralizador da *Raspberry Pi*, responsável por controlar as ações de GPIO.

```
final GridLayout layout = new GridLayout(7, 13);

Button button1 = new Button("PWR");
button1.addListener(this);
layout.addComponent(button1, 0, 0);
```

Figura 3.7: Utilização do *plugin Vaadin*.

Finalizado todo o código, o *Eclipse IDE* utilizado para o desenvolvimento permite exportar o programa para uma extensão WAR (*Web Application Archive*). Adicionando este arquivo ao diretório *"Tomcat/webapps/"* o programa *Tomcat7* fará automaticamente a sua extração, disponibilizando ele para o acesso *HTTP* imediatamente.

As funcionalidades presentes no código desenvolvido para a página *web*, assim como o layout produzido para o acesso *HTTP* serão melhor descritos em tópicos futuros, durante as apresentações dos controladores utilizados pelos usuários.

3.2.5 *LIRC*

O *LIRC* (*Linux Infrared Remote Control*) é um pacote, também disponível no repositório *apt-get*, que permite decodificar e enviar sinais infra-vermelho dos controles remotos mais comuns. [27] Para isso, utilizando a *Raspberry Pi*, é necessário apenas adicionar um receptor (IRM3638) e um emissor (PHIV590) de infravermelhos utilizando os pinos de GPIO disponíveis. No desenvolvimento deste projeto foram escolhidos o pino GPIO17 para ler os sinais do receptor de infravermelho, e o pino GPIO22 para a transmissão de dados. No entanto, visando uma potência maior na emissão de dados, foi utilizado um transistor 2N3904 para que pudesse ser utilizada uma tensão de 5V, ao invés da tensão de 3V fornecida pelos pinos de GPIO. O circuito final implementado para a utilização dos dois sensores é apresentado na Figura 3.8.

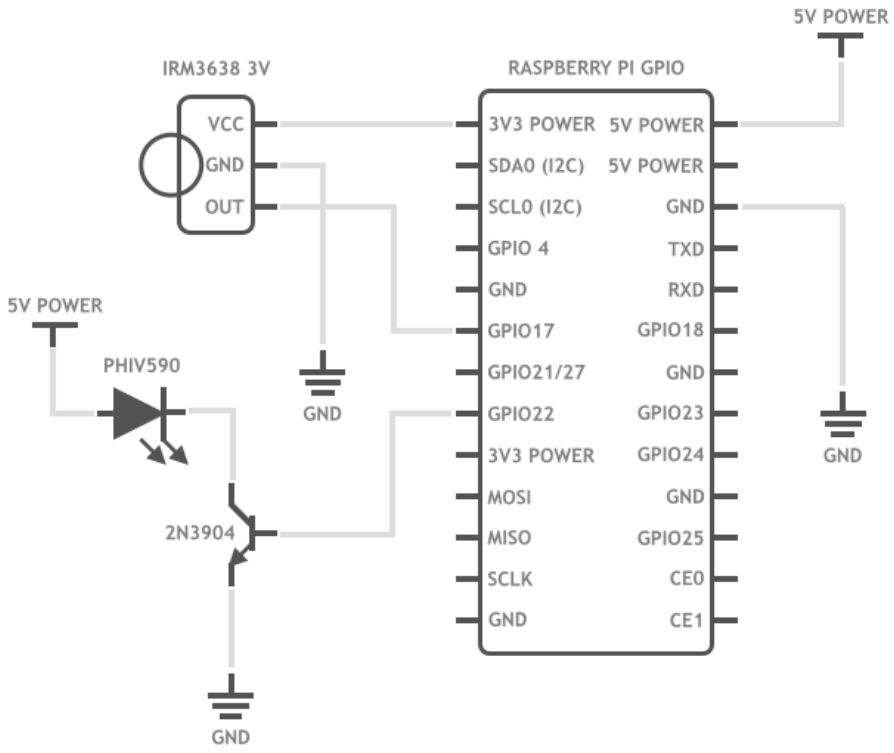

Figura 3.8: Esquemático do circuito emissor e receptor IV utilizando a Raspberry Pi.

Finalizado o circuito físico, é necessária a configuração do pacote *LIRC* na plataforma de desenvolvimento. Para isso, deve-se inicialmente instalar o pacote utilizando o comando "*sudo apt-get install lirc"*, seguida de alterações em dois arquivos. O primeiro deles é o arquivo "*/etc/modules*", onde duas linhas devem ser adicionadas para a definição dos pinos de entrada e saída, sendo elas:

```
lirc_dev
lirc_rpi  gpio_in_pin=17  gpio_out_pin=22
```

Já o segundo é um arquivo nomeado "*hardware.conf*", localizado dentro do diretório "*/etc/lirc/*", nele devem ser realizadas as maiores mudanças, conforme apresentadas na Figura 3.9.

```
# /etc/lirc/hardware.conf
#
# Arguments which will be used when launching lircd
LIRCD_ARGS="--uinput"

#Don't start lircmd even if there seems to be a good config file
#START_LIRCMD=false

#Don't start irexec, even if a good config file seems to exist.
#START_IREXEC=false

#Try to load appropriate kernel modules
LOAD_MODULES=true

# Run "lircd --driver=help" for a list of supported drivers.
DRIVER="default"
# usually /dev/lirc0 is the correct setting for systems using udev
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"

# Default configuration files for your hardware if any
LIRCD_CONF=""
LIRCMD_CONF=""
```

Figura 3.9: Configurações do arquivo hardware.conf.

Finalizadas essas etapas o *LIRC* deve ser reiniciado para que as mudanças sejam aplicadas, para isso basta utilizar os comandos "*sudo /etc/init.d/lirc stop*" e "*sudo /etc/init.d/lirc start*" em sequência.

Agora, para utilizar o emissor infravermelho com sucesso devemos inicialmente obter as mensagens que queremos transmitir, para isso devemos utilizar o comando "*irrecord -d /dev/lirc0 / etc/lirc/lircd.conf*" com o LIRC fora de execução, e seguir as etapas requisitadas pelo programa, as quais devem solicitar o nome do comando que deseja-se gravar e a respectiva tecla do controle remoto. Esta deve ser pressionada para que o receptor infravermelho obtenha as características do comando, relacionadas essas à espaços e marcas. Após realizar todas as etapas do programa e finalizá-lo, um novo arquivo será gerado, "*/etc/lirc/lircd.conf*", contendo as informações fornecidas anteriormente, para que sejam utilizadas agora pelo emissor.

Gravados todos os comandos no novo arquivo, pode-se utilizar então o emissor infravermelho conectado à *Raspberry Pi* para realizar a comunicação com aparelhos televisivos, da mesma forma que um controle remoto tradicional. Para enviar os comandos deve-se iniciar novamente o LIRC e acionar o comando "*irsend SEND_ONCE /etc/lirc/lircd.conf KEY*", onde *KEY* é o comando definido durante a criação do arquivo *lircd.conf* . Para a implementação deste projeto foram implementadas uma quantidade limitada de funcionalidades, conforme pode-se observar na Figura 3.10.

```
begin codes
    KEY_POWER               0x100C
    KEY_VOLUMEUP            0x1010
    KEY_VOLUMEDOWN          0x1011
    KEY_CHANNELUP           0x1020
    KEY_CHANNELDOWN         0x1021
    KEY_MUTE                0x100D
    KEY_1                   0x1001
    KEY_2                   0x1002
    KEY_3                   0x1003
    KEY_4                   0x1004
    KEY_5                   0x1005
    KEY_6                   0x1006
    KEY_7                   0x1007
    KEY_8                   0x1008
    KEY_9                   0x1009
    KEY_0                   0x1000
    KEY_C                   0x103A
end codes
```

Figura 3.10: Comandos definidos no arquivo lircd.conf.

3.2.6 Ativação dos Relés

O controle do ventilador e das luminárias definidos na arquitetura do projeto ocorreu através da utilização de relés, para que a *Raspberry Pi* conseguisse controlar circuitos externos com grandes correntes. Inicialmente foram usadas relés individuais com o diodo de proteção para o circuito de acionamento. No entanto, devido a necessidade de várias relés, foi utilizado um módulo contendo oito relés, Figura 3.11, as quais já possuem o circuito de proteção, e podem vir a permitir que o circuito possa ser expandido futuramente para o controle de mais dispositivos eletrônicos. Além disso, o módulo apresenta LEDS de acionamento, facilitando a realização de testes.

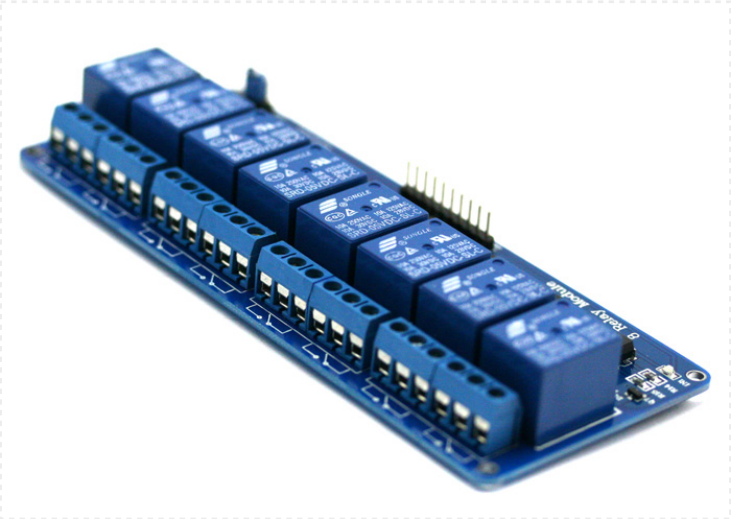

Figura 3.11: Módulo de relés utilizado.

Para o funcionamento correto dos relés presentes no módulo é necessária uma tensão de alimentação de 5V. No entanto, para o chaveamento é possível utilizar uma tensão de referência de 3.3V fornecida pelos pinos de GPIO. Com isso, o circuito físico projetado ficou conforme apresentado na Figura 3.12.

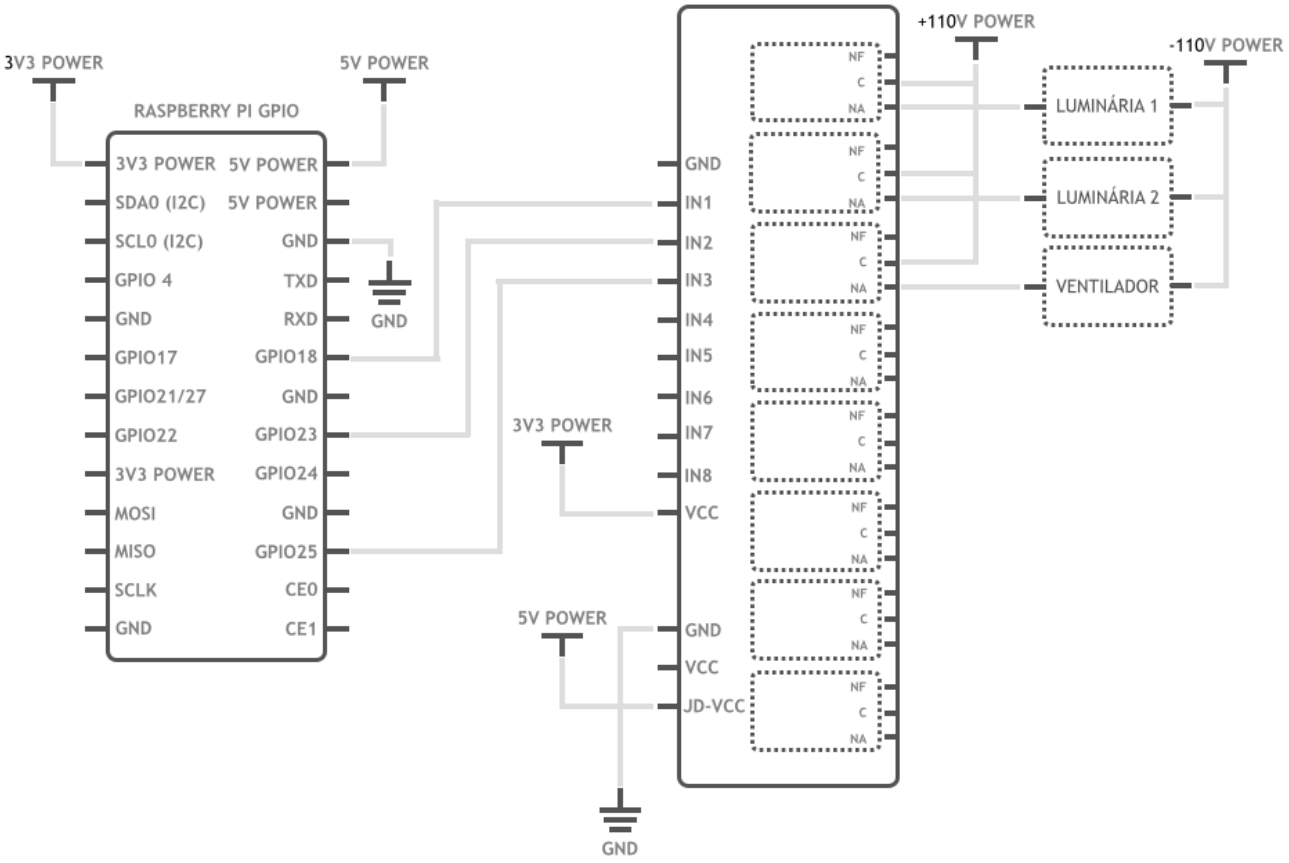

Figura 3.12: Circuito físico para utilização dos relés.

Controlar os pinos de GPIO pode ser uma tarefa simplificada se utilizadas as bibliotecas especializadas para tal função. Durante o desenvolvimento do projeto fez-se uso da biblioteca *Pi4J*. Para instalar a biblioteca é necessário apenas dois comandos no terminal da *Raspberry Pi*, o primeiro deles é o "*wget http://pi4j.googlecode.com/files/pi4j-0.0.5.deb*", onde será realizado o download do seu arquivo instalador, o segundo é o "*sudo dpkg -i pi4j-0.0.5.deb*", responsável por realizar o processo de instalação propriamente [28]. Para compilar programas que fazem uso da biblioteca é necessário inserir um novo parâmetro, o seu endereço, por exemplo o comando "*javac -classpath .:classes:/opt/pi4j/lib/'*' *.java*". Esse endereço precisa ser inserido também durante a execução dos programa, o qual deve ser executado utilizando o modo privilegiado.

Com a construção do circuito e a instalação da biblioteca, a atividade de processar requisições relacionadas aos dispositivos eletrônicos tornou-se fácil, tendo em vista que para acioná-los ou desligá-los são necessárias poucas linhas de código, como pode ser observado na Figura 3.12 a troca de estado.

```
// create gpio controller
final GpioController gpio = GpioFactory.getInstance();
final GpioPinDigitalOutput pin;


        if (cmd.equals("d1")){
                // provision gpio pin #01 as an output pin and turn on
                pin = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, "MyLED", PinState.HIGH);
                pin.toggle();
```

Figura 3.13: Código utilizando a biblioteca Pi4J para acionamento do relé.

3.3 Controladores

Para o desenvolvimento dos controles, foram utilizados alguns dos dispositivos eletrônicos mais utilizados no cotidiano das pessoas, os *smartphones.* Além dos celulares, foram utilizados também o *Kinect* devido a sua simplicidade e os controladores *HTTP*, sendo este um meio difundido em diferentes plataformas de acesso e com possível expansão para o oferecimento de outros serviços.

Todos os controles foram implementados visando ter uma interface de simples utilização, intuitiva. Para realizar a comunicação com o servidor central de comandos, a *Raspberry Pi*, todos utilizam a comunicação via *sockets*, com exceção do servidor *HTTP*.

A seguir serão descritos, de forma mais sucinta, como foram desenvolvidos individualmente cada interface controladora.

3.3.1 Aplicativo *Android*

Para implementar o aplicativo *Android*, nomeado então de *UNIvController*, foi utilizado a todo momento a linguagem de programação *Java* e alguns trechos de XML para a criação da interface gráfica. O ambiente de desenvolvimento escolhido foi o *Android Developer Tools*, programa de código aberto, onde foram definidas todas as classes usadas, contendo as suas respectivas rotinas, conforme apresentadas na Figura 3.14.

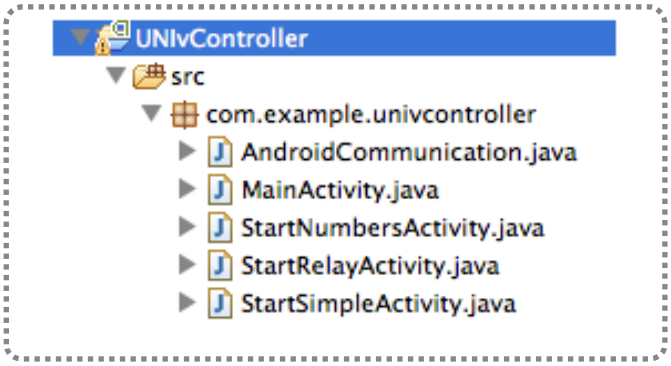

Figura 3.14: Classes utilizadas para o desenvolvimento do projeto *Android*.

Inicialmente, no projeto foi definido que cada classe, com excessão da *AndroidCommunication* (responsável pela comunicação com o servidor), possuiria uma interface gráfica, ou seja, cada classe apresentaria uma *"activity"* própria, arquivo XML, responsável pela disposição e representação gráfica dos campos, botões, listas e textos. Um exemplo deste arquivo pode ser visto na Figura 3.15.

Outro fator importante na criação dos arquivos *"activity"* e as suas classes são as suas respectivas declarações no arquivo *AndroidManifest.xml* conforme pode ser visualizada na Figura 3.16 onde é feita a declaração da classe *MainActivity*. Além disso, o arquivo *AndroidManifest.xml* é responsável por informar a versão mínima da *API Android* e pelas permissões fornecidas ao usuário, por exemplo, a permissão de que o usuário tenha acesso a internet, fundamental neste projeto.

```
<LinearLayout
    android:id="@+id/linearLayout1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_gravity="center"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginRight="40dp"
        android:layout_marginLeft="40dp"
        android:src="@drawable/projicon" />

    <TextView
        android:id="@+id/textTitulo1"
```

Figura 3.15: Parte do arquivo *activity_main.xml* e sua respectiva representação gráfica.

Como visto na Figura 3.15, na representação da interface gráfica inicial, o aplicativo possui três tipos diferentes de controle, o primeiro deles, o *Simple Control*, apresenta comandos básicos para o controle televisivo, relacionados a alteração de canais, volume e funcionalidade liga/desliga. Já o segundo, *Numbers Control*, possui as mesmas funcionalidades do anterior, no entanto, com algumas funções extras, como o teclado numérico e botão *mute*. Por fim, o *Relay Control* é utilizado apenas para funções relacionadas ao acionamento de dispositivos eletrônicos, controlados por relés no servidor embarcado *Raspberry Pi*. Os três controles são apresentados na Figura 3.16.

Figura 3.16: Interface dos controles *Simple*, *Numbers* e *Relay* respectivamente.

Todas as ações de comunicação do aplicativo cliente com o servidor, ativas ao pressionarem um botão do controle, são realizadas chamando funções da classe *AndroidCommunication*. Foi definido também nela que toda a comunicação seria iniciada e finalizada após o recebimento e envio das mensagens, ou seja, a todo momento que o usuário pressionar um botão, a comunicação *socket* será iniciada para o envio da mensagem e finalizada logo após, evitando assim o congestionamento da rede quando o controle permanece inativo durante muito tempo.

Para a realização de testes com o aplicativo foi utilizado a todo momento um celular *Samsung Galaxy S3 Mini* conectado, via USB, ao computador de desenvolvimento, tendo em vista que o ambiente *Android Developer Tools* fornece suporte a este tipo de atividade.

3.3.2 Aplicativo *iOS*

A codificação do aplicativo para o *iPhone*, *iPod* e *iPad* foi realizado utilizando o ambiente de desenvolvimento exigido pela *Apple*, o *Xcode*. Nas configurações iniciais do projeto decidiu-se utilizar a funcionalidade *Storyboard*, responsável pela facilidade na visualização das relações entre as interfaces produzidas, como pode ser observado na Figura 3.18 pela sequência das setas indicativas.

Neste projeto, semelhante ao desenvolvimento do aplicativo *Android*, foi utilizada uma classe para cada interface gráfica dos controles implementados, conforme mostra a Figura 3.17.

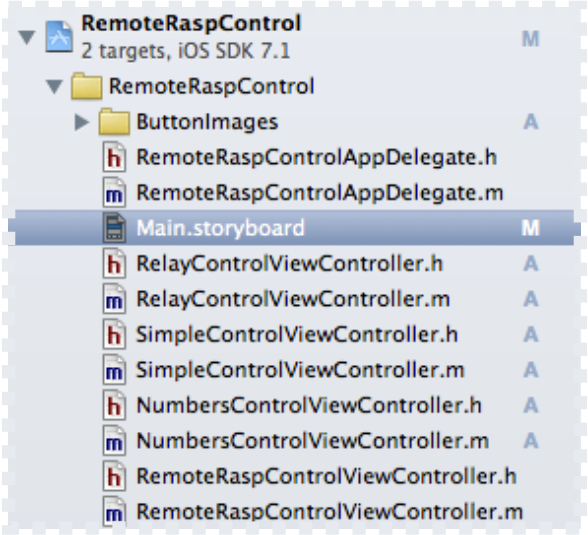

Figura 3.17: Classes utilizadas para o desenvolvimento do projeto *iOS*.

A relação entre as classes criadas são observadas na Figura 3.18, onde são apresentadas as interfaces de cada classe, assim como são definidas as funcionalidades dos três botões do menu principal. O processo de criação dos botões do menu principal se resume ao processo de *drag-and-drop*, arrastar e largar, suportado pelo ambiente conforme apresentado na Figura 3.19. Com os botões inseridos no *layout* principal, foi necessário então segurar a tecla *Ctrl*, clicar em algum deles, e arrasta-lo em direção a nova tela a ser apresentada ao pressionar o objeto.

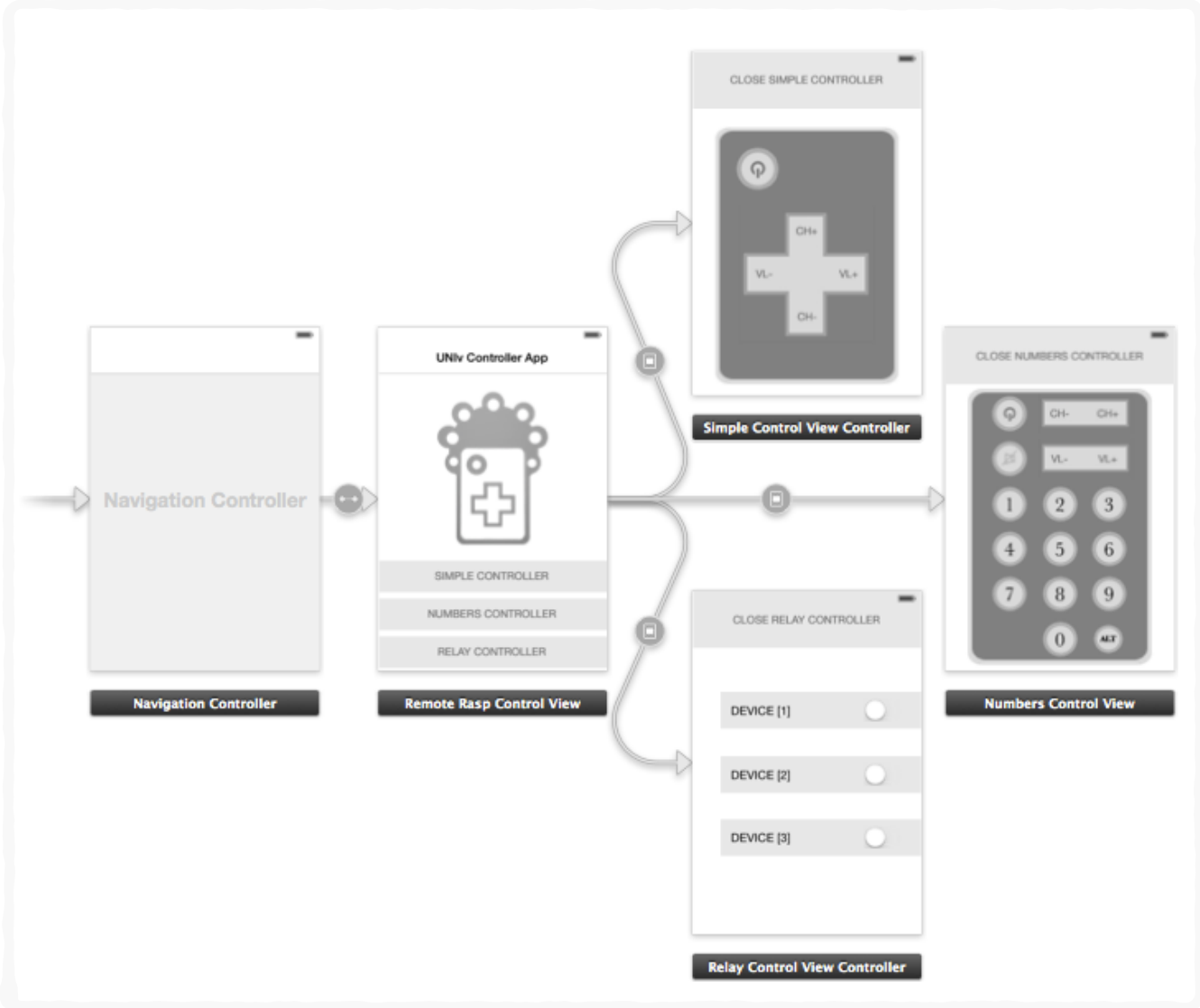

Figura 3.18: *Storyboard* contendo a relação entre as classes.

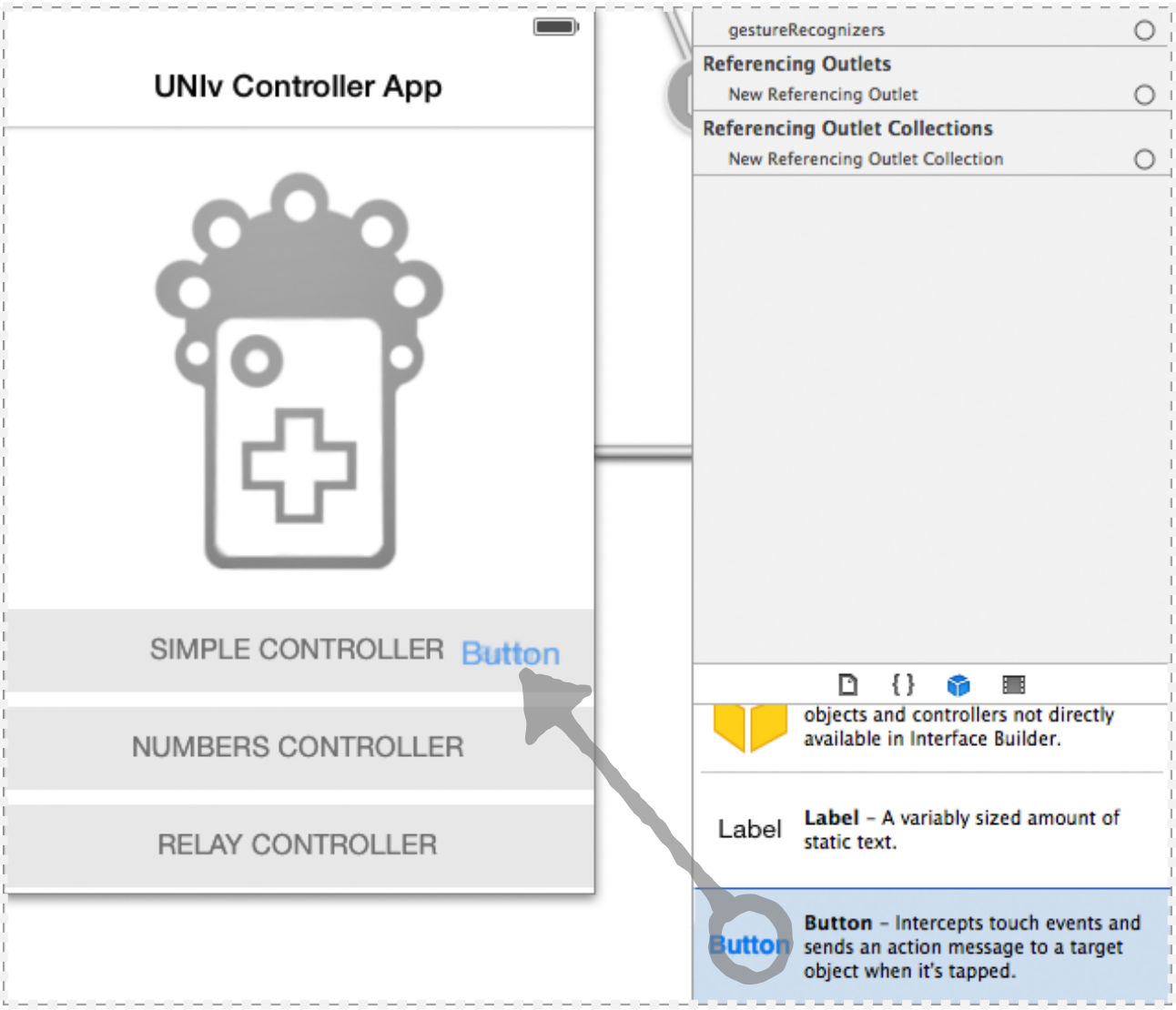

Figura 3.19: Procedimento *drag-and-drop* suportado pelo xCode.

Em relação aos botões presentes nas interfaces dos controles *Simple*, *Numbers* e *Relay,* eles foram criados pelo mesmo processo. No entanto, na edição das suas funcionalidades é necessário implementá-las em código, tendo em vista que o processo de funcionamento deles é igual ao implementado no aplicativo *Android*, onde ao pressionar qualquer um deles será inicializada uma comunicação *socket* com o servidor *linux* para o envio do comando, seguido do encerramento da comunicação. Parte do código utilizado para o envio dessas mensagens, na comunicação *socket*, é observada na Figura 3.20.

```
- (void)sendNetworkCommunication:(NSString*)command {
    CFReadStreamRef readStream;
    CFWriteStreamRef writeStream;
    CFStreamCreatePairWithSocketToHost(NULL, (CFStringRef)@"192.168.0.200", 1500, &readStream, &
        writeStream);
    inputStream = (__bridge NSInputStream *)readStream;
    outputStream = (__bridge NSOutputStream *)writeStream;
    [inputStream setDelegate:self];
    [outputStream setDelegate:self];
    [inputStream open];
    [outputStream open];

    NSString *response  = [NSString stringWithFormat:@"%@", command];
    NSData *data = [[NSData alloc] initWithData:[response dataUsingEncoding:NSASCIIStringEncoding]];
    [outputStream write:[data bytes] maxLength:[data length]];

    [inputStream close];
    [outputStream close];
}
```

Figura 3.20: Código em *Objective-C* utilizado para a comunicação *socket*.

Para a realização dos testes com o aplicativo desenvolvido para o *iOS* não foi possível utilizar um dispositivo físico, devido a necessidade de uma licença paga de um ano da *Apple*. Com isso, todos as análises feitas em cima do aplicativo foram baseadas no uso do simulador disponibilizado pelo ambiente *xCode*, o qual apresenta bom desempenho e interface gráfica formidável, conforme observado na Figura 3.21.

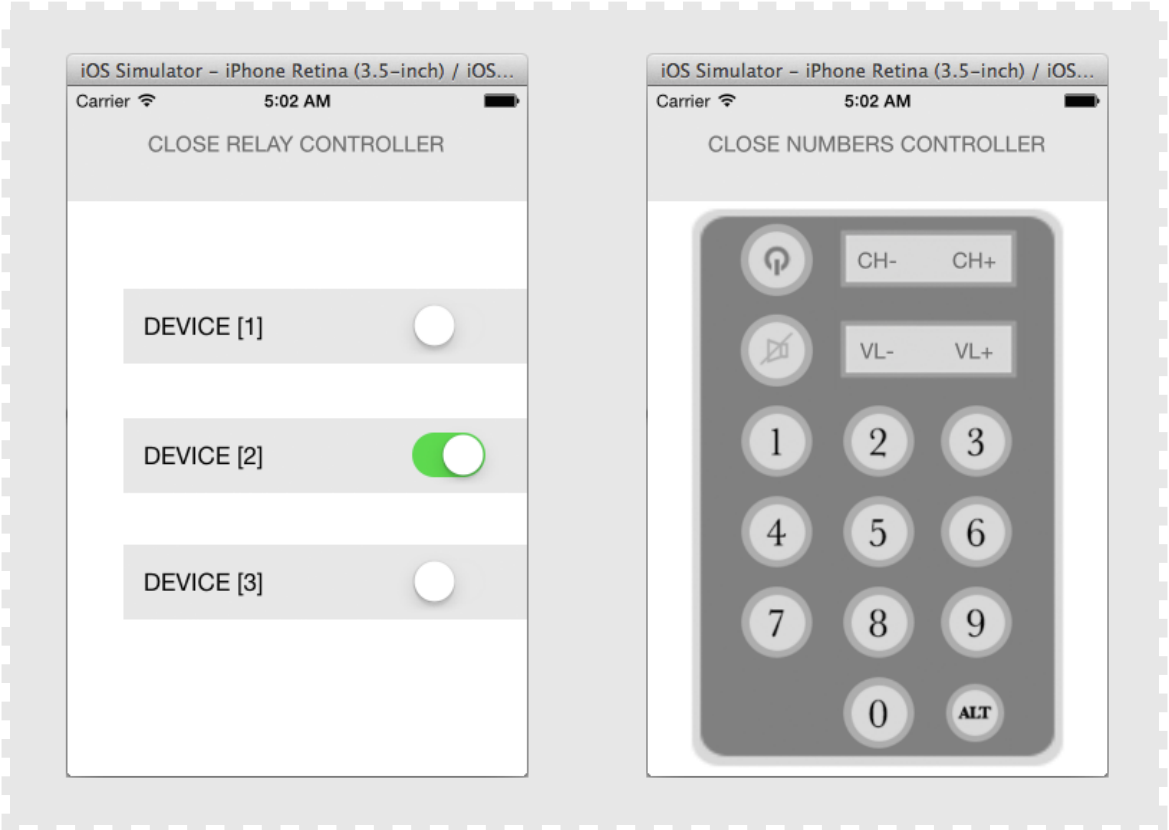

Figura 3.21: Interface gráfica do simulador disponibilizado pelo xCode.

3.3.3 Acesso *HTTP*

A utilização de um servidor de acesso via *HTTP* ocorreu devido a sua praticidade, considerando que atualmente a maioria dos dispositivos pessoais, como celulares, *tablets* e *notebooks*, podem utilizar tal comunicação. Com isso em mente foi desenvolvido uma aplicação com layout simples e de fácil uso, conforme apresentado na Figura 3.22. Nela é possível notar também alguns detalhes da sua implementação, onde foi utilizado o componente principal de *layout*, na vertical, contendo um outro de matriz, utilizado para os botões de controle televisivo e dos dispositivos.

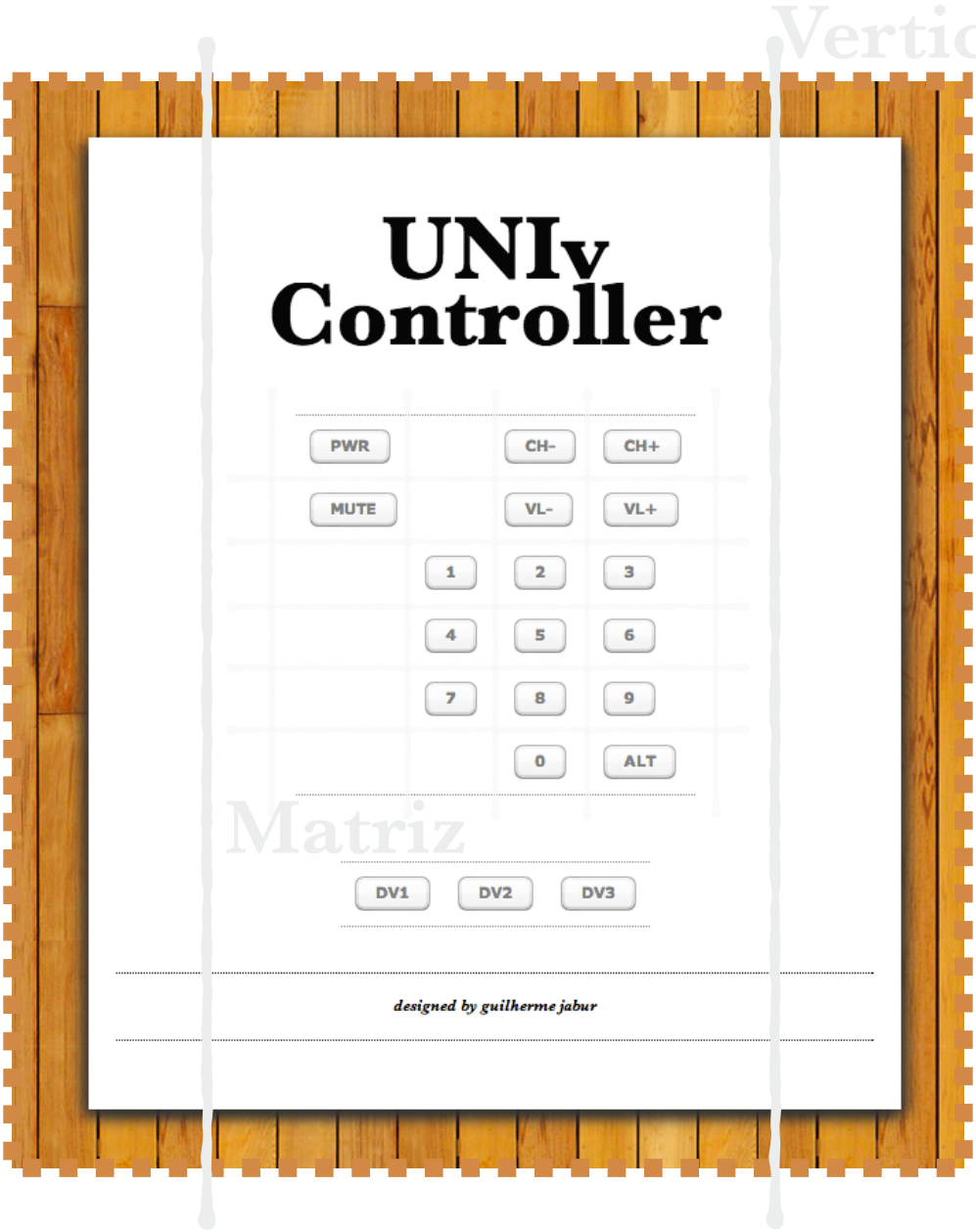

Figura 3.22: *Layout* da tela de acesso via *HTTP*.

Os botões superiores são responsáveis pelo controle televisivo, já as três teclas inferiores controlam o estado dos dispositivos eletrônicos ativados por relés.

Para obter o acesso via *HTTP* basta utilizar um navegador disponível no mercado e acessar o endereço *web* com o IP do servidor, a porta de acesso do serviço (foi utilizada a padrão), e o nome do arquivo desenvolvido, nesse caso *RemoteControlServerHTTP* conforme apresentado na Figura 3.23.

Figura 3.23: Endereço de acesso do serviço *HTTP*.

3.3.4 Controle *Kinect*

Para a implementação do controle por *Kinect* foi necessária a instalação inicial de uma série de pacotes no *notebook* contendo o sistema operacional *Mac OS X*, utilizado para desenvolvimento do projeto, visando a utilização dos sensores presentes no dispositivo *Microsoft* ao conectá-lo via USB no computador responsável pelo processamento.

Utilizando o *software MacPorts* [26] no terminal do *Macbook* foram instaladas as primeiras dependências, as bibliotecas Libtool e Libusb, através dos comandos:

- sudo port install libtool
- sudo port install libusb +universal

Com as bibliotecas ativas, foi então obtido o *kit* de desenvolvimento *OpenNI 1.5.7*, disponibilizado antigamente no *website* do projeto [15], para a sua posterior instalação através do comando "*sudo ./install.sh*" utilizado no terminal, dentro do diretório criado pelo arquivo previamente adquirido e extraído. Por fim, foi então realizado o *download* do módulo chamado *SensorKinect*, adquirido na página do projeto *GitHub* e utilizado para ajudar na interação entre o

Kinect e o *framework* do *OpenNI*. A instalação deste módulo ocorre de forma idêntica a do primeiro.

Finalizada todas essas etapas já estavam prontos os primeiros exemplos para serem executados e verificar o seu funcionamento. Por exemplo a utilização do rastreador de mãos (*NiHandTracker*), código este que foi modificado e renomeado para o desenvolvimento do projeto, contendo uma série de arquivos com extensão *.cpp* e *.h*, Figura 3.24, responsáveis por rastrear as mãos dos usuários e identificar alguns movimentos como o de *click* e *wave*, descritos futuramente.

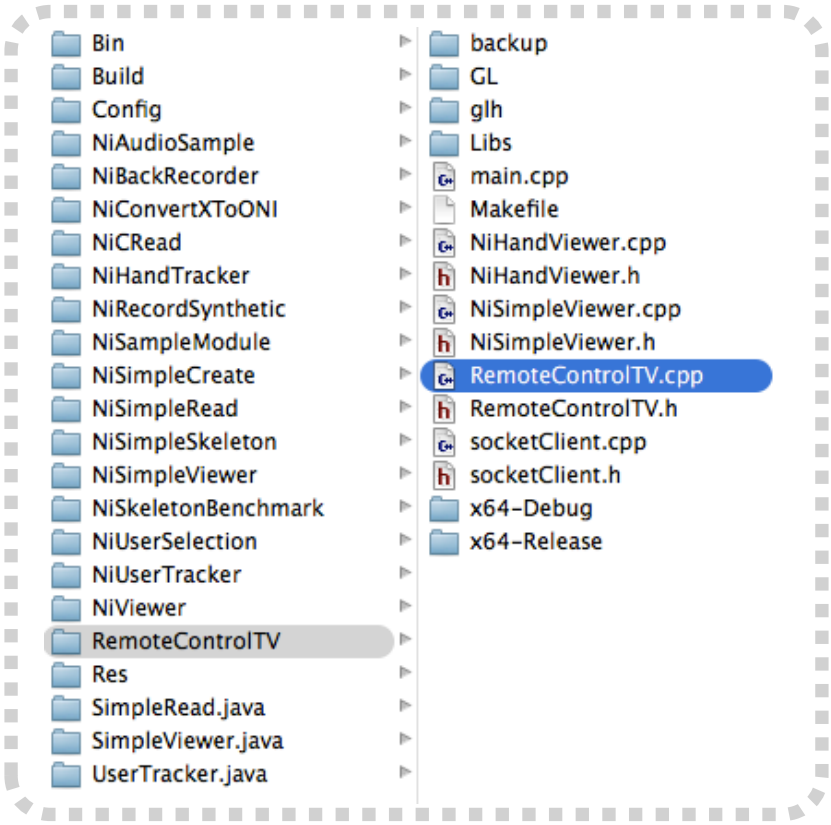

Figura 3.24: Diretório do projeto utilizado, *RemoteControlTV*.

Dentro do diretório apresentado anteriormente na Figura 3.24 é possível notar a presença de algumas outras pastas contendo algumas bibliotecas de dependências e um arquivo chamado *Makefile*, utilizado para simplificar as etapas de compilação.

Para compilar e testar os arquivos desenvolvidos no projeto são necessários os comandos "*sudo make all*", dentro do diretório "*/RemoteControlTV*" contendo os códigos, e "*./KinectRemoteControl*" na pasta "*/Bin/x64-Release*", tendo em vista que o arquivo *Makefile* modificado cria o arquivo executável em outra pasta.

Todas as modificações no código visaram a criação de uma interface simples de utilização do *Kinect*, para isso foram definidos quatro novos movimentos além dos dois já existentes no código original, todos eles são utilizados apenas para o controle televisivo e são melhores descritos adiante:

1. *Wave:* Produzido ao acenar com as mãos de um lado para o outro repetidamente, Figura 3.25. Este é um dos principais comandos, utilizado para "adquirir" ou "abandonar" o controle. Toda vez que o movimento é realizado pela primeira vez por um usuário ele irá obter o domínio do controle da TV, podendo então realizar os outros comandos. No entanto, caso o mesmo usuário utilize o *wave* já estando com o domínio do controle, ou qualquer outra pessoa faça o movimento, então o poder de controlar a televisão será perdido, ou ganho pelo outro usuário.

Figura 3.25: Movimento *wave*.

2. *Click:* Movimento executado ao aproximar e distanciar a mão abruptamente do sensor *Kinect*, Figura 3.26. Ao utiliza-lo, a televisão irá alterar o seu estado de ligada para desligada, ou vice-versa.

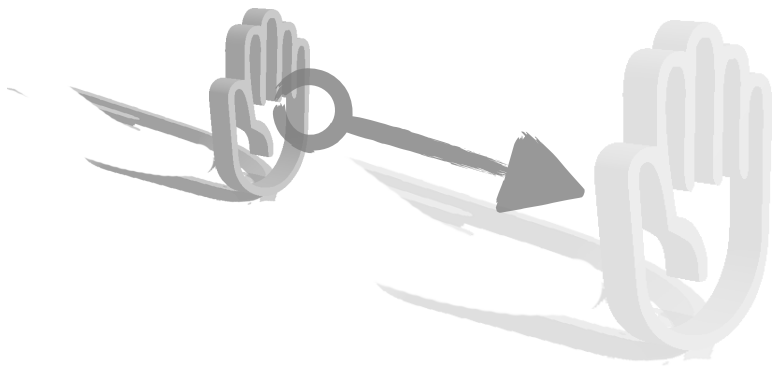

Figura 3.26: Movimento *click*.

3. *Swipe Up:* Comando responsável por aumentar o volume da televisão. Trata-se do gesto de deslizar a mão para cima rapidamente, conforme representado na Figura 3.27.

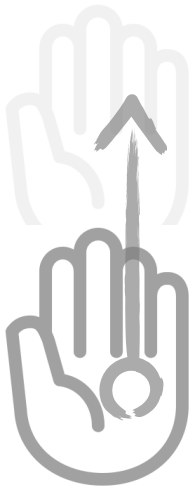

Figura 3.27: Movimento *swipe up*.

4. *Swipe Down:* Movimento oposto ao *Swipe Up*. Deve-se deslizar a mão controladora para baixo. Utilizado para diminuir o volume da TV.

5. *Swipe Right:* Formado ao deslizar a mão para a direita do usuário, conforme apresentado na Figura 3.28. O comando enviado ao servidor ao gesticula-lo será o de *Channel+*, próximo canal.

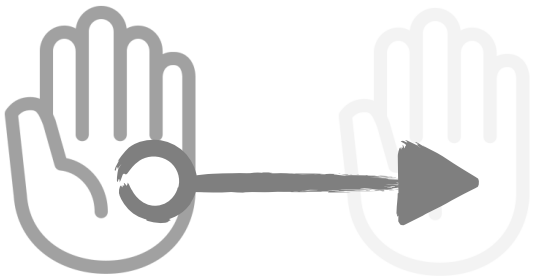

Figura 3.28: Movimento *swipe right*.

6. *Swipe Left:* Utilizado para retornar ao canal anterior, *Channel-*, movimento produzido ao deslizar para a esquerda a mão, oposto do comando *Swipe Right*.

Os movimentos de *swipe* não eram disponibilizados na versão 1.5.7 do *OpenNI* utilizada durante o projeto, com isso foi necessário desenvolve-los utilizando um vetor que armazenava as coordenadas das mãos do usuário e verificava se o avanço dela em um determinado tempo era o suficiente para ser considerado um gesto de deslize. Parte do código é apresentado na Figura 3.29 adiante, onde cada valor retornado indica o sentido do movimento.

```
int HandTracker::DetectSwipe (const XnPoint3D*  pPosition){
    int i=0;

    float x0 = ArrayPositionX[0];
    float xn = ArrayPositionX[SIZE_ARRAY_SWIPE-1];
    float y0 = ArrayPositionY[0];
    float yn = ArrayPositionY[SIZE_ARRAY_SWIPE-1];

    if (fabs(xn-x0) > MIN_X_DELTA && fabs(yn-y0) < MAX_Y_DELTA){
        if (x0 > xn) return 1;
        else return 2;
    }

    if (fabs(xn-x0) < MAX_X_DELTA && fabs(yn-y0) > MIN_Y_DELTA){
        if (y0 > yn) return 3;
        else return 4;
    }

    return 0;
}
```

Figura 3.29: Função responsável por detectar gestos de *swipe*.

Demais alterações implementadas foram todas relacionadas ao comportamento de "ganho e perda" do controle e ao envio das mensagens *sockets*, realizadas a todo momento que um dos gestos é produzido, com exceção do *wave*.

3.4 Considerações Finais

Com o desenvolvimento das atividades descritas nesse capítulo foi possível perceber que os dispositivos de controle implementados para os usuários trabalham independentes entre si, não sendo necessária a implementação de todos para o funcionamento correto do sistema como um todo. No entanto, a união de todos os componentes fornecem um ambiente controlado por multiplataformas, o que minimiza a problemática de um usuário comum não possuir algum

dispositivo de acesso para a troca de mensagens com o servidor embarcado. Em relação à *Raspberry Pi* nota-se que o desenvolvimento dos métodos de controle foram facilitados devido a presença de algumas bibliotecas auxiliares. Com isso, finalizado a união de todos os componentes descritos neste capítulo foi possível obter o ambiente final para a realização dos testes de desempenho do sistema desenvolvido.

Capítulo 4: Resultados

4.1 Resultados Obtidos

Todas as aplicações desenvolvidas para os *smartphones*, servidor *HTTP* e *Kinect* apresentaram excelentes resultados, sendo necessário um intervalo inferior à um segundo entre o recebimento das requisições e o seu processamento, controlando corretamente os dispositivos os quais eles se comprometiam a gerenciar, como a televisão e os aparelhos acionados por relés (luminárias e ventiladores). No entanto foram encontrados atrasos consideráveis durante o carregamento inicial da página *web*, sendo estes descritos futuramente.

Por se tratar de um sistema embarcado, procurou-se avaliar se o mesmo seria capaz de suportar uma demanda de processamento ao qual o estaríamos impondo em casos extremos. Por isso, testamos o sistema funcionando enquanto mantém-se inativo sem receber requisições, Figura 4.1, e para diversos usuários solicitando serviços concomitantemente. Assim verificamos o consumo de memória e de processamento do servidor *Raspberry Pi*. A Figura 4.2 apresenta o desempenho do sistema enquanto recebia requisições de todos os dispositivos. Entretanto, com o *website* do acesso *HTTP* pré-carregado.

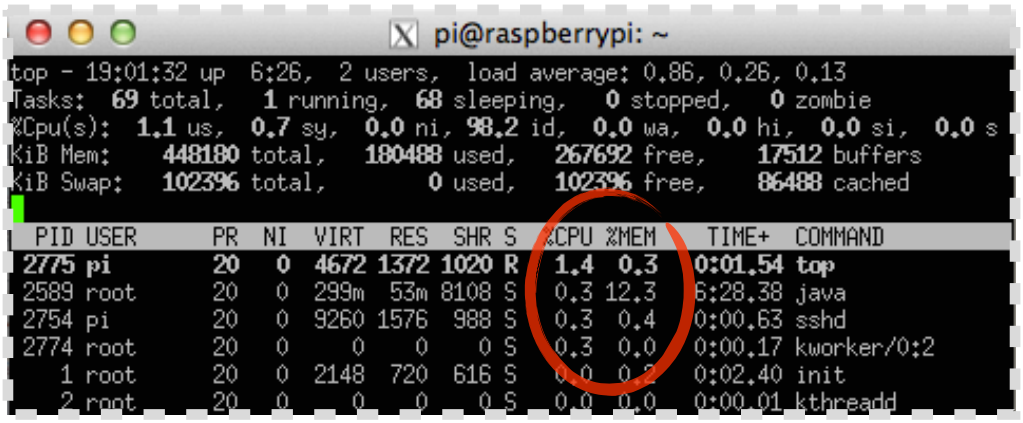

Figura 4.1: Desempenho do servidor em estado ocioso.

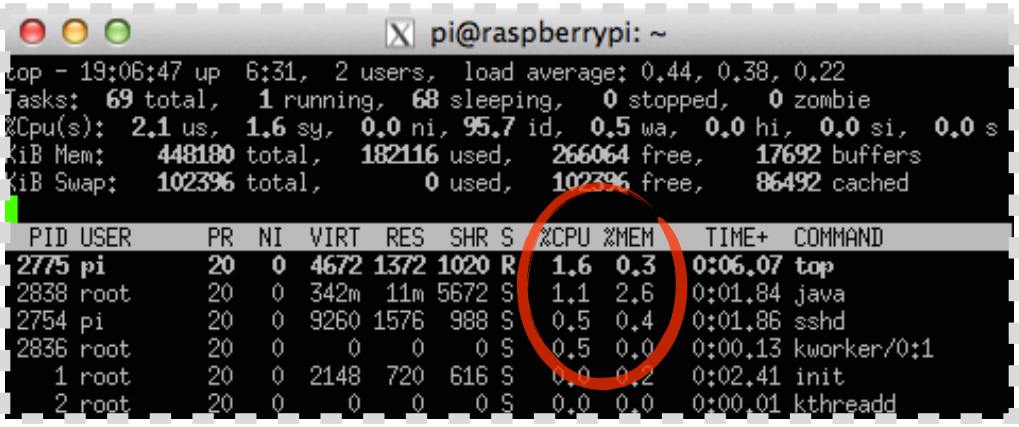

Figura 4.2: Desempenho do servidor em estado de execução.

Um novo caso analisado foi durante a requisição de um usuário para obter o acesso *web* em um determinado dispositivo. Nesse caso mencionado encontrou-se um problema grave com o servidor *Apache Tomcat* utilizado, tendo em vista que o processo para carregar a página completamente pode demorar até cinco minutos. Isso ocorre devido ao elevado uso da CPU da *Raspberry Pi* usada durante esse processo, conforme pode ser observado na Figura 4.3.

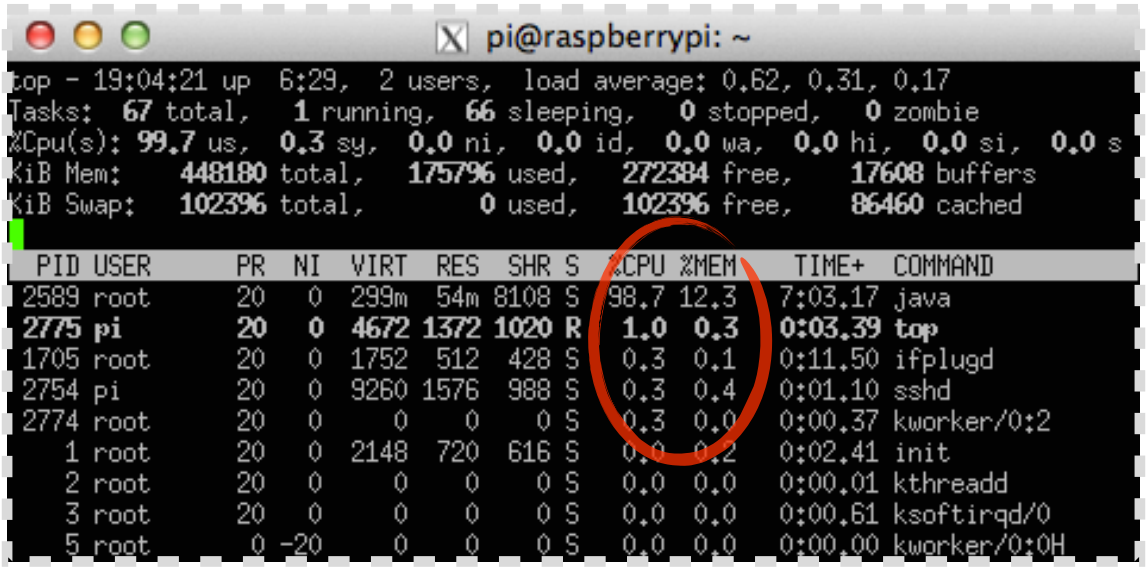

Figura 4.3: Desempenho do servidor quando solicitado acesso HTTP pela primeira vez.

Quando finalizada a inicialização da interface gráfica *HTTP* esse processamento retorna ao seu valor de funcionamento normal, abaixo de dois porcento, no entanto, toda vez que a página é requisitada essa problemática retorna. A alta porcentagem de uso da CPU não afeta em nada as

requisições realizadas por outros usuários. Todos os comandos requisitados por outros dispositivos foram executados durante a realização dos testes.

Outro resultado estudado foi a precisão obtida com o uso do emissor infravermelho. Para a realização destes testes foram utilizados diferentes ângulos de emissão, assim como foi alterada a distância do componente em relação ao receptor televisivo. Os resultados obtidos ao variar a distância entre os sensores, mantendo o ângulo direto de emissão, são apresentados no Gráfico 4.1.

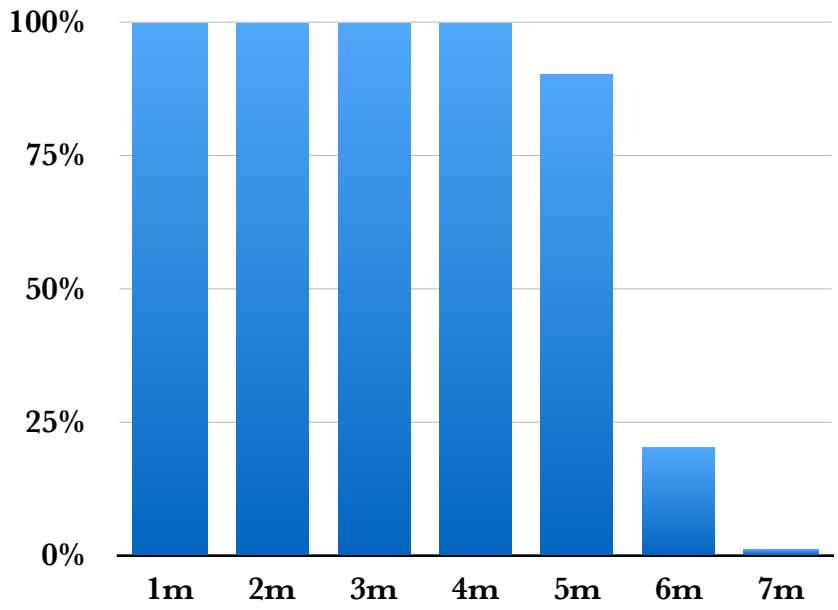

Gráfico 4.1: Porcentagem de acerto *vs* distância entre emissor e receptor.

Observando o Gráfico 4.1 é possível notar que o emissor infravermelho passa a não ser confiável para distâncias superiores à cinco metros. Além disso, durante a realização dos testes foi obtido que para distâncias de até três metros, ao variar o ângulo do emissor em até 30 graus a taxa de acerto ainda era elevada, no entanto, ao aumentar o ângulo essa taxa era reduzida bruscamente.

Por fim, foi analisada a precisão dos movimentos ao utilizar o Kinect. Para a realização destes testes foi selecionado um conjunto pequeno de pessoas que não possuíam grandes conhecimentos da tecnologia, sendo informado apenas os gestos necessários para a realização dos comandos. A média das porcentagens de acertos é exposta no Gráfico 4.2 adiante.

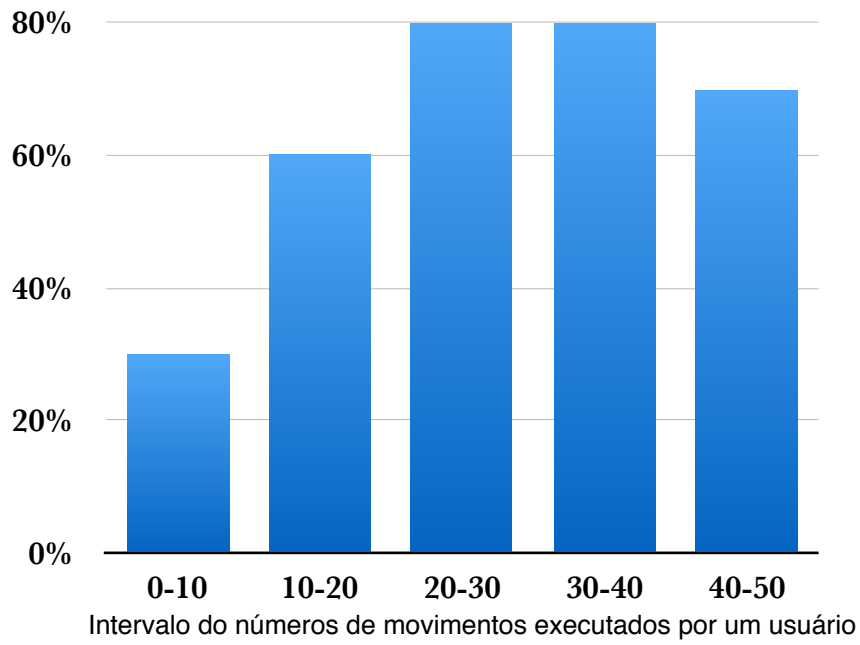

Gráfico 4.1: Porcentagem de acerto utilizando o *Kinect vs* quantidade de movimentos executados.

Com o Gráfico 4.1 é possível notar a falta de experiência do usuário com a interface nos primeiros gestos, no entanto, com um aprendizado maior a utilização do *Kinect* passou a apresentar bons resultados. Durante a execução dos testes algumas pequenas falhas ocorreram, como a perda das mãos pelo sistema de rastreamento do sensor, causadas pela velocidade de alguns movimentos. Outro ponto importante a destacar é a necessidade de uma iluminação apropriada para o uso do *Kinect*, sendo necessária a presença de luzes no ambiente, evitando a luz solar.

4.2 Dificuldades e Limitações

Inúmeras foram as dificuldades encontradas durante o desenvolvimento do projeto, a maioria delas ocorreu devido à falta de familiaridade com algumas tecnologias utilizadas, fazendo com que o progresso do projeto ocorresse de maneira mais lenta, por exemplo, a falta de conhecimento no desenvolvimento de aplicativos para o sistema operacional *iOS*.

Outra tecnologia que ofereceu grandes desafios foi a utilização do *Kinect* no *Mac OS X*. A dificuldade inicial surgiu da falta de conhecimento para a utilização das APIs fornecidas pelos

grupos de desenvolvimento especializado. No entanto, o maior desafio em relação ao uso deste dispositivo veio com o encerramento inesperado de todo o conteúdo disponibilizado no *website* do projeto *OpenNI*, ocorrido devido a compra do projeto pela *Apple*. Além desses, outros pequenos problemas ocorreram, por exemplo, a falta de iluminação adequada para o uso dos sensores.

Em relação a plataforma *Raspberry Pi* escolhida, as dificuldades ocorreram enquanto tentou-se utilizar o módulo *Wi-Fi USB* e durante o acesso ao servidor *Apache Tomcat*, o qual apresentou sobrecargas de processamento no servidor embarcado em função do *Java*, acarretando em elevado tempo de espera para a utilização deste serviço. Além disso, a implementação de *sockets* na *Raspberry Pi* foi dificultada devido a necessidade de comunicação com usuários codificados em outras linguagens.

Capítulo 5: Conclusões

Neste projeto foi implementado com sucesso uma arquitetura multiplataforma para o controle de um ambiente residencial, utilizando para isso um servidor embarcado responsável por centralizar todas as requisições *sockets* enviadas pelos usuários finais e atende-las, seja acionando os relés com os pinos de GPIO ou através de alterações televisivas produzidas com o uso do emissor infravermelho.

Comparando os resultados finais obtidos com os objetivos propostos no início do projeto pode-se notar o desempenho inferior do servidor *web* utilizado, acarretando em uma interface dificultada para o usuário final devido ao seu tempo de espera.

Quanto aos resultados obtidos em relação a distância limite para a utilização do emissor infravermelho à televisão e o número de acertos dos usuários na execução dos movimentos com o *Kinect*, todos foram considerados excelentes, tendo em vista que seis metros para ambientes residências como salas pode ser considerado longe, assim como uma taxa de acerto próxima de 80% no *Kinect* é considerada condizente devido a inexperiência do usuário testado.

Com exceção do *Apache Tomcat*, todas as tecnologias utilizadas pelos usuários obtiveram bons resultados práticos, apresentando uma interface de simples execução e com um bom tempo de resposta, considerado baixo para o tipo da atividade proposta, abaixo de um segundo para o processamento dos comandos. O uso do *Kinect* foi um grande facilitador para o controle televisivo, tendo em vista que não fez-se necessária a utilização de nenhum dispositivo físico para o envio de comandos. Além disso, o *Android* e o *iOS* demonstraram ser excelentes plataformas, com um vasto material de referência e ótimos ambientes de desenvolvimento, a exemplo do *Android Development Tools* e do *Xcode*, usados para a realização deste projeto.

Um dos desafios concluídos com êxito durante a implementação da arquitetura foi a comunicação por *sockets* entre os dispositivos multiplataformas e o servidor, sendo necessária a estruturação diferenciada para cada linguagem de programação.

Finalizando, em relação à *Raspberry Pi*, todo o processamento das requisições *sockets* enviadas e a execução das atividades usando os pinos GPIO ocorreram de forma perfeita, tornando-

a uma plataforma economicamente viável para a implementação do projeto como um todo, permitindo inclusive futuras expansões.

5.1 Conhecimentos Adquiridos

A realização do projeto como um todo possibilitou a utilização de diferentes conceitos adquiridos durante toda a graduação no curso de Engenharia de Computação. Além disso, incentivou ao aprendizado de novos tópicos, novas plataformas, como o uso do *Kinect* e do *iOS*.

Um desafio proposto pela arquitetura e que acarretou em grande contribuição pessoal foi o desenvolvimento de um sistema multiplataforma, o qual trouxe um conhecimento em diferentes linguagens de programação (*Java, C++ e Objective-C*), assim como a associação de todas elas ao utilizar a comunicação via *sockets*.

Outro conhecimento fundamental adquirido foi na implementação de aplicativos para os sistemas *Android* e *iOS*, visto que essas plataformas estão difundidas mundialmente na atualidade, sendo de grande contribuição para o desenvolvimento profissional. A maior diferença compreendida entras as duas foi a forma de trabalho de ambas. Enquanto para o desenvolvimento *Android* foi necessário em diversos momentos escrever códigos, editar arquivos XML, no *iOS* grande parte é realizada através da sua interface gráfica.

5.2 Trabalhos Futuros

Neste projeto há a possibilidade de implementação de diversas novas funcionalidades, visto que em um ambiente residencial o número de atividades ligadas ao controle e automação são inúmeras, por exemplo, a utilização de câmeras, fechaduras eletrônicas e sensores de presença. Além disso, para um aprimoramento do ambiente proposto pode-se aplicar algumas mudanças em relação ao servidor *web*, responsável pelo elevado processamento do servidor embarcado durante a requisição inicial do seu serviço.

Ainda no lado servidor, possíveis alterações podem ser realizadas em relação a segurança, considerando que a única barreira para a utilização dos controles trata-se do acesso correto ao roteador utilizado na rede doméstica, sendo necessária a implementação de mecanismos de segurança durante a comunicação cliente/servidor.

Outro fator a ser aprimorado são as exceções de programa, pois as aplicações executadas pelo *Android*, *iOS* e *Kinect* não fazem todas as verificações necessárias como deveriam, havendo casos que não são tratados, reduzindo assim a confiabilidade dos *softwares* produzidos em alguns pontos. É necessária a realização de testes estruturais e funcionais para uma melhor análise.

Na plataforma *Kinect*, há uma série de mudanças a serem realizadas. A primeira delas seria a utilização do novo modelo produzido pela *Microsoft*, tendo em vista que foi utilizado o modelo antigo para a realização do projeto. Outro fator importante seria a utilização do sensor de microfone do *Kinect*, permitindo assim a realização de comandos de voz e o uso de todo o seu poderio sensorial. Por fim, outra alteração seria a utilização de uma nova plataforma servidora, com um maior poder de processamento, o que poderia evitar assim a necessidade de um *notebook* na arquitetura para a simples atividade de processamento dos dados adquiridos pelos sensores do *Kinect*.

Referências

[1] Carlo Carrenho. (2013). *Brazil at the Digital Tipping Point.* Disponível em: <http://publishingperspectives.com/2013/10/brazil-at-the-digital-tipping-point/>. Acesso em 20 Maio 2014.

[2] David M. Rieder. (2013). *Present Tense*. Disponível em: <http://www.presenttensejournal.org/wp-content/uploads/2013/09/Rieder.pdf/>. Acesso em 21 Maio 2014.

[3] Edna Barros, Sérgio Cavalcante. *Introdução aos Sistemas Embarcados.* Disponível em: <http://www.cin.ufpe.br/~vba/periodos/8th/s.e/aulas/STP%20-%20Intro%20Sist%20Embarcados.pdf/>. Acesso em 21 Maio 2014.

[4] Controle Universal. *A História do Controle Universal.* Disponível em: <http://controle-universal.info/mos/view/A_história_do_controle_universal/>. Acesso em 17 Maio 2014.

[5] SB-Projects. *IR Remote Control Theory.* Disponível em: <http://www.sbprojects.com/knowledge/ir/index.php/>. Acesso em 25 Maio 2014.

[6] Damon Poeter. (2012). *Remote Control Inventor Eugene Polley.* Disponível em: <http://www.pcmag.com/article2/0,2817,2404761,00.asp/>. Acesso em 18 Maio 2014.

[7] SB-Projects. *Philips RC-5 Protocol.* Disponível em: <http://www.sbprojects.com/knowledge/ir/rc5.php/>. Acesso em 25 Maio 2014.

[8] Instituto Newton C Braga. *Tudo sobre relés*. Disponível em: <http://www.newtoncbraga.com.br/index.php/como-funciona/597-como-funcionam-os-reles?showall=1&limitstart=/>. Acesso em 26 Maio 2014.

[9] Raspberry Pi. *FAQS.* Disponível em: <http://www.raspberrypi.org/faqs/>. Acesso em 28 Maio 2014.

[10] Raspberry Pi. *Installing Operating System Images*. Disponível em: <http://www.raspberrypi.org/documentation/installation/installing-images/README.md/>. Acesso em 28 Maio 2014.

[11] Carl Ledbetter, Kit Hui. *Hardware at Microsoft, Kinect for Xbox 360.* Disponível em: <http://www.microsoft-careers.com/content/hardware/hardware-story-kinect/>. Acesso em 10 Maio 2014.

[12] Wired.com. *Kate Francis/Brown Bird Design.* Disponível em: <http://archive.wired.com/magazine/2011/06/mf_kinect/2/>. Acesso em 12 Maio 2014.

[13] John MacCormick. (2014). *How does the kinect work?*. Disponível em: <http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf>. Acesso em 15 Maio 2014.

[14] M.R. Andersen, T. Jensen, P. Lisouski, A.K. Mortensen, M.K. Hansen, T. Gregersen and P. Ahrendt: Kinect Depth Sensor Evaluation for Computer Vision Applications, 2012. Department of Engineering, Aarhus University. Denmark. 37 pp. - Technical report ECE-TR-6

[15] OpenNI. Disponível em: <http://www.openni.org/>. Acesso em 28 Março 2014.

[16] Alex Armstrong. (2014). *OpenNI to Close.* Disponível em: <http://www.i-programmer.info/news/194-kinect/7004-openni-to-close-.html/>. Acesso em 15 Maio 2014.

[17] BARROS, André Ricardo Gouveia. Implementação de um servidor utilizando Linux embarcado com acesso e gerenciamento através de smartphone. 2013. 1 v. TCC (Graduação) - Curso de Engenharia de Computação, Departamento de Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2013.

[18] Brad Ward. (2013). *Google: 900 million Android activations, 48 billion app installs.* Disponível em: <http://www.androidauthority.com/google-io-android-activations-210036/>. Acesso em 23 Maio 2014.

[19] Statista. (2014). *Numbers of available applications in the Google Play Store from December 2009 to July 2013.* Disponível em: <http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. Acesso em 23 Maio 2014.

[20] Priya Viswanathan. *The Apple iOS.* Disponível em: <http://mobiledevices.about.com/od/glossary/g/The-Apple-Ios.htm/>. Acesso em 27 Maio 2014.

[21] Alasdair Allan (2013). *Learning iOS Programming*. 3rd ed. United States of America: O'Reilly. 7-21.

[22] Apple Inc. (2014). *App Store Sales Top $10 Billion.* Disponível em: <http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html/>. Acesso em 18 Maio 2014.

[23] IDC Worldwide Mobile Phone Tracker. (2014). *Android and iOS Continue to Dominate the Worldwide Smartphone Market with Android Shipments Just Shy of 800 Million in 2013*. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS24676414>. Acesso em 25 Maio 2014.

[24] Aleksa Vukotic, James Goodwill (2011). *Apache Tomcat 7*. Apress. 1-17.

[25] Geoff Bryant. *Programming TCP/IP with Sockets.*. Disponível em: <http://www.opus1.com/www/presentations/nm056.pdf>. Acesso em 24 Maio 2014.

[26] MacPorts. *Quickstart.* Disponível em: <http://www.macports.org/install.php/>. Acesso em 26 Maio 2014.

[27] LIRC. *What is LIRC?*. Disponível em: <http://www.lirc.org/>. Acesso em 20 Maio 2014.

[28] Pi4J. *The Pi4J project.* Disponível em: <http://pi4j.com/>. Acesso em 27 Maio 2014.

Apéndice A - Código do Servidor

· rccommunication.java

```
package rcserver;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class rccommunication {

      public rccommunication () { }

      public rccommunication (int PORT) {
            this.PORT = PORT;
      }

      private ServerSocket providerSocket;
      private Socket connection = null;
      private OutputStreamWriter out;
      private InputStreamReader in;
      private String message;
      private int PORT;

      void run() throws InterruptedException{
        try {
            providerSocket = new ServerSocket(PORT, 10);
            System.out.println("Waiting for connection");
            connection = providerSocket.accept();
            System.out.println("\nConnection received from " +
                  connection.getInetAddress().getHostName());
            out = new OutputStreamWriter(connection.getOutputStream());
            out.flush();
            in = new InputStreamReader(connection.getInputStream());
            BufferedReader br = new BufferedReader(in);
            sendMessage("Connection successful");
            rccommands recev = new rccommands();
                  message = br.readLine();
                  System.out.println("  client>" + message);
            recev.command(message);

        catch(IOException ioException) {
            ioException.printStackTrace();
        }
        finally {
            try{
                in.close();
                out.close();
                providerSocket.close();
            }
            catch(IOException ioException){
                ioException.printStackTrace();
```

```
            }
        }
    }

      void sendMessage(String msg)
    {
        try{
            out.write(msg+"\n");
            out.flush();
            System.out.println("server>" + msg);
        }
        catch(IOException ioException){
            ioException.printStackTrace();
        }
    }

      public void start() throws InterruptedException{
        while(true){
            run();
        }
      }
}
```

· rccommands.java

```
package rcserver;

import java.io.IOException;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;
import com.pi4j.io.gpio.PinPullResistance;

public class rccommands {

    private static final GpioController gpio = GpioFactory.getInstance();
    private static final GpioPinDigitalOutput pin1
        = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, "Device1",
                PinState.HIGH);;
    private static final GpioPinDigitalOutput pin4
        = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_04, "Device2",
                PinState.HIGH);;
    private static final GpioPinDigitalOutput pin6
        = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_06, "Device3",
                PinState.HIGH);;

      public rccommands () { }

      public void command (String cmd) throws InterruptedException, IOException{

      Runtime run = Runtime.getRuntime();

            if (cmd.equals("pw")){
              run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_POWER");
              System.out.println("server> Power Button!");
            }
            else if (cmd.equals("up")){
              run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_CHANNELUP");
```

```
  System.out.println("server> Channel +");
}
else if (cmd.equals("dw")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_CHANNELDOWN");
  System.out.println("server> Channel -");
}
else if (cmd.equals("lf")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_VOLUMEDOWN");
  System.out.println("server> Volume -");
}
else if (cmd.equals("rg")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_VOLUMEUP");
  System.out.println("server> Volume +");
}
else if (cmd.equals("mt")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_MUTE");
  System.out.println("server> Mute");
}
else if (cmd.equals("al")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_C");
  System.out.println("server> Alternate");
}
else if (cmd.equals("b0")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_0");
  System.out.println("server> Button0");
}
else if (cmd.equals("b1")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_1");
  System.out.println("server> Button1");
}
else if (cmd.equals("b2")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_2");
  System.out.println("server> Button2");
}
else if (cmd.equals("b3")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_3");
  System.out.println("server> Button3");
}
else if (cmd.equals("b4")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_4");
  System.out.println("server> Button4");
}
else if (cmd.equals("b5")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_5");
  System.out.println("server> Button5");
}
else if (cmd.equals("b6")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_6");
  System.out.println("server> Button6");
}
else if (cmd.equals("b7")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_7");
  System.out.println("server> Button7");
}
else if (cmd.equals("b8")){
  run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_8");
  System.out.println("server> Button8");
}
else if (cmd.equals("b9")){
run.exec("irsend SEND_ONCE /home/pi/lircd.conf KEY_9");
```

```
            System.out.println("server> Button9");
          }
          else if (cmd.equals("n1")){
            pin1.low();
            System.out.println("server> Turn On Device1");
          }
          else if (cmd.equals("n2")){
            pin4.low();
            System.out.println("server> Turn On Device2");
          }
          else if (cmd.equals("n3")){
            pin6.low();
            System.out.println("server> Turn On Device3");
          }
          else if (cmd.equals("f1")){
            pin1.high();
            System.out.println("server> Turn Off Device1");
          }
          else if (cmd.equals("f2")){
            pin4.high();
            System.out.println("server> Turn Off Device2");
          }
          else if (cmd.equals("f3")){
            pin6.high();
            System.out.println("server> Turn Off Device3");
          }
          else if (cmd.equals("d1")){
            pin1.toggle();
            System.out.println("server> Toogle Device1");
          }
          else if (cmd.equals("d2")){
            pin4.toggle();
            System.out.println("server> Toogle Device2");
          }
          else if (cmd.equals("d3")){
            pin6.toggle();
            System.out.println("server> Toogle Device3");
          }
          else if (cmd.equals("hi")){
            System.out.println("server> Hello");
          }
          else {
            System.out.println("server> Incorrect command reveived.");
          }

        Thread.sleep(400);

      }
}
```

Apéndice B - Código do *Apache Tomcat*

· RemotecontrolserverhttpUI.java

```
package com.example.remotecontrolserverhttp;

import javax.servlet.annotation.WebServlet;

import com.vaadin.annotations.Theme;
import com.vaadin.annotations.VaadinServletConfiguration;
import com.vaadin.server.VaadinRequest;
import com.vaadin.server.VaadinServlet;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;
import com.vaadin.ui.GridLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.Panel;
import com.vaadin.ui.UI;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Window;

@SuppressWarnings("serial")
@Theme("remotecontrolserverhttp")
public class RemotecontrolserverhttpUI extends UI implements ClickListener {

      private int PORT = 1500;

      @WebServlet(value = "/*", asyncSupported = true)
      @VaadinServletConfiguration(productionMode = false, ui =
            RemotecontrolserverhttpUI.class)
      public static class Servlet extends VaadinServlet {
      }

      @Override
      protected void init(VaadinRequest request) {

        final VerticalLayout layoutVertical = new VerticalLayout();
        layoutVertical.addStyleName("pstyle");
        layoutVertical.setWidth("60%");


        final GridLayout layout = new GridLayout(7, 13);

        Button button1 = new Button("PWR");
        button1.addListener(this);
        layout.addComponent(button1, 0, 0);

        Button button2 = new Button("CH-");
        button2.addListener(this);
        layout.addComponent(button2, 2, 0, 3, 0);

        Button button3 = new Button("CH+");
        button3.addListener(this);
        layout.addComponent(button3, 5, 0, 6, 0);
```

```
Button button4 = new Button("MUTE");
button4.addListener(this);
layout.addComponent(button4, 0, 2);

Button button5 = new Button("VL-");
button5.addListener(this);
layout.addComponent(button5, 2, 2, 3, 2);

Button button6 = new Button("VL+");
button6.addListener(this);
layout.addComponent(button6, 5, 2, 6, 2);

Button button7 = new Button("1");
button7.addListener(this);
layout.addComponent(button7, 1, 4);

Button button8 = new Button("2");
button8.addListener(this);
layout.addComponent(button8, 3, 4);

Button button9 = new Button("3");
button9.addListener(this);
layout.addComponent(button9, 5, 4);

Button button10 = new Button("4");
button10.addListener(this);
layout.addComponent(button10, 1, 6);

Button button11 = new Button("5");
button11.addListener(this);
layout.addComponent(button11, 3, 6);

Button button12 = new Button("6");
button12.addListener(this);
layout.addComponent(button12, 5, 6);

Button button13 = new Button("7");
button13.addListener(this);
layout.addComponent(button13, 1, 8);

Button button14 = new Button("8");
button14.addListener(this);
layout.addComponent(button14, 3, 8);

Button button15 = new Button("9");
button15.addListener(this);
layout.addComponent(button15, 5, 8);

Button button16 = new Button("0");
button16.addListener(this);
layout.addComponent(button16, 3, 10);

Button button17 = new Button("ALT");
button17.addListener(this);
layout.addComponent(button17, 5, 10);

final GridLayout layout2 = new GridLayout(3, 1);

Button button18 = new Button("DV1");
```

```
    button18.addListener(this);
    layout2.addComponent(button18, 0, 0);

    Button button19 = new Button("DV2");
    button19.addListener(this);
    layout2.addComponent(button19, 1, 0);

    Button button20 = new Button("DV3");
    button20.addListener(this);
    layout2.addComponent(button20, 2, 0);

    layoutVertical.addComponent(new Label(" ", Label.CONTENT_XHTML));
    layoutVertical.addComponent(new Label("UNIv"));
    layoutVertical.addComponent(new Label("Controller\n\n"));
    layoutVertical.addComponent(new Label(" ", Label.CONTENT_XHTML));
    layoutVertical.addComponent(layout);
    layoutVertical.addComponent(new Label(" ", Label.CONTENT_XHTML));
    layoutVertical.addComponent(layout2);
    layoutVertical.addComponent(new Label(" ", Label.CONTENT_XHTML));
    layoutVertical.addComponent(new Label(" ", Label.CONTENT_XHTML));

    Label dsg = new Label("designed by guilherme jabur");
    dsg.addStyleName("pline");
    layoutVertical.addComponent(dsg);

    layoutVertical.setComponentAlignment(layout, Alignment.MIDDLE_CENTER);

    setContent(layoutVertical);

  }

public void buttonClick(ClickEvent event) {
    // Get the button that was clicked
    Button button = event.getButton();

    // Calculate the new value
    SendMessageControl(button);
}

private void SendMessageControl(Button button) {

  char p0 = button.getCaption().charAt(0);

  if (p0 == 'P'){
        clientcommunication client = new clientcommunication(PORT, "pw");
              client.start();
  }
  else if (p0 == 'C'){
        char p2 = button.getCaption().charAt(2);
        if (p2 == '+'){
              clientcommunication client = new clientcommunication(PORT,
                    "up");
              client.start();
        }
        else {
              clientcommunication client = new clientcommunication(PORT,
                    "dw");
              client.start();
        }
  }
```

```
else if (p0 == 'D'){
      char p2 = button.getCaption().charAt(2);
      if (p2 == '1'){
            clientcommunication client = new clientcommunication(PORT,
                  "d1");
            client.start();
      }
      else if (p2 == '2'){
            clientcommunication client = new clientcommunication(PORT,
                  "d2");
            client.start();
      }
      else {
            clientcommunication client = new clientcommunication(PORT,
                  "d3");
            client.start();
      }
}
else if (p0 == 'M'){
            clientcommunication client = new clientcommunication(PORT,
                  "mt");
            client.start();
}
else if (p0 == 'V'){
      char p2 = button.getCaption().charAt(2);
      if (p2 == '+'){
            clientcommunication client = new clientcommunication(PORT,
                  "rg");
            client.start();
      }
      else {
            clientcommunication client = new clientcommunication(PORT,
                  "lf");
            client.start();
      }
}
else if (p0 == 'A'){
            clientcommunication client = new clientcommunication(PORT,
                  "al");
            client.start();
}
else if (p0 == '0'){
            clientcommunication client = new clientcommunication(PORT,
                  "b0");
            client.start();
}
else if (p0 == '1'){
            clientcommunication client = new clientcommunication(PORT,
                  "b1");
            client.start();
}
else if (p0 == '2'){
            clientcommunication client = new clientcommunication(PORT,
                  "b2");
            client.start();
}
else if (p0 == '3'){
            clientcommunication client = new clientcommunication(PORT,
                  "b3");
            client.start();
```

```
      }
      else if (p0 == '4'){
                  clientcommunication client = new clientcommunication(PORT,
                        "b4");
                  client.start();
      }
      else if (p0 == '5'){
                  clientcommunication client = new clientcommunication(PORT,
                        "b5");
                  client.start();
      }
      else if (p0 == '6'){
                  clientcommunication client = new clientcommunication(PORT,
                        "b6");
                  client.start();
      }
      else if (p0 == '7'){
                  clientcommunication client = new clientcommunication(PORT,
                        "b7");
                  client.start();
      }
      else if (p0 == '8'){
                  clientcommunication client = new clientcommunication(PORT,
                        "b8");
                  client.start();
      }
      else if (p0 == '9'){
                  clientcommunication client = new clientcommunication(PORT,
                        "b9");
                  client.start();
      }
    }
}
```

· clientcommunication.java

```
package com.example.remotecontrolserverhttp;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.net.UnknownHostException;

public class clientcommunication {

      public clientcommunication () { }

      public clientcommunication (int PORT, String str) {
            this.PORT = PORT;
            this.cmd = str;
      }

      private Socket requestSocket;
      private OutputStreamWriter out;
      private InputStreamReader in;
      private String message;
```

```
  private int PORT;
  private String cmd;

void run()
{
    try{
        requestSocket = new Socket("127.0.0.1", PORT);
        System.out.println("Connected to localhost in port " + PORT);
        out = new OutputStreamWriter(requestSocket.getOutputStream());
        out.flush();
        in = new InputStreamReader(requestSocket.getInputStream());

        BufferedReader br = new BufferedReader(in);

        message = br.readLine();
              sendMessage(cmd);

    }
    catch(UnknownHostException unknownHost){
        System.err.println("You are trying to connect to an unknown host!");
    }
    catch(IOException ioException){
        ioException.printStackTrace();
    }
    finally{
        try{
            in.close();
            out.close();
            requestSocket.close();
        }
        catch(IOException ioException){
            ioException.printStackTrace();
        }
    }
}

void sendMessage(String msg)
{
    try{
        out.write(msg+"\n");
        out.flush();
    }
    catch(IOException ioException){
        ioException.printStackTrace();
    }
}

public void start() {
    run();
}

}
```

Apéndice C - Código do *Android*

· StartSimpleActivity.java

```
package com.example.univcontroller;

import com.example.univcontroller.R;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class StartSimpleActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_startsimple);
    }

    public void onClickUp(View v){
        AndroidCommunication c = new AndroidCommunication("up");
        c.start();

        while (!c.getIsReady()) {}
        Toast.makeText(getApplicationContext(), "Channel +" ,
Toast.LENGTH_LONG).show();
    }

    public void onClickDown(View v){
        AndroidCommunication c = new AndroidCommunication("dw");
        c.start();

        while (!c.getIsReady()) {}
        Toast.makeText(getApplicationContext(), "Channel -" ,
Toast.LENGTH_LONG).show();
    }

    public void onClickRight(View v){
        AndroidCommunication c = new AndroidCommunication("rg");
        c.start();

        while (!c.getIsReady()) {}
        Toast.makeText(getApplicationContext(), "Volume +" ,
Toast.LENGTH_LONG).show();
    }

    public void onClickLeft(View v){
        AndroidCommunication c = new AndroidCommunication("lf");
        c.start();

        while (!c.getIsReady()) {}
        Toast.makeText(getApplicationContext(), "Volume -" ,
Toast.LENGTH_LONG).show();
    }
```

```
    public void onClickPower(View v){
        AndroidCommunication c = new AndroidCommunication("pw");
        c.start();

        while (!c.getIsReady()) {}
        Toast.makeText(getApplicationContext(), "Power Button" ,
Toast.LENGTH_LONG).show();
    }

}
```

· StartRelayActivity.java

```
package com.example.univcontroller;

import com.example.univcontroller.R;
import android.app.Activity;
import android.os.Bundle;
import android.widget.CompoundButton;
import android.widget.Toast;
import android.widget.ToggleButton;

public class StartRelayActivity extends Activity {

      private ToggleButton button1;
      private ToggleButton button2;
      private ToggleButton button3;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_startrelay);

        ListenerButton();
    }

    public void ListenerButton(){

      button1 = (ToggleButton) findViewById(R.id.onButtonDevice1);
      button2 = (ToggleButton) findViewById(R.id.onButtonDevice2);
      button3 = (ToggleButton) findViewById(R.id.onButtonDevice3);

          button1.setOnCheckedChangeListener(new
                  CompoundButton.OnCheckedChangeListener() {
              public void onCheckedChanged(CompoundButton buttonView, boolean
                        isChecked) {
                  if (isChecked) {
                        AndroidCommunication c = new AndroidCommunication("n1");
                        c.start();

                        while (!c.getIsReady()) {}
                        Toast.makeText(getApplicationContext(), "Turn on Device
                              1" , Toast.LENGTH_LONG).show();
                  } else {
                        AndroidCommunication c = new AndroidCommunication("f1");
                        c.start();

                        while (!c.getIsReady()) {}
                        Toast.makeText(getApplicationContext(), "Turn off Device
                              1" , Toast.LENGTH_LONG).show();
```

```
                }
            }
        });

        button2.setOnCheckedChangeListener(new
                CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView, boolean
                    isChecked) {
                if (isChecked) {
                    AndroidCommunication c = new AndroidCommunication("n2");
                    c.start();

                    while (!c.getIsReady()) {}
                    Toast.makeText(getApplicationContext(), "Turn on Device
                            2" , Toast.LENGTH_LONG).show();
                } else {
                    AndroidCommunication c = new AndroidCommunication("f2");
                    c.start();

                    while (!c.getIsReady()) {}
                    Toast.makeText(getApplicationContext(), "Turn off Device
                            2" , Toast.LENGTH_LONG).show();
                }
            }
        });

        button3.setOnCheckedChangeListener(new
                CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView, boolean
                    isChecked) {
                if (isChecked) {
                    AndroidCommunication c = new AndroidCommunication("n3");
                    c.start();

                    while (!c.getIsReady()) {}
                    Toast.makeText(getApplicationContext(), "Turn on Device
                            3" , Toast.LENGTH_LONG).show();
                } else {
                    AndroidCommunication c = new AndroidCommunication("f3");
                    c.start();

                    while (!c.getIsReady()) {}
                    Toast.makeText(getApplicationContext(), "Turn off Device
                            3" , Toast.LENGTH_LONG).show();
                }
            }
        });

    }

}
```

Apéndice D - Código do *iOS*

· SimpleControlViewController.m

```
#import "SimpleControlViewController.h"

@interface SimpleControlViewController ()

@end

@implementation SimpleControlViewController

@synthesize inputStream, outputStream;

- (void)sendNetworkCommunication:(NSString*)command {
    CFReadStreamRef readStream;
    CFWriteStreamRef writeStream;
    CFStreamCreatePairWithSocketToHost(NULL, (CFStringRef)@"192.168.0.200",
            1500, &readStream, &writeStream);
    inputStream = (__bridge NSInputStream *)readStream;
    outputStream = (__bridge NSOutputStream *)writeStream;
    [inputStream setDelegate:self];
    [outputStream setDelegate:self];
    [inputStream open];
    [outputStream open];

    NSString *response  = [NSString stringWithFormat:@"%@", command];
      NSData *data = [[NSData alloc] initWithData:[response
dataUsingEncoding:NSASCIIStringEncoding]];
      [outputStream write:[data bytes] maxLength:[data length]];

    [inputStream close];
      [outputStream close];
}

- (IBAction)powerButton:(id)sender {
    [self sendNetworkCommunication:@"pw"];
}

- (IBAction)chUpButton:(id)sender {
    [self sendNetworkCommunication:@"up"];
}

- (IBAction)chDownButton:(id)sender {
    [self sendNetworkCommunication:@"dw"];
}

- (IBAction)vlUpButton:(id)sender {
    [self sendNetworkCommunication:@"rg"];
}

- (IBAction)vlDownButton:(id)sender {
    [self sendNetworkCommunication:@"lf"];
}

- (IBAction)CloseSimpleControl:(id)sender {
```

```
    [self dismissModalViewControllerAnimated: YES];
}

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    [self sendNetworkCommunication:@"hi"];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*
#pragma mark - Navigation

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
}
*/

@end
```

· SimpleControlViewController.h

```
#import <UIKit/UIKit.h>


@interface SimpleControlViewController : UIViewController <NSStreamDelegate> {

    NSInputStream *inputStream;
    NSOutputStream *outputStream;

}

@property (nonatomic, retain) NSInputStream *inputStream;
@property (nonatomic, retain) NSOutputStream *outputStream;

- (void)sendNetworkCommunication:(NSString*)command;
- (IBAction)powerButton:(id)sender;
- (IBAction)chUpButton:(id)sender;
- (IBAction)chDownButton:(id)sender;
- (IBAction)vlUpButton:(id)sender;
- (IBAction)vlDownButton:(id)sender;
- (IBAction)CloseSimpleControl:(id)sender;

@end
```

Apéndice E - Código do *Kinect*

· RemoteControlTV.cpp

```
#include "RemoteControlTV.h"
#include "socketClient.h"
#include <cassert>
#include <string>
#include <math.h>
#include <sys/socket.h>
#include <time.h>


using namespace xn;

#define LENGTHOF(arr)               (sizeof(arr)/sizeof(arr[0]))
#define FOR_ALL(arr, action)  {for(int i = 0; i < LENGTHOF(arr); ++i)
{action(arr[i])}}

#define ADD_GESTURE(name)           {if(m_GestureGenerator.AddGesture(name,
NULL) != XN_STATUS_OK){printf("Unable to add gesture"); exit(1);}}
#define REMOVE_GESTURE(name)  {if(m_GestureGenerator.RemoveGesture(name) !=
XN_STATUS_OK){printf("Unable to remove gesture"); exit(1);}}

#define ADD_ALL_GESTURES            FOR_ALL(cGestures, ADD_GESTURE)
#define REMOVE_ALL_GESTURES         FOR_ALL(cGestures, REMOVE_GESTURE)

#define SIZE_ARRAY_SWIPE        5
#define MIN_X_DELTA             70
#define MAX_X_DELTA             40
#define MIN_Y_DELTA             70
#define MAX_Y_DELTA             40

static char                 cCommand[] = "pw";

// Gestures to track
static const char             cClickStr[] = "Click";
static const char             cWaveStr[] = "Wave";
static const char             cSwipeRightStr[] = "Swipe Right";
static const char             cSwipeLeftStr[] = "Swipe Left";
static const char             cSwipeUpStr[] = "Swipe Up";
static const char             cSwipeDownStr[] = "Swipe Down";
static const char* const      cGestures[] =
{
      cClickStr,
      cWaveStr
};

XnListT<HandTracker*>   HandTracker::sm_Instances;

static float  ArrayPositionX[SIZE_ARRAY_SWIPE];
static float  ArrayPositionY[SIZE_ARRAY_SWIPE];
static float  ArrayPositionZ[SIZE_ARRAY_SWIPE];
static int  nHands=0;
static int  flag=0;
static time_t timePresent, timeCommand;
```

```
void XN_CALLBACK_TYPE HandTracker::Gesture_Recognized(
xn::GestureGenerator&   /*generator*/,

      const XnChar*                 strGesture,

      const XnPoint3D*        /*pIDPosition*/,

      const XnPoint3D*        pEndPosition,

      void*                         pCookie)
{
      printf("Gesture recognized: %s\n", strGesture);

    if ((!strcmp (strGesture, cWaveStr)) && flag != 0){
        if (nHands > 0)
            printf("Hand %d do not control TV anymore!\n", nHands);

        else printf("Hand %d controlling TV!\n", -nHands);

        nHands = -nHands;
    }

    else if ((!strcmp (strGesture, cClickStr)) && flag != 0){
        strcpy (cCommand, "pw");
        class socketClient sckt;

        sckt.connectingSocket();
        sckt.receiveMessage();
        sckt.sendMessage(cCommand);
        sckt.closeSocket();
    }

    HandTracker*  pThis = static_cast<HandTracker*>(pCookie);
    if(sm_Instances.Find(pThis) == sm_Instances.End())
    {
        printf("Dead HandTracker: skipped!\n");
        return;
    }

    pThis->m_HandsGenerator.StartTracking(*pEndPosition);
}

void XN_CALLBACK_TYPE HandTracker::Hand_Create( xn::HandsGenerator& /
      *generator*/,XnUserID nId, const XnPoint3D* pPosition, XnFloat /*fTime*/,
            void* pCookie)
{
      printf("New Hand: %d @ (%f,%f,%f)\n", nId, pPosition->X, pPosition->Y,
pPosition->Z);

    flag++;
    nHands = nId;
    setArrayPosition(nId, pPosition);

      HandTracker*      pThis = static_cast<HandTracker*>(pCookie);
      if(sm_Instances.Find(pThis) == sm_Instances.End())
      {
            printf("Dead HandTracker: skipped!\n");
            return;
      }
```

```
      pThis->m_History[nId].Push(*pPosition);
}

void XN_CALLBACK_TYPE HandTracker::Hand_Update( xn::HandsGenerator& /
      *generator*/, XnUserID nId, const XnPoint3D* pPosition, XnFloat
            /*fTime*/, void* pCookie)
{
      HandTracker*      pThis = static_cast<HandTracker*>(pCookie);
      if(sm_Instances.Find(pThis) == sm_Instances.End())
      {
            printf("Dead HandTracker: skipped!\n");
            return;
      }

      // Add to this user's hands history
      TrailHistory::Iterator it = pThis->m_History.Find(nId);
      if (it == pThis->m_History.End())
      {
            printf("Dead hand update: skipped!\n");
            return;
      }

    if (nId == nHands){
        refreshArrayPosition (nId, pPosition);

        int result = DetectSwipe(nId, pPosition);

        timePresent = time(NULL);
        if (timePresent-timeCommand > 1){
            if (result == 1){
                printf("Moviment Swipe: %s \n", cSwipeRightStr);
                strcpy (cCommand, "up");
            }

            else if (result == 2){
                printf("Moviment Swipe: %s \n", cSwipeLeftStr);
                strcpy (cCommand, "dw");
            }

            else if (result == 3){
                printf("Moviment Swipe: %s \n", cSwipeUpStr);
                strcpy (cCommand, "rg");
            }

            else if (result == 4){
                printf("Moviment Swipe: %s \n", cSwipeDownStr);
                strcpy (cCommand, "lf");
            }

            if (result != 0){
                timeCommand = time(NULL);
                setArrayPosition(nId, pPosition);
                class socketClient sckt;

                sckt.connectingSocket();
                sckt.receiveMessage();
                sckt.sendMessage(cCommand);
                sckt.closeSocket();
            }
```

```
        }
    }

    //printf("HandPosition: %d @ (%f,%f,%f)\n", nId, pPosition->X, pPosition->Y,
pPosition->Z);
      it->Value().Push(*pPosition);
}

void XN_CALLBACK_TYPE HandTracker::Hand_Destroy(      xn::HandsGenerator& /
*generator*/, XnUserID nId, XnFloat /*fTime*/, void* pCookie)
{
      printf("Lost Hand: %d\n", nId);

    flag--;
      HandTracker*      pThis = static_cast<HandTracker*>(pCookie);
      if(sm_Instances.Find(pThis) == sm_Instances.End())
      {
            printf("Dead HandTracker: skipped!\n");
            return;
      }

      // Remove this user from hands history
      pThis->m_History.Remove(nId);
}


HandTracker::HandTracker(xn::Context& context) : m_rContext(context)
{
      // Track all living instances (to protect against calling dead pointers in
the Gesture/Hand Generator hooks)
      XnStatus rc = sm_Instances.AddLast(this);
      if (rc != XN_STATUS_OK)
      {
            printf("Unable to add NiHandTracker instance to the list.");
            exit(1);
      }
}

HandTracker::~HandTracker()
{
      // Remove the current instance from living instances list
      XnListT<HandTracker*>::ConstIterator it = sm_Instances.Find(this);
      assert(it != sm_Instances.End());
      sm_Instances.Remove(it);
}

XnStatus HandTracker::Init()
{
      XnStatus                rc;
      XnCallbackHandle  chandle;

      // Create generators
      rc = m_GestureGenerator.Create(m_rContext);
      if (rc != XN_STATUS_OK)
      {
            printf("Unable to create GestureGenerator.");
            return rc;
      }

      rc = m_HandsGenerator.Create(m_rContext);
```

```
	if (rc != XN_STATUS_OK)
	{
		printf("Unable to create HandsGenerator.");
		return rc;
	}

	rc = m_GestureGenerator.RegisterGestureCallbacks(Gesture_Recognized,
Gesture_Process, this, chandle);
	if (rc != XN_STATUS_OK)
	{
		printf("Unable to register gesture callbacks.");
		return rc;
	}

	rc = m_HandsGenerator.RegisterHandCallbacks(Hand_Create, Hand_Update,
Hand_Destroy, this, chandle);
	if (rc != XN_STATUS_OK)
	{
		printf("Unable to register hand callbacks.");
		return rc;
	}

	return XN_STATUS_OK;
}

XnStatus HandTracker::Run()
{
	XnStatus    rc = m_rContext.StartGeneratingAll();
	if (rc != XN_STATUS_OK)
	{
		printf("Unable to start generating.");
		return rc;
	}

	ADD_ALL_GESTURES;

	return XN_STATUS_OK;
}

void HandTracker::setArrayPosition (XnUserID    nId, const XnPoint3D*
pPosition){
    for (int i=0; i<SIZE_ARRAY_SWIPE; i++){
        ArrayPositionX[i] = pPosition->X;
        ArrayPositionY[i] = pPosition->Y;
        ArrayPositionZ[i] = pPosition->Z;
    }
}

void HandTracker::refreshArrayPosition (XnUserID nId, const XnPoint3D*
pPosition){
    for (int i=1; i<SIZE_ARRAY_SWIPE; i++){
        ArrayPositionX[i] = ArrayPositionX[i-1];
        ArrayPositionY[i] = ArrayPositionY[i-1];
        ArrayPositionZ[i] = ArrayPositionZ[i-1];
    }
    ArrayPositionX[0] = pPosition->X;
    ArrayPositionY[0] = pPosition->Y;
    ArrayPositionZ[0] = pPosition->Z;
}
```

```
int HandTracker::DetectSwipe (XnUserID    nId, const XnPoint3D*   pPosition){
    int i=0;

    float x0 = ArrayPositionX[0];
    float xn = ArrayPositionX[SIZE_ARRAY_SWIPE-1];
    float y0 = ArrayPositionY[0];
    float yn = ArrayPositionY[SIZE_ARRAY_SWIPE-1];

    if (fabs(xn-x0) > MIN_X_DELTA && fabs(yn-y0) < MAX_Y_DELTA){
        if (x0 > xn) return 1;
        else return 2;
    }

    if (fabs(xn-x0) < MAX_X_DELTA && fabs(yn-y0) > MIN_Y_DELTA){
        if (y0 > yn) return 3;
        else return 4;
    }

    return 0;
}
```

· RemoteControlTV.h

```
#ifndef REMOTE_CONTROL_TV_H__
#define REMOTE_CONTROL_TV_H__

#include <XnCppWrapper.h>
#include <XnCyclicStackT.h>
#include <XnHashT.h>

// Hand position history length (positions)
#define MAX_HAND_TRAIL_LENGTH 10

typedef XnCyclicStackT<XnPoint3D, MAX_HAND_TRAIL_LENGTH> Trail;
typedef XnHashT<XnUserID, Trail> TrailHistory;

class HandTracker
{
public:
      HandTracker(xn::Context& context);
      ~HandTracker();

      XnStatus Init();
      XnStatus Run();

      const TrailHistory&     GetHistory()      const {return m_History;}

private:
      // OpenNI Gesture and Hands Generator callbacks
      static void XN_CALLBACK_TYPE Gesture_Recognized(xn::GestureGenerator&
generator, const XnChar* strGesture, const XnPoint3D* pIDPosition, const
XnPoint3D* pEndPosition, void* pCookie);
      static void XN_CALLBACK_TYPE Gesture_Process(   xn::GestureGenerator&   /
*generator*/, const XnChar* /*strGesture*/, const XnPoint3D* /*pPosition*/,
XnFloat /*fProgress*/, void* /*pCookie*/) {}
      static void XN_CALLBACK_TYPE Hand_Create( xn::HandsGenerator& generator,
XnUserID nId, const XnPoint3D* pPosition, XnFloat fTime, void* pCookie);
      static void XN_CALLBACK_TYPE Hand_Update( xn::HandsGenerator& generator,
XnUserID nId, const XnPoint3D* pPosition, XnFloat fTime, void* pCookie);
```

```
      static void XN_CALLBACK_TYPE Hand_Destroy( xn::HandsGenerator& generator,
XnUserID nId, XnFloat fTime, void* pCookie);

      xn::Context&                  m_rContext;
      TrailHistory                  m_History;
      xn::GestureGenerator    m_GestureGenerator;
      xn::HandsGenerator            m_HandsGenerator;

      static XnListT<HandTracker*>  sm_Instances;     // Living instances of the
class

    static int DetectSwipe  (XnUserID     nId, const XnPoint3D*   pPosition);
   // Detect swipe gestures, return: (1) right, (2) left, (3) up, (4) down

    static void setArrayPosition   (XnUserID    nId, const XnPoint3D*
pPosition);
                        // Start array of position with start hand position

    static void refreshArrayPosition   (XnUserID      nId, const XnPoint3D*
pPosition);
                        // Add new position and refresh array of position


private:
      XN_DISABLE_COPY_AND_ASSIGN(HandTracker);
};

#endif //REMOTE_CONTROL_TV_H__
```