CENTRO UNIVERSITÁRIO DE BRASÍLIA -UniCEUB

CURSO DE ENGENHARIA DE COMPUTAÇÃO

SERGIO OLIVEIRA BERGMANN

ACESSIBILIDADE ATRAVÉS DA AUTOMAÇÃO RESIDENCIAL

Orientador: Prof. MsC. Francisco Javier De Obaldia Diaz

Brasília

Novembro, 2013

SERGIO OLIVEIRA BERGMANN

ACESSIBILIDADE ATRAVÉS DA AUTOMAÇÃO RESIDENCIAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. MsC. Francisco Javier De Obaldia Diaz

Brasília

Novembro, 2013

SERGIO OLIVEIRA BERGMANN

ACESSIBILIDADE ATRAVÉS DA AUTOMAÇÃO RESIDENCIAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. MsC. Francisco Javier De Obaldia Diaz

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Abiezer Amarilia Fernandes
Coordenador do Curso

Banca Examinadora:

_____

Prof. Francisco Javier De Obaldia Diaz
Mestre, UniCEUB

_____

Profª. Layany Zambrano Horta Damázio
Mestre, UniCEUB

_____

Prof. Luciano Henrique Duque
Mestre, UniCEUB

_____

Prof. Marco Antônio Araújo
Mestre, UniCEUB

DEDICATÓRIA

Dedico este trabalho a minha noiva Jaqueline Osiro que me apoiou durante todo o curso de graduação, fazendo entender que às vezes é preciso abrir mão de alguns momentos para conquistar objetivos maiores.

AGRADECIMENTOS

Agradeço a minha noiva Jaqueline Osiro pelo apoio prestado durante a elaboração deste trabalho. Aos meus orientadores do PIC (Programa de Iniciação Científica): Carlo Kleber, professor doutor do UniCEUB e Marco Antônio, professor mestre do UniCEUB, que contribuíram de forma significativa para o desenvolvimento da minha capacidade de pesquisa e escrita durante o PIC.

“O desejo de ir em direção ao outro, de se comunicar com ele, ajudá-lo de forma eficiente, faz nascer em nós uma imensa energia e uma grande alegria, sem nenhuma sensação de cansaço.”

Dalai Lama

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO 13
1.1 - APRESENTAÇÃO DO PROBLEMA 13
1.2 - OBJETIVOS DO TRABALHO 13
1.3 - JUSTIFICATIVA E IMPORTÂNCIA DO TRABALHO 13
1.4 - ESCOPO DO TRABALHO 14
1.5 - RESULTADOS ESPERADOS 15
1.6 - ESTRUTURA DO TRABALHO 15

CAPÍTULO 2 - APRESENTAÇÃO DO PROBLEMA 16

CAPÍTULO 3 - REFERENCIAL TEÓRICO 23
3.1 - INFRAESTRUTURA E *HARDWARE* 23
3.1.1 - *Power Line Communication* (PLC) 23
3.1.2 - Modem PLC 23
3.1.3 - Microcontrolador 24
3.1.4 - Mini-PC 26
3.1.5 - Relé 27
3.1.6 - NFC (*Near Field Communication*) 28
3.2 - ARQUITETURA E *SOFTWARE* 29
3.2.1 - *Web Service* 29
3.2.2 - Linguagem de programação 29
3.2.2.1 - Linguagem Java 29
3.2.2.2 - Linguagem Python 30
3.2.3 - Sistema Operacional 31
3.2.4 - *Microsoft Kinect* 32
3.2.5 - *Framework OpenNI* 33
3.2.6 - *Framework libfreenect* (*OpenKinect*) 34

CAPÍTULO 4 - SOLUÇÃO PROPOSTA 36
4.1 - APRESENTAÇÃO GERAL DO SOLUÇÃO PROPOSTA 36
4.2 - DESCRIÇÃO DAS ETAPAS DA SOLUÇÃO PROPOSTA 36
4.3 - DESCRIÇÃO DA IMPLEMENTAÇÃO 37
4.3.1 - Servidor Residencial 37
4.3.2 - Módulos de Automação (Atuadores) 40
4.3.3 - Módulos Cliente 42
4.3.3.1 - Módulo *Kinect* 42
4.3.3.2 - Módulo *Smartphone* 46

CAPÍTULO 5 - APLICAÇÃO PRÁTICA DA SOLUÇÃO PROPOSTA 49
5.1 - APRESENTAÇÃO DA ÁREA DE APLICAÇÃO DA SOLUÇÃO 49
5.2 - DESCRIÇÃO DA APLICAÇÃO DA SOLUÇÃO 49
5.3 - RESULTADOS DA APLICAÇÃO DA SOLUÇÃO 50
5.4 - CUSTOS DA SOLUÇÃO 52
5.5 - AVALIAÇÃO GLOBAL DA SOLUÇÃO 54

CAPÍTULO 6 - CONCLUSÃO 55
6.1 - CONCLUSÕES 55
6.2 - SUGESTÕES PARA TRABALHOS FUTUROS 56

REFERÊNCIAS **57**

APÊNDICE A – SCRIPT *WEB SERVER* AUTOMÁTICO **60**

APÊNDICE B – *WEB SERVICE* RESTFUL: RESTWEBSERVICES **61**

APÊNDICE C – MAMBA_SERVER **64**

APÊNDICE D – MAMBA_RELAY **66**

APÊNDICE E – APLICATIVO *KINECTCLIENT* JAVA **68**

APÊNDICE F – APLICATIVO *KINECTCLIENT* PYTHON **74**

APÊNDICE G – APLICATIVO *ANDROIDCLIENT* **77**

LISTA DE FIGURAS

Figura 1.1 – Topologia do projeto 14
Figura 2.1 – Rede PLC 19
Figura 3.1 – *Mamba* PLC 24
Figura 3.2 – *Arduino Uno* 25
Figura 3.3 – *Cubieboard* 26
Figura 3.4 – *Shield Relay* para *Arduino* 27
Figura 3.5 – Dispositivo *Kinect* 32
Figura 4.1 – Solução Proposta 36
Figura 4.2 – Servidor Residencial (*hardware*) 37
Figura 4.3 – Implementação física do Servidor Residencial 38
Figura 4.4 – Servidor Residencial (*software*) 39
Figura 4.5 – Módulo de Automação (*hardware*) 40
Figura 4.6 – Implementação física do Módulo de Automação 41
Figura 4.7 – Módulo de Automação (*software*) 41
Figura 4.8 – Módulo Cliente *Kinect* (*hardware*) 42
Figura 4.9 – Implementação física do Módulo Cliente *Kinect* 43
Figura 4.10 – Módulo *Kinect* Java (*software*) 44
Figura 4.11 – Módulo *Kinect* Python (*software*) 45
Figura 4.12 – Módulo de *Smartphone* (*hardware*) 46
Figura 4.13 – Módulo *Smartphone* (*software*) 47
Figura 4.14 – *AndroidClient* (telas do aplicativo) 47
Figura 4.15 – *Tags* NFC utilizadas no projeto 48
Figura 5.1 – Acionamento de uma lâmpada através do Módulo de Automação 51
Figura 5.2 – Tratamento de imagem feito pelo aplicativo *KinectClient* Python 52

LISTA DE TABELAS

Tabela 5.1 – Custos Servidor Residencial 52
Tabela 5.2 – Custos Módulo de Automação 53
Tabela 5.3 – Custos Módulo *Smartphone* 53
Tabela 5.4 – Custos Módulo *Kinect* 53
Tabela 5.5 – Custos Solução Proposta 53

RESUMO

A diversidade atual das formas de comunicação, a necessidade de informações a todo instante e em qualquer lugar, exigem tecnologias de transmissão de dados que apresentem aceitável relação custo/benefício e uma satisfatória velocidade de comunicação, independentemente das localizações geográficas do transmissor e do receptor, respectivamente. Este universo se torna cada vez mais complexo quando abordamos o cenário de uma construção antiga, automação residencial, acessibilidade e custo/benefício. Deste modo, este trabalho tem como principal objetivo atender as necessidades deste cenário complexo provendo a melhoria de vida para deficientes físicos. Tal solução será implementada através da utilização da tecnologia PLC (*Power Line Communication*), tecnologia NFC (*Near Field Communication*), câmeras e sensores para o reconhecimento facial e dispositivos eletrônicos básicos utilizados comumente em Sistemas de Automação Residencial.

Palavras Chave: Sistemas de Automação Residencial, PLC, *Power Line Communication*, NFC, Reconhecimento Facial.

ABSTRACT

The current diversity of communications, the need for information at any moment and anywhere, require data transmission technologies which have acceptable cost/benefit ratio and a satisfactory communication speed, regardless of the geographical locations of the transmitter and receiver, respectively. This universe becomes increasingly complex when we approached the scenario of an old building, home automation, accessibility and cost/benefit. This work has as main objective to meet the needs of this complex scenario providing a better life for disabled people. Such a solution will be implemented through the use of PLC technology (Power Line Communication), NFC technology (Near Field Communication), cameras and sensors for face recognition and basic electronic devices commonly used in Home Automation Systems.

Keywords: Home Automation Systems, PLC, Power Line Communication, NFC, Facial Recognition.

CAPÍTULO 1 - INTRODUÇÃO

1.1 - APRESENTAÇÃO DO PROBLEMA

A acessibilidade tem sido colocada em primeiro lugar quando o assunto é construção civil, seja ela residencial ou comercial. Este fato levou as empresas a se comportarem de maneira diferente perante o mercado, investindo em tecnologias que pudessem permitir a acessibilidade em novos empreendimentos. Mas e o que fazer com as construções antigas?

1.2 - OBJETIVOS DO TRABALHO

Objetivo Geral – desenvolver e implementar solução que permite a automação residencial provendo acessibilidade aos portadores de deficiência física;

Objetivos Específicos:

- transmitir dados através da rede elétrica de qualquer edificação;
- reconhecer imagens capturadas de movimentos da cabeça e rosto do deficiente (tetraplégico);
- criar aplicativo para *smartphone Android* e *tags* NFC que permitirão a utilização por deficientes (paraplégico);
- prover serviço *web* que controla as requisições dos aplicativos clientes e envia os comandos à rede PLC;
- acionar dispositivos eletrônicos através da rede PLC reduzindo o esforço de deficientes permitindo a acessibilidade.

1.3 - JUSTIFICATIVA E IMPORTÂNCIA DO TRABALHO

Para solucionar este problema no cenário proposto (construções antigas, bom custo-benefício e curto tempo de instalação), propõe-se a utilização de duas tecnologias: Processamento e Controle por imagem e *Power Line Communication* - PLC (Comunicação através da Rede Elétrica).

O Controle por imagem permitirá a acessibilidade com pouco esforço já que todos os comandos da automação ocorrerão através de gestos e movimentos limitados do usuário.

A tecnologia PLC será utilizada como meio de transmissão dos comandos de automação dos usuários. O grande benefício de se utilizar esta tecnologia está no fato do projeto reaproveitar uma infraestrutura já instalada: a rede elétrica da construção, o que diminuiria consideravelmente o custo e tempo de implementação do projeto.

1.4 - ESCOPO DO TRABALHO

Transmissão de dados através da rede elétrica de qualquer edificação (rede PLC), com finalidade de automatizar o acionamento de eletrodomésticos e iluminação (Módulo de Automação).

O projeto possui três meios de interação com o usuário:

- o reconhecimento de movimentos da cabeça e dos olhos através de imagens capturadas do usuário (Módulo Cliente *Kinect*);
- a utilização de um aplicativo *mobile* em um *smartphone* (Módulo Cliente *Smartphone*);
- e a comunicação através da tecnologia NFC (Módulo Cliente *Smartphone*).

Sendo assim, a topologia do projeto pode ser dividida em três redes: NFC, *wireless* e PLC. A rede NFC é compartilhada pelas *tags* NFC e pelo Módulo Cliente *Smartphone*. Já a rede *wireless* é compartilhada pelos módulos que interagem com o usuário (Módulo Cliente *Smartphone* e Módulo Cliente *Kinect*) e o Servidor Residencial. Por fim, a rede PLC compartilha dados com os dois Módulos de Automação e com o Servidor Residencial. Esta topologia está ilustrada na Figura 1.1.

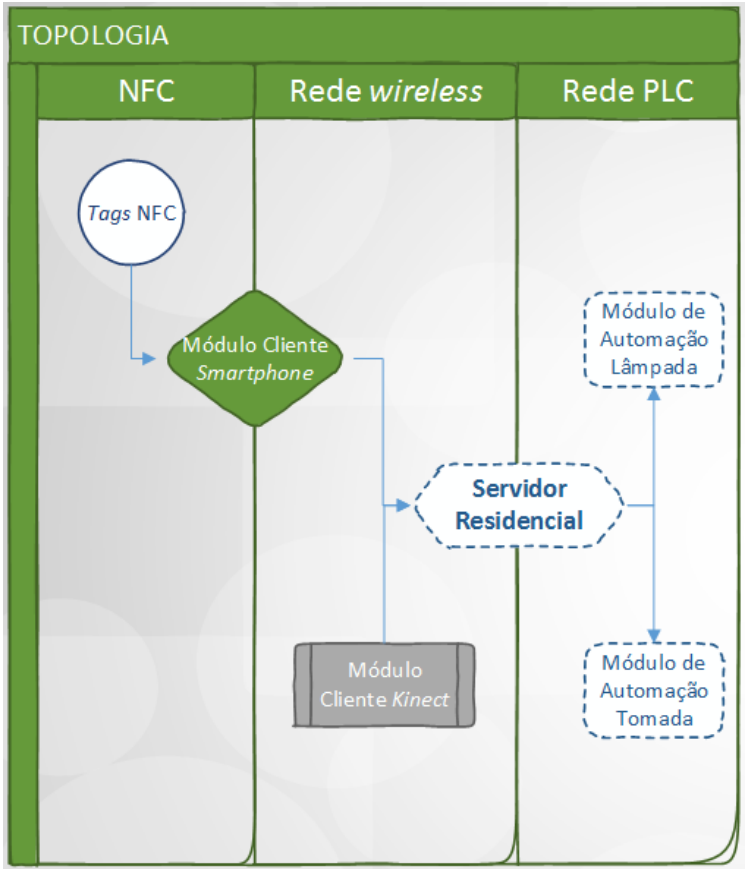

Figura 1.1 – Topologia do projeto
(Fonte: Autor)

1.5 - RESULTADOS ESPERADOS

Transmitir dados através da rede elétrica, considerando apenas um circuito na mesma fase, com taxa de transmissão aceitável e controle e detecção de erros de forma a evitar perdas em qualidade de serviço causadas por interferências decorrentes da utilização de dispositivos elétricos conectados ao circuito elétrico.

Identificação de gestos da cabeça e rosto do usuário através do tratamento de imagens, os quais serão utilizados como comandos para o acionamento de dispositivos eletrônicos. Além disto, analisar o rastreamento das imagens obtidas de modo a evitar envio de comandos desnecessários ao servidor.

Garantir generalização no acesso das aplicações clientes ao serviço *web* criado, de forma a abstrair o controle e acionamento de dispositivos eletrônicos presentes na rede PLC, evitando implementações específicas a cada aplicação cliente. Para exemplificar esta generalização os dispositivos poderão ser acionados através do reconhecimento de imagens, do aplicativo para *smartphone* e do reconhecimento de *tags* NFC.

Através deste conjunto de implementações e tecnologias, obter um Sistema de Automação Residencial de fácil e rápida instalação com baixo custo perante às soluções similares encontradas no mercado.

Por fim, auxiliar os deficientes nas atividades cotidianas em uma residência por meio do reconhecimento de movimentos com câmera que acionará dispositivos dentro da residência provendo automação.

1.6 - ESTRUTURA DO TRABALHO

O restante deste trabalho tem a organização descrita a seguir. O Capítulo 2 apresenta de forma detalhada o problema. O Capítulo 3 discorre acerca da tecnologia PLC e das tecnologias utilizadas para o tratamento de imagem, destacando os principais desafios a serem enfrentados para o emprego eficiente dessas tecnologias. O objetivo é fornecer fundamentos teóricos para o entendimento pleno das arguições posteriores que seguem ao longo do texto. O Capítulo 4 apresenta a solução proposta, ilustrando o modelo utilizado para a resolução do problema. O Capítulo 5 aborda a análise dos resultados obtidos através da aplicação da solução proposta. Por fim, as conclusões e as sugestões para trabalhos futuros constituem o Capítulo 6.

CAPÍTULO 2 - APRESENTAÇÃO DO PROBLEMA

A acessibilidade consiste na facilidade de acesso e de uso de ambientes, produtos e serviços por qualquer pessoa e em diferentes contextos. Envolve o *Design* Inclusivo, oferta de um leque variado de produtos e serviços que cubram as necessidades de diferentes populações (incluindo produtos e serviços de apoio), adaptação, meios alternativos de informação, comunicação, mobilidade e manipulação (GODINHO, 2010).

Esse conceito está presente na vida de pessoas com necessidade especial: pessoas com algum tipo de deficiência, idosos e outras pessoas com alguma incapacidade. Para dimensionar esta população, existem alguns dados importantes:

- A quantidade de pessoas com algum tipo de deficiência declarada vem aumentando e representa um bilhão de pessoas, 15% da população mundial de acordo com o primeiro relatório global em deficiência realizado pela WHO – *World Health Organization* em conjunto com o Banco Mundial (WORLD HEALTH ORGANIZATION; THE WORLD BANK, 2011).
- De acordo com o último Censo do Instituto Brasileiro de Geografia e Estatística (IBGE), 24% da população brasileira se definiu como portadora de algum tipo de deficiência. O percentual equivale a cerca de 45 milhões de pessoas (IBGE, 2010).

Esta crescente demanda tem colocado a acessibilidade em primeiro lugar quando o assunto é construção civil, seja ela residencial ou comercial. O que levou as empresas se comportarem de maneira diferente perante o mercado, investindo em tecnologias que pudessem permitir esta acessibilidade (SAAD, 2011) em novos empreendimentos.

Tecnologias voltadas para automação residencial, por exemplo, contribuem para o provimento de acessibilidade, pois proporcionam meios alternativos de comunicação, mobilidade e manipulação. Para este segmento da população (pessoas com necessidade especial), estes sistemas não são apenas uma questão de conveniência, são ferramentas necessárias as quais devolvem ao indivíduo sua independência.

Os principais sistemas de automação residencial são (AURESIDE, 2013):

- Segurança: alarmes, monitoramento, circuito fechado de TV, controle de acesso;
- Entretenimento: *home theater*, áudio e vídeo distribuídos;
- Controle de iluminação;
- *Home Office*: telefonia e redes;
- Ar condicionado e aquecimento;
- Portas e cortinas automáticas;

- Utilidades: bombas e limpeza de piscinas, controle de sauna, irrigação automática, aspiração central a vácuo;
- Infraestrutura: cabeamento dedicado, cabeamento estruturado, painéis, quadros de distribuição;
- Controladores e centrais de automação;
- *Softwares* de controle e integração;

Existem três níveis de interação em Sistemas de Automação residencial: Sistemas Autônomos, Integração de Sistemas, e a Residência Inteligente.

Nos Sistemas Autônomos podemos ligar ou desligar um subsistema ou um dispositivo específico de acordo com um ajuste pré-definido. Porém, neste esquema, cada dispositivo ou subsistema é tratado independentemente, sem que dois dispositivos tenham relação um com o outro.

A Integração de Sistemas é projetada para ter múltiplos subsistemas integrados a um único controlador. A limitação deste sistema é que cada subsistema deve ainda funcionar unicamente como o seu fabricante pretendia. Esta integração já permite uma ampla variedade de benefícios aos usuários e lhe garante a máxima eficiência no aproveitamento dos recursos utilizados.

Na Residência Inteligente o produto manufaturado pode ser personalizado para atender às necessidades do proprietário. O Integrador de Sistemas em conjunto com o proprietário delinearão instruções específicas para modificar o uso do produto. Assim, o sistema torna-se um gerenciador ao invés de apenas um controlador remoto. Os sistemas residenciais inteligentes dependem de comunicação de mão-dupla e *feedback* de status entre todos os subsistemas para um bom desempenho.

Os construtores, de forma a participar deste lucrativo novo mercado, precisam educar a si próprios e treinar ou contratar indivíduos para atuar como Integradores e Instaladores de sistemas residenciais (AURESIDE, 2013).

A comunicação entre o construtor e/ou Integrador e Instaladores com os proprietários é a chave para o sucesso. Um bom projeto de Automação Residencial resulta numa interface amigável para o usuário final, que poderá obter variados benefícios, dos quais destacamos:

- Economia;
- Segurança;
- Comodidade;
- Conforto;
- Entretenimento;

- Confiabilidade;
- Velocidade;
- Interatividade.

Estes sistemas são muito bem-vindos na vida das pessoas que estejam largamente dependentes de outros para realizar até as mais simples das atividades. Sistemas personalizáveis efetuam o controle ambiental. De comandos de voz simples ou chaves de toque (*touch pad*), estes sistemas controlam dispositivos elétricos, tais como luminárias, televisores e portas internas e externas adaptadas. Dispositivos hidráulicos como válvulas de descarga e de banheiras podem ser operadas com ajustes de água. Dispositivos de infravermelho tais como *Blu-ray player* ou televisor, podem ser controlados enquanto o sistema aumenta ou diminui as temperaturas do termostato.

Estes sistemas podem ser usados para operar funções normais de computador tais como o processador de palavras e navegar na Internet. Além disto, podem até mesmo atender ou discar telefones, o que permite ao usuário comunicar-se de casa ao seu trabalho, o que de outra maneira seria inviável (AURESIDE, 2013).

Neste contexto, as empresas de construção civil preparam a edificação para suportar uma tubulação dedicada ao cabeamento redes. Esta ação diminui a necessidade de obras para adequação de sistemas de automação residencial, pois, em sua grande maioria utilizam uma rede sobre IP para a integração dos sistemas, conforme especificado anteriormente.

Porém, em construções antigas, a conscientização se torna mais difícil, pois poucos condomínios e pessoas investem em obras ou tecnologias que proporcionem acessibilidade a qualquer cidadão. O motivo da falta de investimento está, principalmente, na complexidade de implementação destas tecnologias e custos associados a mesma.

Em construções antigas não existia a preocupação com o cabeamento de redes. Neste ambiente, projetos de automação possuem duas opções: realização de obras para implementação de um meio guiado ou utilização de um meio de comunicação sem fio. As duas opções elevam o custo do projeto e desmotivam as pessoas quanto ao investimento necessário.

Dentre as opções de meios guiado temos o cabo de par trançado, o cabo coaxial, a fibra ótica ou o cabo de cobre utilizado nas linhas de energia elétrica de uma residência. Esta última tecnologia é conhecia como *Power Line Communication* (PLC). As três primeiras opções não reaproveitam nenhuma característica da edificação, enquanto que a fiação elétrica está presente em todas edificações, seja ela antiga e ou recente. Isto já elimina a necessidade de obras para a sua implementação.

A dificuldade em usar a fiação elétrica domiciliar como uma rede é que ela foi projetada para distribuir energia elétrica. Essa tarefa é muito diferente de distribuir sinais de dados (HAYKIN, 2007), algo para a qual a fiação doméstica é pouco eficiente. Os sinais elétricos são enviados a 50-60 Hz e a fiação atenua os sinais de frequências muito mais altas (MHz) necessários para a comunicação de dados de alto nível (em geral frequências na ordem dos MHz).

As propriedades elétricas da fiação variam de uma casa para outra e mudam à medida que os aparelhos são ligados e desligados, fazendo com que os sinais de dados oscilem pela fiação. As correntes transitórias quando os aparelhos são ligados e desligados criam ruídos por uma larga faixa de frequências (LAMPE, 2011).

Sem o traçado cuidadoso dos pares trançados, a fiação elétrica atua como uma boa antena, captando sinais externos e emitindo sinais próprios. Este comportamento significa que, para atender aos requisitos da regulamentação, o sinal de dados precisa excluir frequências licenciadas, como as faixas de radioamador (TANENBAUM e WETHERALL, 2012).

Apesar dessas dificuldades, já é possível enviar dados a uma velocidade de 500 Mbps (EDUP, 2013) pela fiação elétrica doméstica (SCHWAGER, STADELMEIER e ZUMKELLER, 2005) usando esquemas de comunicação e protocolos de acesso ao meio (KUROSE, 2010) que resistem a frequências enfraquecidas e eclosões de erros (CHEN, 2008). Muitos produtos usam diversos padrões próprios para as redes de energia elétrica, de modo que os padrões internacionais estão em desenvolvimento ativo.

Além disto, devido ao potencial da tecnologia, diversos pesquisadores destinam suas pesquisas para a análise de performance, deste tipo de comunicação, em aplicações específicas como o tráfego dados de voz e vídeo (CARMONA e PELAES, 2012).

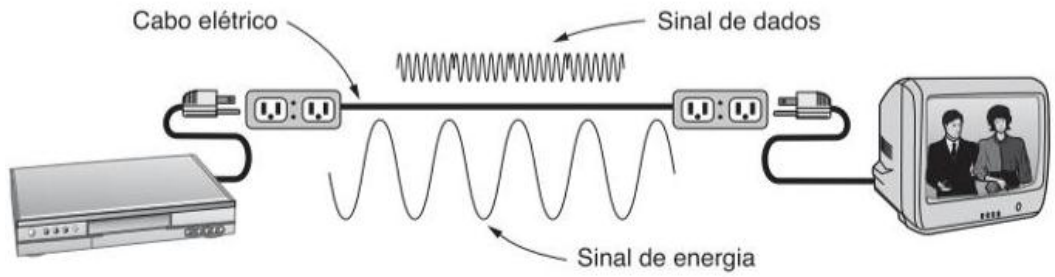

Figura 2.1 – Rede PLC
(Fonte: TANENBAUM e WETHERALL, 2012)

Além da utilização de Sistemas de Automação Residencial que utilizem a comunicação através da fiação elétrica para o provimento da acessibilidade, este trabalho busca formas alternativas de interação com este sistema, mais especificamente o sensoriamento humano por imagem.

Existem diversos trabalhos e aplicações de sucesso em Análise de Movimento Humano baseado em Visão, os quais classificam os trabalhos em taxonomia distintas. Os autores categorizam os trabalhos desta área em Detecção, Rastreamento e Entendimento de Comportamento Humano.

Detecção refere-se à abstração de um objeto em uma imagem (quadro ou *frame*). Rastreamento, no mapeamento desse objeto nos frames subsequentes à (primeira) detecção. Entendimento de comportamento humano, aborda a classificação de um comportamento característico a um objeto, que neste caso é um ser humano.

Os problemas que surgem na localização de uma pessoa por meio de câmeras são vários, tais como: posição relativa da pessoa na visão da câmera (frente, perfil, costas, sentado, deitado, inclinado), posição da câmera em relação ao ambiente, tamanho/roupa das pessoas, luminosidade do ambiente, fundo (*background*), perda temporária de localização da pessoa, ruído de sensores, velocidade do movimento do agente, objetos estáticos ou em movimento (como carro, animais, outras pessoas), movimento da câmera/pessoa (POPPE, 2010).

Há três principais aplicações na área de Análise de Movimento Humano sendo vigilância, controle e análise (MOESLUND e GRANUM, 2001). O trabalho aqui abordado refere-se à área de controle na qual uma particular detecção de movimento humano é utilizada para controle de algo – acionamento de um dispositivo elétrico integrado a um sistema de automação residencial.

Pode-se classificar a maior parte das propriedades humanas passíveis de mensuração em três: espaço-temporais, comportamentais e fisiológicas. A propriedade espaço-temporal é definida pelas características como presença, quantidade, localização, rastreamento e identidade de pessoas. Nas comportamentais, características como pose, ação, atividade, comportamento individual, comportamento em grupo, são consideradas. Por fim, as fisiológicas revelam-se por meio de informações quanto à frequência cardíaca, temperatura, peso, cor de pele, etc (CAVALCANTE, 2012).

Algumas propriedades fisiológicas do ser humano são sempre passíveis de observação, independente da ação que este possa realizar – nomeadas traços intrínsecos estáticos.

Algumas são movimentos dos batimentos cardíacos ou dos pulmões, diferença de temperatura do corpo com o ambiente, odor corporal, etc. (CAVALCANTE, 2012).

Outros traços intrínsecos e dinâmicos se revelam quando o ser humano está em atividade. Alguns exemplos: se uma pessoa está em movimento, então pode-se detectar o movimento dos respectivos membros; se caminhando, a percepção da pressão, vibração, ou som produzido pelos pés ao se caminhar; se falando, a detecção de voz (TEIXEIRA, DUBLON e SAVVIDES, 2010).

Quanto aos meios para abstração desses dados, estes variam largamente dependendo dos tipos de informações produzidas e dos problemas tratados, tais como: sensores infravermelhos passivos, pisos sensíveis à pressão, detecção de campos elétricos, sensores de vibração, sensores laser/ultrassom/radio, câmeras, sensores inerciais, microfones, etc (TEIXEIRA, DUBLON e SAVVIDES, 2010).

Igualmente, o sensoriamento humano apresenta diversos desafios a serem tratados, tais como (TEIXEIRA, DUBLON e SAVVIDES, 2010): ruído de sensores; variações ambientais não esperadas (chuva, neblina, variações de temperatura) e similaridades com o *background*; movimento de objetos como carros, cadeiras, outras pessoas; variabilidade na aparência e imprevisibilidade; fraude ativa.

O sistema ou o meio pelo qual se obtém a localização de um ser humano provém de sensores e do respectivo processamento dos dados destes (MOESLUND e GRANUM, 2001). Pode-se classificar os sensores como instrumentados e não-instrumentados referentes à utilização ou não de sensores no corpo do usuário, respectivamente (TEIXEIRA, DUBLON e SAVVIDES, 2010).

Abordagens não-instrumentadas manifestam-se por meio de sensores binários, sensores de vibração, *Ranger-Finders*, *Doppler-shift*, câmeras (e outros *imageadores*), etc. (TEIXEIRA, DUBLON e SAVVIDES, 2010).

Sensores binários, basicamente, retornam um lógico 1 se alguma pessoa é detectada num ambiente, e 0 caso contrário. Destes, há PIRs (*Passive infrared*), sensores de contato, sensores de campo elétrico, etc. Sensores de vibração medem os sinais produzidos pelos passos de uma pessoa – em contextos maiores, estes podem ser encontrados em sensores geossísmicos.

Varredura de *Range-Finders* transmitem um sinal e verificam o eco desse sinal com relação ao tempo ou energia. Câmeras e outros *imageadores* permitem a abstração de uma quantidade considerável de informações concernentes a uma pessoa (como movimento, posição, comportamento, aparência, tipos de objetos presentes no ambiente, etc) ao custo de

alta dimensionalidade comparada a outras modalidades de sensores (TEIXEIRA, DUBLON e SAVVIDES, 2010).

Em abordagens instrumentadas a detecção do usuário ocorre através de sensores no corpo de uma pessoa. Esta modalidade pode-se traduzir por meio de dispositivos operando em conjunto com uma rede externa, por exemplo, o Sistema de Posicionamento Global (GPS) utilizado através de telefones móveis com a pessoa.

Sinais *WiFi* podem ser utilizados para estimar a localização de assinaturas de força (*strength signature*) de sinal em cômodos distintos. Sinais de comunicação de telefonia móvel também podem ser trabalhados para estimação de localização de pessoa. Sensores inerciais no corpo de uma pessoa estimam a trajetória de movimento do corpo com relação a sua velocidade e aceleração por meio de acelerômetros e giroscópios.

Campos magnéticos podem ser trabalhados por meio da emissão artificial de correntes DC em três eixos. Receptores são posicionados em algumas regiões do corpo humano e medem os campos magnéticos caracterizados nos três eixos.

Potenciômetros eletromecânicos podem ser utilizados diretamente na estimação da orientação de algumas junções do corpo humano ao serem posicionadas no corpo de uma pessoa. Também, a intensidade de pulsos acústicos emitidas por transmissores ultrassônicos podem ser utilizadas para estimação de posição do corpo de uma pessoa por meio de microfones posicionados em locais específicos do usuário.

Os sensores, de modo geral, podem ser caracterizados como passivos ou ativos. Sensores ativos são dispositivos artificiais presentes no ambiente que auxiliam no problema do rastreamento (MOESLUND e GRANUM, 2001). Estes podem estar acoplados no usuário e/ou ambiente, como sensores de detecção de movimento, variação de temperatura, tátil, etc.

Em uma abordagem com sensores passivos utilizam-se fontes naturais como a própria luz do ambiente (natural ou elétrica), ondas infravermelhas de calor humano, etc. Desta forma, permite uma interação mais próxima ao contexto de comunicação de humanos, livre de auxiliadores.

A presença de alguns auxiliadores como marcos de papéis, bolas, vestimentas de cor homogênea, não deixam de serem sensores passivos, visto que os mesmos não emitem e nem recebem (processam) sinal artificial algum, diferente da abordagem ativa (MOESLUND e GRANUM, 2001).

Desta forma, este trabalho busca simplificar o processo de implementação da automação residencial. Aplicando tecnologias voltadas para pessoas com necessidade especial com um baixo custo, como será exposto nos próximos capítulos.

CAPÍTULO 3 - REFERENCIAL TEÓRICO

A diversidade atual das formas de comunicação e a necessidade de informações a todo instante e em qualquer lugar exigem tecnologias de transmissão de dados que apresentem aceitável relação custo/benefício e uma satisfatória velocidade de comunicação (ZATTAR, CORRÊA e CARRIJO, 2012), independentemente das localizações geográficas do transmissor e do receptor, respectivamente.

3.1 - INFRAESTRUTURA E *HARDWARE*

3.1.1 - *Power Line Communication* (PLC)

Uma tecnologia que apresenta essas características é a comunicação de dados por meio das linhas de transmissão da rede elétrica (SCHWAGER, STADELMEIER e ZUMKELLER, 2005). Conhecida como *Power Line Communication* (PLC), esta surge como opção principalmente em regiões onde o custo/benefício da rede *wireless* ou a cabo não é atrativo, pois a mesma se beneficia diretamente da infraestrutura da própria rede elétrica (KWON, SEO, *et al.*, 2009), permitindo assim a sua implementação em qualquer edificação que utilize a rede elétrica.

A configuração básica de uma rede de comunicação em que se emprega a tecnologia PLC, doravante referenciada simplesmente como rede PLC, consiste em um modem *master* instalado próximo ao transformador de baixa tensão. Comparativamente a uma residência, tal localização seria a entrada de energia elétrica da rua para a residência. O modem *master* tem a função de gerenciar e distribuir a transmissão dos dados da rede PLC para modems internos à residência (ZATTAR, CORRÊA e CARRIJO, 2012).

Dentre as diversas opções de modem PLC do mercado foi escolhido para utilização neste projeto o dispositivo *Mamba PLC*. Este dispositivo apresenta as seguintes especificações técnicas:

- Tensões e frequências de funcionamento: 110/240V, 50/60Hz;
- Comunicação através de interface SPI;
- Taxa de transmissão: 2048 bps;
- Formato de *shield* para *Arduino*.

3.1.2 - Modem PLC

O modem *Mamba PLC* apresenta duas vantagens perante aos demais: menor custo e fácil integração com o microcontrolador a ser utilizado neste trabalho, o *Arduino*. A fácil

integração ocorre porque este dispositivo possui formato de um *shield* (formato empilhável na placa microcontroladora *Arduino*), conforme mostrado na Figura 3.1. Porém, apresenta taxa de transmissão baixa, mas o suficiente ou aceitável para aplicação na automação, visto que, a rede PLC possui a finalidade exclusiva de automação, o que não exige altas taxa de transmissão.

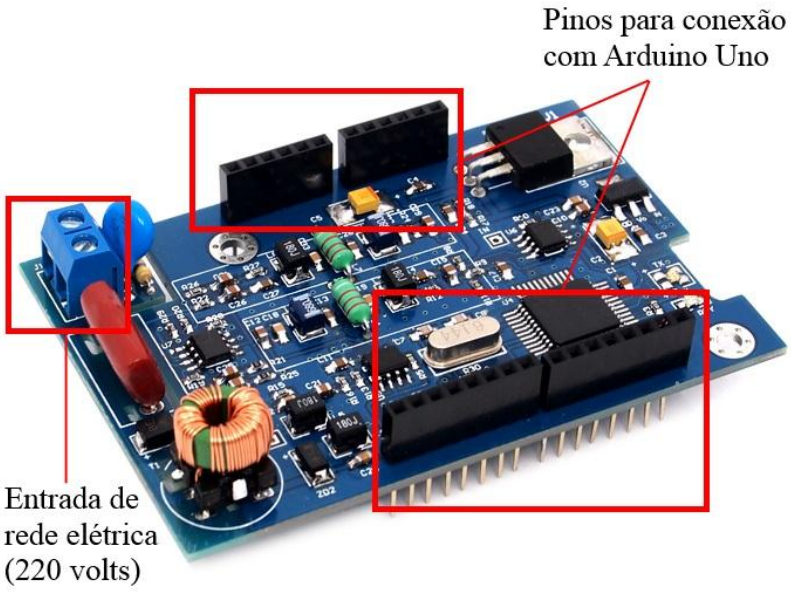

Figura 3.1 – *Mamba* PLC
(Fonte: www.seeedstudio.com/depot/mamba-narrow-band-power-line-communication-shield-p-1296.html, adaptada)

Para integração dos demais dispositivos e comunicação através da rede elétrica, o modem PLC escolhido (*Mamba PLC*) necessita de um microcontrolador, um microprocessador que pode ser programado para funções específicas, em contraste com outros microprocessadores de propósito geral (como os utilizados nos PCs). Eles são embarcados no interior de algum outro dispositivo (geralmente um produto comercializado) para que possam controlar as funções ou ações do produto.

3.1.3 - Microcontrolador

Neste trabalho foi escolhido a placa de microcontrolador *Arduino Uno*. Esta placa utiliza o microcontrolador *ATmega328*. Possui 14 entradas/saídas digitais (das quais 6 podem ser usados como saídas PWM), 6 entradas analógicas, uma conexão USB, um conector de alimentação, um cabeçalho ICSP, e um botão de reset (ARDUINO, 2013). Segue a descrição detalhada:

- Microcontrolador: ATmega328;
- Tensão de operação: 5V;

- Tensão de entrada (recomendada): 7-12V;
- Tensão de entrada (limites): 6-20V;
- Pinos de entrada/saída digital: 14;
- Pinos de entrada analógico: 6;
- Corrente contínua por pino entrada/saída: 40 mA;
- Corrente contínua para o pino de 3.3V: 50 mA;
- Memoria *Flash*: 32 KB (ATmega328);
- SRAM: 2 KB (ATmega328);
- EEPROM: 1 KB (ATmega328);
- Velocidade do *Clock*: 16 MHz.

A tensão de operação utilizada é de 12 volts (dentro dos limites permitidos 6-20V) e dos 14 pinos de entrada/saída foram utilizados os pinos com numeração de 0 a 7, conforme mostrado na Figura 3.2 A placa microcontroladora *Arduino Uno*, possui características essenciais ao projeto: *open source* e baixo custo. Estas duas características correspondem a expectativa de uma Solução de automação residencial de baixo custo e pela caracterização de *open source* permite uma grande customização. Outra qualidade para a escolha deste dispositivo foi a alta compatibilidade aos demais que serão utilizados no projeto como: o *Mamba PLC* e o *Shield Relay*.

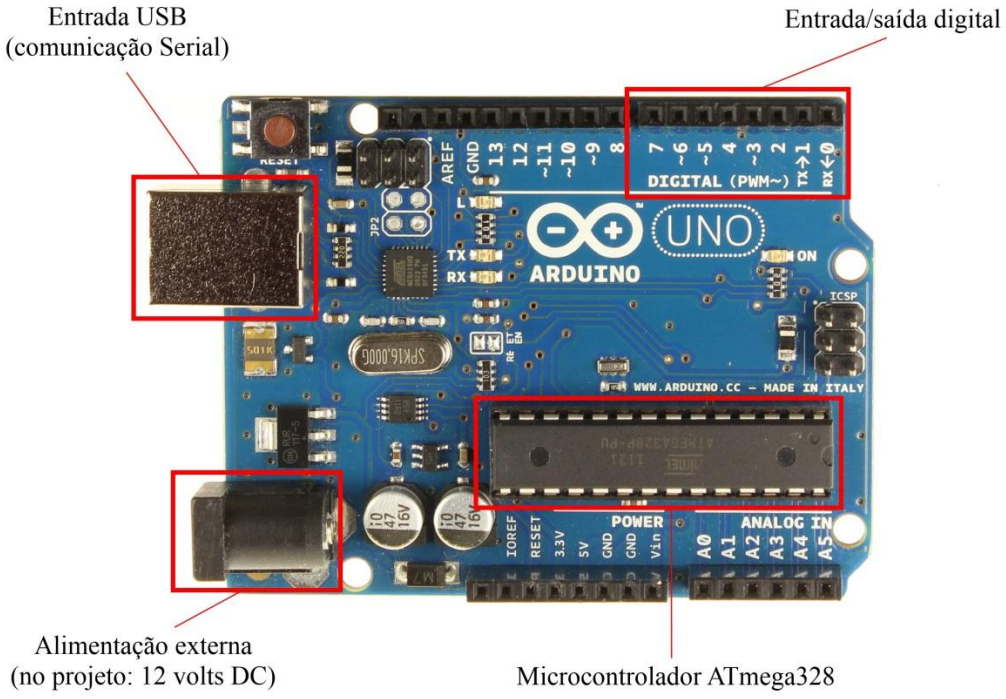

Figura 3.2 – *Arduino Uno*
(Fonte: www.arduino.cc/en/Main/ArduinoBoardUno, adaptada)

3.1.4 - Mini-PC

Para o processamento dos dados necessário no projeto é necessário um computador, desta forma foi escolhido a abordagem de um mini-PC. Este dispositivo é um computador com tamanho bem reduzido, baixo consumo energético e baixo custo. Todo *hardware* é integrado em uma única placa. O objetivo inicial deste tipo de dispositivo era estimular o ensino de computação básica em escolas. Porém, atualmente, vem sendo utilizado em diversas aplicações que se beneficiem de suas características.

O *Cubieboard*, mostrado na Figura 3.3, é um exemplo de mini-PC desenvolvido por uma empresa Chinesa, a *Cubietech Limited* (CUBIEBOARD, 2013). Este dispositivo possui três versões: *Cubieboard*, *Cubieboard 2* e *Cubietruck*. O modelo utilizado neste trabalho será o *Cubieboard* 2 que apresenta as seguintes características:

- Escalonamento dinâmico do *clock* da CPU: 60MHz~1GHz;
- Processador: *dual-core*;
- Memória RAM: 1GB DDR3;
- Armazenamento: 4GB *Nand Flash*;
- 2D/3D GPU;
- Processador de vídeo: 2160p HD;
- I2C, 96 GPIOs, TWM, 2.5' SATA, *Ethernet*.

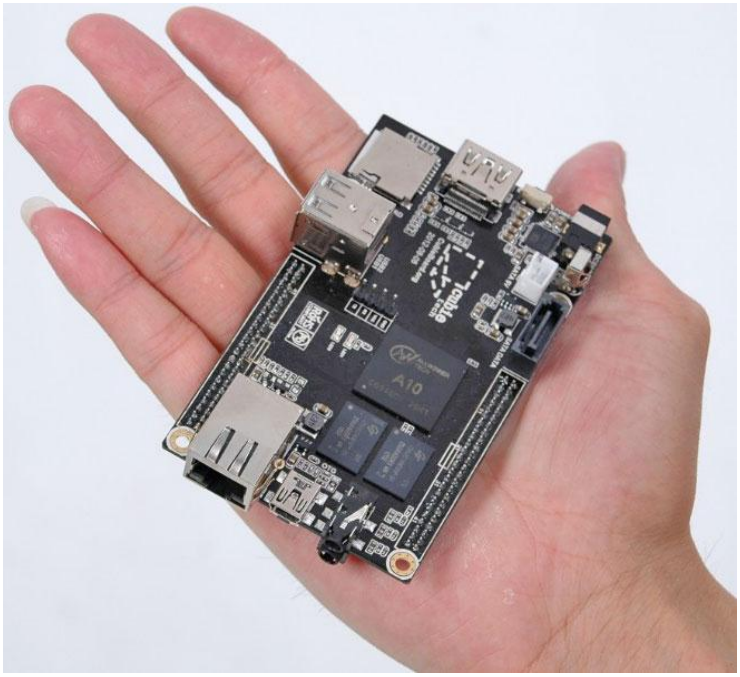

Figura 3.3 – *Cubieboard*
(Fonte: www.j1nx.nl/cubieboard-about-to-get-released)

Um mini-PC como o *Cubieboard* se adequa perfeitamente à proposta do projeto devido a vários motivos, dos quais destacam-se: *open source*, baixo custo, tamanho compacto (cabendo inclusive na palma da mão, conforme observado na Figura 3.3), baixo consumo de energia e boa capacidade de processamento.

O baixo custo e a caracterização *open source* são inerentes a todos os componentes escolhidos para o projeto, pois um dos objetivos específicos é alcançar o baixo custo. A alta customização necessária no sistema necessita da abertura do código fonte de vários componentes.

As características restantes, como o tamanho compacto, baixo consumo de energia e a capacidade de processamento caracterizam o dispositivo como ferramenta ideal a ser acoplada em uma cadeira de rodas, pois apresenta alta mobilidade, autonomia energética e capacidade para o processamento de imagens.

3.1.5 - Relé

Para o acionamento dos dispositivos eletrônicos integrados ao Sistema de Automação Residencial, foi utilizado o dispositivo relé (*relay* em inglês), um interruptor eletromecânico. A movimentação física deste interruptor ocorre quando a corrente elétrica percorre as espiras da bobina do relé, criando assim um campo magnético que por sua vez atrai a alavanca responsável pela mudança do estado dos contatos.

Neste projeto será utilizado uma placa de fácil integração (formato que permite o empilhamento) com o *Arduino Uno*, o *Shield Relay*. Esta placa que possui 4 relés com 4 pinos digitais para controle, porém só será utilizado o relé 4, conforme mostrado na Figura 3.4.

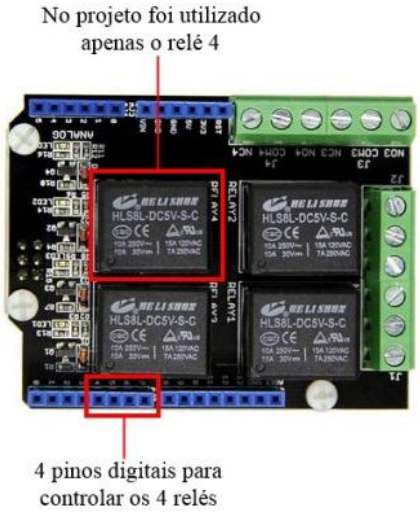

Figura 3.4 – *Shield Relay* para *Arduino*
(Fonte: www.seeedstudio.com/wiki/Relay_Shield_V2.0, adaptada)

3.1.6 - NFC (*Near Field Communication*)

Near Field Communication (NFC) é uma tecnologia sem fio de curto alcance, geralmente exigindo uma distância de 4 cm ou menos para iniciar uma conexão. A tecnologia NFC permite que você compartilhe pequenas cargas de dados entre uma *tag* (etiqueta) NFC e um dispositivo com esta tecnologia (ANDROID, 2013).

As *tags* podem variar em complexidade: *tags* simples permitem ler e escrever semânticas apenas uma vez, sendo assim, após a gravação, a *tag* torna-se somente leitura. *Tags* mais complexas podem oferecer operações matemáticas, e possuem um *hardware* criptográfico para autenticar o acesso a um setor. As *tags* mais sofisticados contêm ambientes operacionais, permitindo interações complexas com o código executado na *tag*.

A transmissão pode ocorrer de dois modos:

- Passivo: neste modo, apenas um dos dispositivos gera o sinal de radiofrequência da conexão. O segundo é apenas alimentado por este. Com isso, é possível colocar etiquetas NFC em itens que não recebem alimentação elétrica direta, como cartões, embalagens e cartazes;

- Ativo: no modo ativo, ambos os dispositivos geram o sinal de rádio. É o modo que é utilizado, por exemplo, em um sistema de pagamento envolvendo um *smartphone* e um receptor no caixa de uma loja.

Os dados armazenados na *tag* também podem ser escritos em uma variedade de formatos, mas muitas das APIs para *Android* são baseadas em torno de um padrão NFC *Forum* chamado NDEF (NFC *Data Exchange Format*) (ANDROID, 2013).

Dispositivos *Android* que possuem a tecnologia NFC suportam simultaneamente três modos de operação (ANDROID, 2013):

- modo de leitura/escrita, permitindo que o dispositivo leia e/ou escreva em uma *tag* NFC;
- modo ponto-a-ponto, permitindo que o dispositivo NFC transfira dados com outros pares NFC, este modo de operação, em dispositivos *Android*, é usado através da opção *Android Beam*.
- e o modo de emulação de cartão. Neste modo, o dispositivo NFC pode se passar por um cartão inteligente, de forma que o aparelho leitor não consiga distinguir um do outro.

Devido a simplicidade de utilização, neste trabalho foi abordado a transmissão no modo passivo e o modo de operação de leitura/escrita. Neste modo foi possível gravar os dados na *tag* e posteriormente recuperá-los através da aproximação de um *smartphone* com a *tag.*

3.2 - ARQUITETURA E *SOFTWARE*

3.2.1 - *Web Service*

Um serviço *web* (*Web Service*) fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores *web*. Os clientes acessam operações de um serviço *web* por meio de requisições e respostas formatadas em XML e, normalmente, transmitidas por HTTP.

Uma interface de serviço *web* consiste em um conjunto de operações que podem ser usadas por um cliente na Internet. As operações de um serviço *web* podem ser fornecidas por uma variedade de recursos diferentes, por exemplo, programas, objetos ou banco de dados. Um serviço *Web* pode ser gerenciado por um servidor *web*, junto a páginas *web*, ou pode ser um serviço totalmente separado (COULOURIS, DOLLIMORE, *et al.*, 2013).

Uma alternativa é a estratégia *Representational State Transfer* (REST). REST é uma estratégia com um estilo de operação muito restrito, no qual clientes usam URLs e as operações HTTP (*GET*, *PUT*, *DELETE* e *POST*) para manipular recursos representados em XML (FIELDING, 2000).

A ênfase está na manipulação de recursos de dados, em vez de interfaces. Quando um novo recurso é criado, ele recebe um novo URL por meio do qual pode ser acessado ou atualizado. Os clientes recebem o estado inteiro de um recurso, em vez de chamar uma operação para fornecer alguma parte dele (COULOURIS, DOLLIMORE, *et al.*, 2013). Os serviços *web* REST seguem a arquitetura orientada a recursos (*Resource-Oriented Architecture* - ROA) (RICHARDSON e RUBY, 2007).

3.2.2 - Linguagem de programação

3.2.2.1 - Linguagem Java

Um código destinado à execução em um computador não é necessariamente adequado para outro computador, pois, normalmente, os programas executáveis são específicos a um conjunto de instruções e a um sistema operacional.

A estratégia de máquina virtual oferece uma maneira de tornar o código executável em uma variedade de computadores hospedeiros: o compilador de uma linguagem em particular gera código para uma máquina virtual, em vez de código para um processador e sistema operacional específicos.

Por exemplo, o compilador Java produz código para uma máquina virtual Java (JVM, *Java Virtual Machine*), a qual executa por meio de interpretação. A máquina virtual Java

precisa ser implementada uma vez para cada tipo de computador a fim de que os programas em Java sejam executados (COULOURIS, DOLLIMORE, *et al.*, 2013).

A linguagem de programação Java é uma linguagem de alto nível que possui as seguintes características (ZAKHOUR, KANNAN e GALLARDO, 2013):

- Simples;
- Orientado a objeto;
- Distribuído;
- *Multithreaded;*
- Dinâmico;
- *Portable;*
- Alta performance;
- Robusto;
- Seguro.

3.2.2.2 - Linguagem Python

Python é uma linguagem de uso geral que combina funcionalidades de uma linguagem estruturada e os paradigmas da orientação a objeto. Esta linguagem é comumente definida como uma linguagem de script orientada a objeto (LUTZ, 2013).

Dentre os fatores que levam a utilização da linguagem Python, estão (LUTZ, 2013):

- a qualidade de *software*: o foco de Python em legibilidade, coerência e qualidade de *software* em geral o diferencia de outras linguagens baseadas em script. O código Python é projetado para ser legível, e, portanto, reutilizável e de fácil manutenção. A uniformidade do código Python torna-o fácil de entender, para quem não programa em Python. Além disso, Python utiliza mecanismos de reutilização de *software*, tais como a programação orientada a objetos e a programação modular;
- a produtividade do desenvolvedor: Python aumenta a produtividade do desenvolvedor perante as linguagens tipificadas como C, C++ e Java. O código Python é tipicamente um terço a um quinto do tamanho de um código C++ ou Java equivalente. Isso resulta na diminuição da quantidade de linhas de código, menos código para depurar, e menos código para manter. Programas em Python podem ser executados imediatamente, sem a necessidade de etapas de compilação exigidos por algumas linguagens;

- a portabilidade: a maioria dos programas Python são executados sem a necessidade de alteração nas principais plataformas de computador. Para portar um código Python entre Linux e *Windows*, por exemplo, é necessário apenas copiar o código de uma máquina para a outra. Além disso, Python oferece várias opções para a codificação de interfaces gráficas para usuário, programas para acesso a dados e sistemas *web*;
- as bibliotecas de apoio: Python vem com uma grande biblioteca de funcionalidades padrão, conhecido como *standard library*. Além desta biblioteca, Python pode ser estendido com bibliotecas próprias e também com a vasta quantidade de bibliotecas de terceiros existentes na comunidade. Dentre elas destacam-se: bibliotecas para construção de sites, para programação numérica, acesso à porta serial, desenvolvimento de jogos. A extensão *NumPy*, por exemplo, tem sido descrita como uma biblioteca *freeware* poderosa equivalente ao sistema de programação numérica *Matlab*;
- e a integração de componentes: scripts Python podem facilmente se comunicar com outros aplicativos, usando uma variedade de mecanismos de integração. Estas integrações permitem que o código Python seja utilizado como uma ferramenta de extensão. O código Python pode invocar bibliotecas em C e C++, pode ser chamado a partir de programas em C e C++, pode ser integrado com Java e .NET, pode interagir com os dispositivos através de portas seriais e podem interagir em redes que implementem SOAP, REST, XML-RPC e CORBA.

Neste projeto, a utilização da linguagem Python foi diretamente influenciada por três destes fatores: a portabilidade, as bibliotecas de apoio e a integração de componentes. A portabilidade devido a necessidade de rodar o código em *hardware* com arquitetura ARM. As bibliotecas de apoio foram utilizadas para facilitar o tratamento de imagens, dentre elas destaca-se a biblioteca *OpenCV* (OPENCV, 2013), biblioteca construída a partir de algoritmos voltados para a visão computacional. Já a integração de componentes da linguagem Python permitiu o envio das informação através do consumo do *Web Service* REST.

3.2.3 - Sistema Operacional

Linux é um sistema operacional *open source* que permite alta customização e integração dos sistemas utilizados neste trabalho. Além de não possui custo algum, sendo disponibilizado pela licença GPL (versão 2).

Outro fator determinante para sua escolha foi existência de compilações exclusivas para processadores de arquitetura ARM, o qual está presente no mini-PC, anteriormente citado, que será utilizado no projeto.

3.2.4 - *Microsoft Kinect*

O dispositivo *Kinect* foi lançado originalmente para o vídeo game *Xbox* pela *Microsoft Corporation* em novembro de 2010. Este mecanismo permite que o controle do vídeo game seja substituído pelo reconhecimento do movimento do corpo do usuário para interagir com alguns jogos. Em outras palavras, o usuário é o controle.

Ele é constituído de uma câmera RBG, um sensor de profundidade (*depth sensor*) composto de um emissor infravermelho e de um sensor monocromático CMOS, um conjunto de canais de microfones e um tipo de alavanca localizado abaixo do conjunto desses sensores que permite o dispositivo se acomodar ao campo de visão com relação à disponibilidade do usuário na frente do mesmo. Na Figura 3.5 é mostrado o dispositivo.

Figura 3.5 – Dispositivo *Kinect*
(Fonte: Autor)

Além da utilização na área de jogos, o advento do *Microsoft Kinect* permitiu a utilização desse dispositivo em meios computacionais para variadas propostas, inclusive em manipulação de robôs.

Logo após o lançamento deste, houveram algumas tentativas de modificações na operação do *Kinect* que objetivavam sua utilização por meio de plataformas *desktops*.

O resultado dessas intervenções foram amostras de alguns pequenos aplicativos, que permitiam a utilização do dispositivo em um Computador Pessoal (PC). Não muito tempo depois, versões preliminares de bibliotecas foram disponibilizadas para o público por meio de alguns meios, tais como a empresa *PrimeSense* (fabricante de parte do *hardware* do *Kinect*) e pela própria *Microsoft*.

Cerca de um ano e três meses após o lançamento do dispositivo, a *Microsoft* lançou uma versão estável de sua biblioteca chamada *Kinect for Windows SDK* (MICROSOFT, 2013), juntamente com um *hardware Kinect* para utilização apenas em PCs *Windows*.

Além da versão disponibilizada pela *Microsoft*, outras bibliotecas disponíveis para utilização em multi-plataformas são *OpenNI* (*PrimeSense*) (OPENNI, 2013) e *OpenKinect* (OPENKINECT, 2013). O *Framework OpenNI* e o *Framework OpenKinect* são utilizados neste projeto e serão descritos nas próximas Seções.

Neste trabalho o *Microsoft Kinect* será utilizado para o reconhecimento de movimentos de uma pessoa com necessidade especial que esteja se beneficiando do sistema de Automação Residencial. Sua escolha está relacionada à facilidade de desenvolvimento que suas bibliotecas proporcionam e também à quantidade de sensores, além da câmera, existentes em seu *hardware*.

3.2.5 - *Framework OpenNI*

O *Framework OpenNI* (OPENNI, 2013) permite a utilização do *hardware Kinect* em PCs para uma variedade de aplicações. Desde que uma pessoa esteja à frente do dispositivo, é possível compreender a quantidade de pessoas na frente do mesmo, as configurações espaciais e angulares com relação às junções do corpo humano, o reconhecimento de alguns gestos de mão, a utilização de dados brutos em cada um dos sensores que compõem o dispositivo (a câmera RGB, o sensor de profundidade e os canais de som), o reconhecimento de movimento da mão para controle de alguma interface, etc.

Deste *framework*, obtêm-se vantagens de 2 dados principais, o primeiro é a máscara de esqueleto, onde as configurações espaciais e angulares citadas são fornecidas pela biblioteca por meio de uma máscara de esqueleto.

Em poucas palavras, essa máscara é uma representação em forma de esqueleto onde os terminais de cada segmento desse modelo são referenciados como algumas junções do corpo do usuário. Cada uma dessas junções tem 3 propriedades (DUNN e PARBERRY, 2011), sendo elas:

- Posição em 3D: A posição de uma junção em 3D é representada por 3 eixos de coordenadas com origem no centro do dispositivo *Kinect*, onde duas coordenadas (aqui representadas por x e y) representam em milímetros a distância de uma junção com relação a um eixo horizontal (x) e vertical (y) imaginário com referência a esse ponto origem. A outra coordenada (representada por z) representa o quão distante (em milímetros) a junção se encontra afastada a partir do centro do dispositivo;
- Matriz de rotação: Uma matriz de rotação de 3D, cujos valores representam a orientação da base do espaço vetorial de um objeto (no caso a junção) relativo ao espaço de coordenadas do *Kinect*;
- Fator de confiança: O valor de confiança pode ser encontrado em apenas duas formas: 0% (o rastreamento da junção foi perdido ou ocluído por algum outro membro) ou 100% (quando a estimação parece estar funcionando corretamente).

O segundo é o reconhecimento de alguns gestos pré-definidos pelo *OpenNI*. O *framework* possui alguns poucos gestos pré-definidos de mão passíveis de utilização. Neste trabalho são utilizados apenas dois gestos para permitirem a ativação de alguns modos de comportamentos definidos.

Os dados utilizados são produzidos apenas pelo sensor de profundidade do dispositivo, o qual transmite dados a uma faixa de 30 quadros por segundo. Então, este é o único sensor do *Kinect* utilizado no projeto.

3.2.6 - *Framework libfreenect* (*OpenKinect*)

OpenKinect é uma comunidade de pessoas interessadas em fazer uso do *hardware Microsoft Kinect* através de PCs e outros dispositivos. Com foco em bibliotecas *open source* que permitirá que o *Kinect* seja utilizado com o *Windows*, Linux e Mac.

A comunidade *OpenKinect* consiste em mais de 2000 membros que contribuem seu tempo e código para o projeto. O foco principal é, atualmente, o *framework libfreenect*. O código disponibilizado pela comunidade *OpenKinect* é disponibilizado sob uma licença Apache20 ou GPL2 opcional (OPENKINECT, 2013).

O *Framework libfreenect* é distribuído em várias linguagens de programação o que permite ao desenvolvedor escolher a de sua preferência. As distribuições em cada linguagem são chamadas *wrappers* (invólucro). Neste projeto, conforme abordado na Seção 3.2.2.2, foi utilizado a linguagem Python, desta forma optou-se pelo *Wrapper* Python do *Framework libfreenect.*

O *wrapper* para Python fornece interfaces assíncronas (por exemplo, usando *callbacks*) e síncronas (por exemplo, chamadas simples de função) com *libfreenect*. A interface foi projetada de forma similar ao *wrapper* para linguagem C (OPENKINECT, 2013).

Todos os conceitos e tecnologias abordados no decorrer deste capítulo servirão como base para toda implementação do projeto como poderá ser vistos nos próximos capítulos. Todos eles foram abordados por possuírem características que permitirão o atingimento dos resultados esperados ao término da implementação, são elas:

- baixo custo;
- *Open source;*
- e eficiência energética.

CAPÍTULO 4 - SOLUÇÃO PROPOSTA

4.1 - APRESENTAÇÃO GERAL DO SOLUÇÃO PROPOSTA

Para solucionar este problema no Cenário proposto (acessibilidade, construções antigas, bom custo-benefício e curto tempo de instalação), a Solução Proposta se baseia na utilização de duas tecnologias: Sensoriamento Humano e *Power Line Communication* - PLC (Comunicação através da Rede Elétrica). A Figura 4.1 ilustra a Solução Proposta mostrando as interfaces utilizadas como cliente (*smartphone* e *Kinect*), o dispositivo utilizado como servidor (*Cubieboard*) e a utilização de redes *wireless* e PLC.

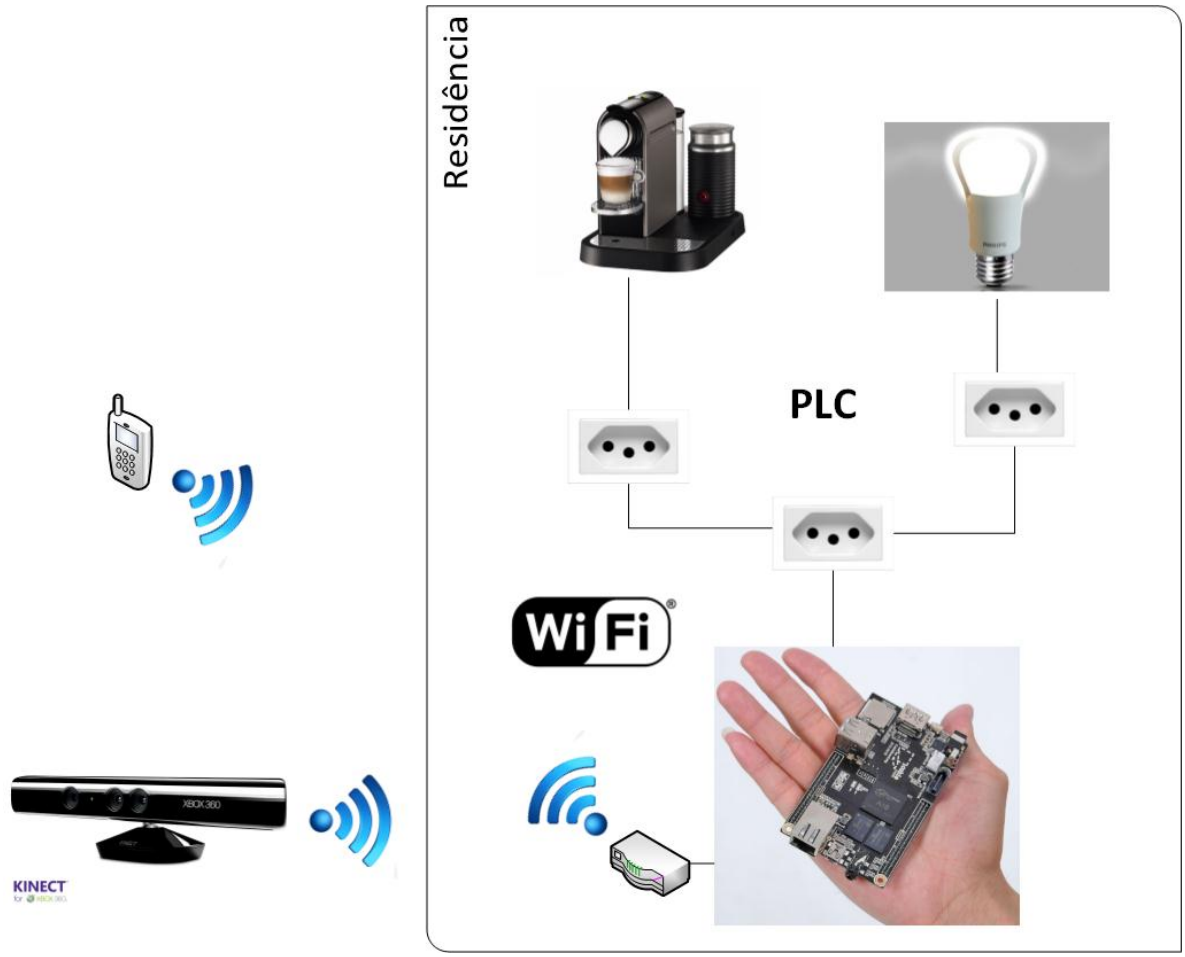

Figura 4.1 – Solução Proposta
(Fonte: Autor)

4.2 - DESCRIÇÃO DAS ETAPAS DA SOLUÇÃO PROPOSTA

O Sensoriamento Humano permitirá a acessibilidade com pouco esforço já que todos os comandos da automação ocorrerão através de gestos e movimentos do usuário, cujas imagens são capturadas e utilizadas para o controle de dispositivos eletrônicos e iluminação.

A tecnologia PLC será utilizada como meio de transmissão dos comandos de automação dos usuários. O grande benefício de se utilizar esta tecnologia está no fato do projeto reaproveitar uma infraestrutura já instalada: a rede elétrica da construção, o que diminui consideravelmente o custo e tempo de implementação do projeto.

4.3 - DESCRIÇÃO DA IMPLEMENTAÇÃO

O ambiente de implementação utilizado foi um circuito de tomadas na mesma fase pertencentes a uma edificação qualquer (rede PLC). Para implementação deste projeto foram utilizadas 3 tomadas de um circuito elétrico 220 V, 60 Hz.

O circuito de tomadas na mesma fase representará a rede PLC. Uma rede sem fio (*wireless*) será implementada para fornecer acesso às aplicações clientes. Desta forma, o projeto foi subdivido em 5 módulos: 1 Servidor Residencial, 2 Módulos de Automação (atuadores), 2 Módulos Cliente, apresentados nas seções a seguir.

4.3.1 - Servidor Residencial

O Servidor Residencial é a central do projeto, sendo composto pelos seguintes *hardwares*: 1 roteador *wireless* (opcional), 1 mini-PC, 1 *dongle WiFi*, 1 microcontrolador e 1 modem PLC, conforme mostrado na Figura 4.2.

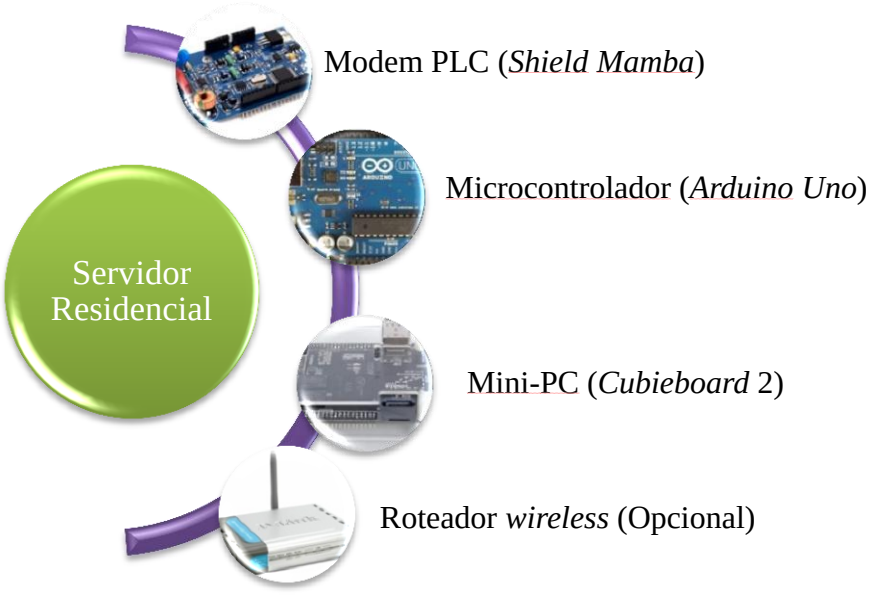

Figura 4.2 – Servidor Residencial (*hardware*)
(Fonte: Autor)

Todas as informações e comandos passam por ele durante a execução e utilização do sistema. Este módulo é responsável por receber os dados dos Módulos Cliente através da rede sem fio e se comunicar com os Módulos de Automação através da rede PLC.

A rede sem fio foi implementada utilizando-se um acessório opcional, um roteador D-LINK modelo DSL-2640T. Outras opções de implementações que são suportadas pelo projeto seriam: a criação de uma rede *ad-hoc* no próprio sistema operacional do Servidor Residencial para compartilhamento de dados sem fio ou a interconexão do Servidor Residencial e Módulos Clientes através da Internet.

Tendo a rede sem fio configurada para compartilhamento de dados através do roteador, foi utilizado um *Cubieboard 2* (mini-PC) para recebimento, processamento e envio de dados. Conforme especificado na Seção 3.1.4, o mini-PC possui características muito semelhantes a um PC comum.

Para o recebimento de dados provenientes dos Módulos Cliente através da rede sem fio, foi conectado ao mini-PC, o *dongle WiFi* da marca *Edup* modelo EP-N8508GS, o qual funciona como uma placa de rede sem fio, visto que o mini-PC só possui placa de rede *Ethernet* de forma nativa, exigindo a conexão de uma placa de rede sem fio.

Para o envio de dados através da rede PLC foi conectado ao mini-PC, através da porta USB, a interface composta pelo microcontrolador *Arduino Uno* em conjunto com um modem PLC *Mamba*. Esta interface será responsável por receber os dados e convertê-los em sinal passível de ser transmitido através do circuito elétrico de tomadas (rede PLC). Toda implementação de *hardware* descrita está ilustrada na Figura 4.3.

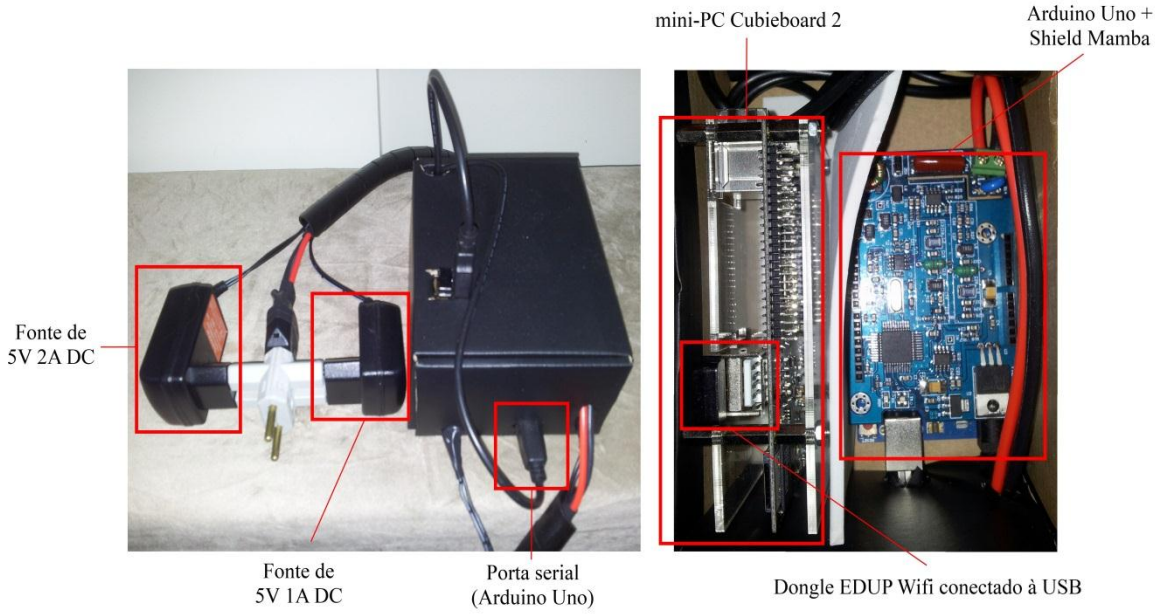

Figura 4.3 – Implementação física do Servidor Residencial
(Fonte: Autor)

O Servidor Residencial utiliza, ainda, *software* para coordenar o *hardware* implementado e a comunicação com as redes (rede *wireless* e rede PLC). Dentre os *software* instalados e desenvolvidos, estão: 1 Sistema Operacional Linux, 1 *Web Server*, 1 *Web Service* e 1 programa para a comunicação através da rede PLC.

A Figura 4.4 ilustra a organização hierárquica das camadas de *software* utilizadas, ou seja, o programa para a comunicação através da rede PLC (camada mais interna) recebe o comando do *Web Service* que está hospedado no *Web Server* que está instalado no Sistema Operacional (camada mais externa).

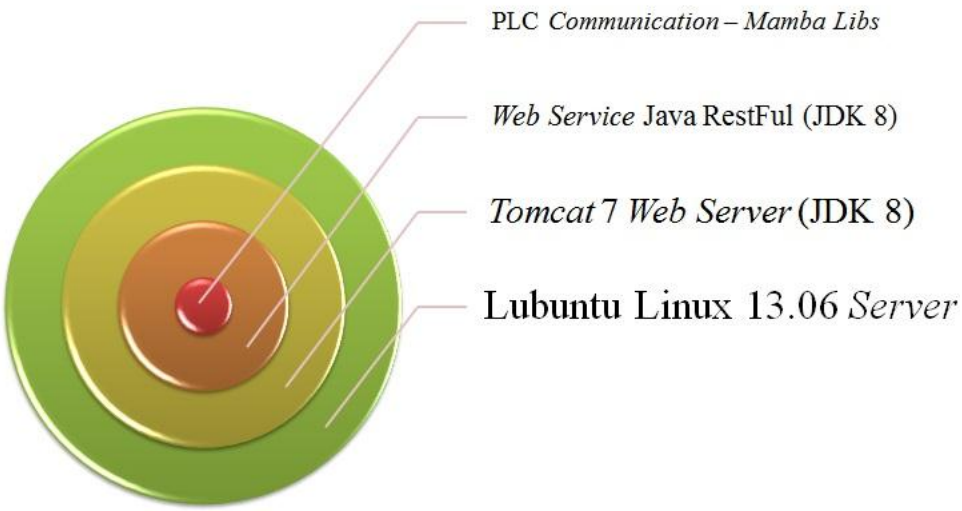

Figura 4.4 – Servidor Residencial (*software*)
(Fonte: Autor)

O Sistema Operacional (SO) instalado foi uma compilação do Sistema Operacional Linux para processadores ARM-hf, o Lubuntu 13.06 *Server* (CUBIEBOARD, 2013). Este SO possui o conjunto de comandos semelhantes a compilação do Linux para PC, porém, as duas compilações se diferem na hora de executar instruções no processador, pois cada processador possui um *set* (conjunto) de instruções específicos além de suportar operações diferentes e tratar operandos de forma distinta.

Como a maior parte da codificação foi desenvolvida utilizando a linguagem Java e a mesma utiliza o conceito de código móvel, foi necessário instalar uma máquina virtual Java no mini-PC para que ela pudesse executar código desenvolvido na linguagem Java. Para este *hardware* foi instalada a compilação do JDK 8 *beta* com suporte a processadores ARM-hf.

Tendo o JDK instalado foi instalado e configurado um *Web Server* (servidor *web*), o *Tomcat* 7. O *Tomcat* foi utilizado como *container* de aplicações, neste caso responsável pela hospedagem de um *Web Service* (serviço *web*). Além da configuração do *Web Server*, foi

criado um script (APÊNDICE A – Script *Web Server* automático) para sua execução automática ao ligar o Servidor Residencial.

Para coordenar o recebimento de dados da rede sem fio e envio de dados em *broadcast* para rede PLC (todos os Módulos de Automação recebem o sinal, porém, somente o endereçado na mensagem será acionado) foi desenvolvido um *Web Service* utilizando o modelo arquitetural *RESTFul* na linguagem Java, o *RestWebServices* (APÊNDICE B – *WebService RESTFul: RestWebServices*). O *RestWebServices*, pode ser utilizado por qualquer cliente (independente da linguagem de programação), como especificado na Seção 3.2.1, apenas por utilizar métodos HTTP.

Para receber os dados provenientes do *Web Service* através da porta USB, foi compilado e gravado um código em Linguagem C, o mamba_server (APÊNDICE C – mamba_server). Este código é responsável por receber dados através da porta serial e enviá-los através do Modem PLC *Mamba* para outros dispositivos presentes na rede PLC.

4.3.2 - Módulos de Automação (Atuadores)

O Módulo de Automação é responsável pela atuação, acionamento dos dispositivos eletrônicos a serem automatizados, por isso pode ser denominado atuador. Este módulo possui os seguintes *hardwares* em sua composição: 1 microcontrolador, 1 modem PLC e 4 circuitos para acionamento de relés, conforme mostrado na Figura 4.5.

Assim como foi especificado na Seção 4.3.1, neste módulo foi utilizado a interface composta pelo microcontrolador *Arduino Uno* em conjunto com um modem PLC *Mamba*. Esta interface será responsável por receber os dados transmitidos pelo Servidor Residencial através do circuito elétrico de tomadas (rede PLC). Este módulo não possui comunicação com a rede *wireless*, a comunicação com ele se dá, única e exclusivamente, através da rede PLC.

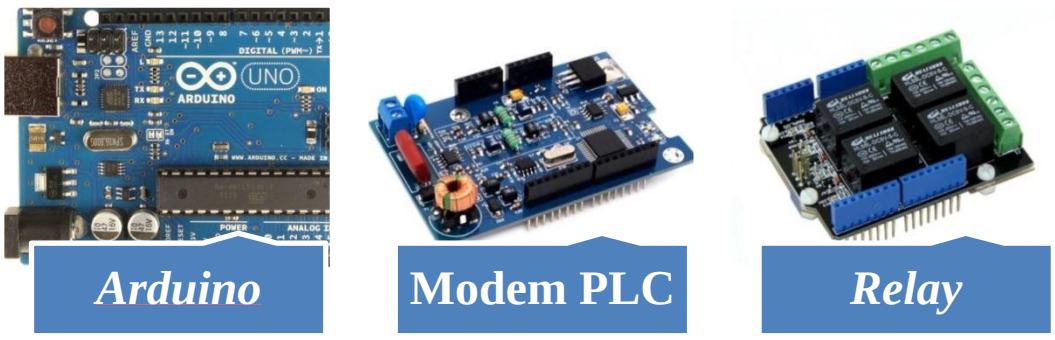

Figura 4.5 – Módulo de Automação (*hardware*)
(Fonte: Autor)

Para acionar ligando ou desligando os dispositivos eletrônicos foi acoplado às portas digitais do microcontrolador *Arduino Uno* um *Shield Relay* v2.0. Este *Shield* possui um circuito que é composto por 4 relés, o que permite a automação de até 4 dispositivos eletrônicos em conjunto com 4 circuitos de segurança composto por fotoacopladores o que evita retorno de descarga elétrica ao circuito do microcontrolador. Toda implementação física do Módulo de Automação está ilustrada na Figura 4.6.

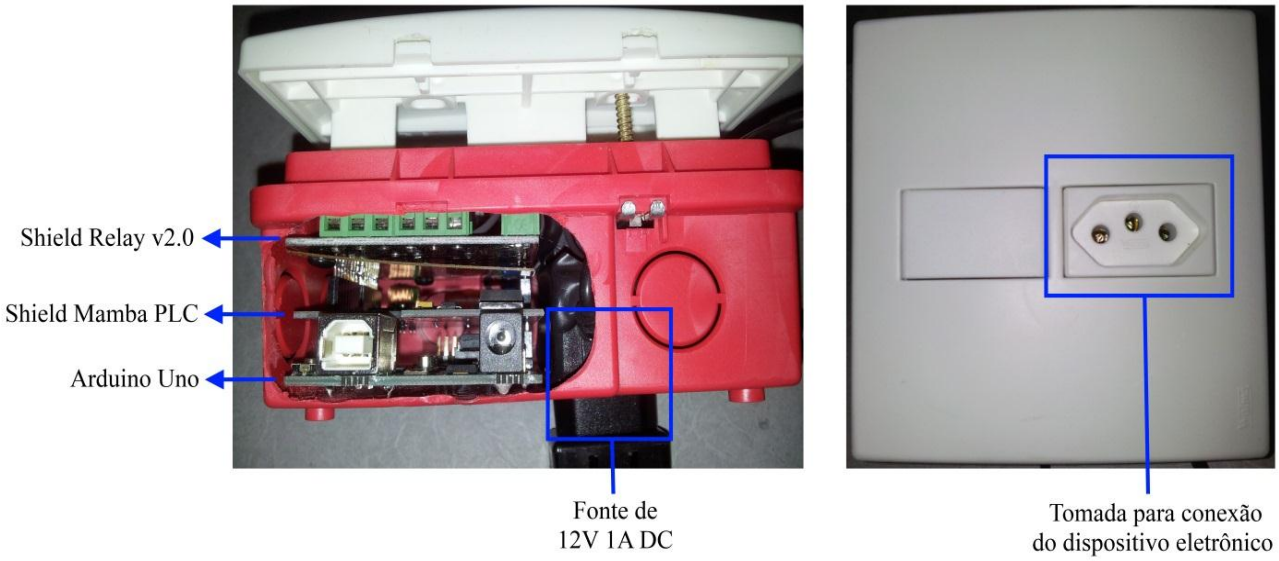

Figura 4.6 – Implementação física do Módulo de Automação
(Fonte: Autor)

Conforme especificado na Seção 4.3.1, para este módulo também foi compilado e gravado um código em linguagem C, o mamba_relay (APÊNDICE D – mamba_relay) que utiliza as bibliotecas *Mamba Libs* para tratamento e envio do sinal através da rede PLC (o código e biblioteca são mostrados na Figura 4.7 como camadas de *software* deste Módulo).

Esse código (Aplicativo em C) não possui comunicação e tratamento de dados com a porta serial do microcontrolador *Arduino*, possui apenas código necessário para o recebimento de dados através do modem PLC *Mamba* que está acoplado ao microcontrolador *Arduino*.

O código mamba_relay também identifica o endereço especificado na mensagem e caso o Módulo de Automação possua tal endereço o mesmo acionará o circuito do relé para ligar o dispositivo eletrônico, caso esteja desligado ou desligá-lo caso esteja ligado.

Figura 4.7 – Módulo de Automação (*software*)
(Fonte: Autor)

4.3.3 - Módulos Cliente

Para ilustrar a generalização que o *Web Service* proporciona, com o foco na interface para o público de deficientes físicos, foi implementado dois Módulos Cliente: 1 Módulo *Kinect* e 1 Módulo *Smartphone*. Nenhum destes módulos possuem comunicação com a rede PLC possuem comunicação apenas com a rede sem fio implementada no projeto. Estes módulos serão apresentados nas seções a seguir:

4.3.3.1 - Módulo *Kinect*

O Módulo *Kinect* é o Módulo Cliente responsável por identificar comandos através de gestos mapeados pela imagem capturada pelo dispositivo *Kinect* da *Microsoft*, conforme especificado na Seção 3.2.4. Este Módulo atende aos deficientes físicos com deficiência em membros inferiores e superiores (tetraplégicos), porque permite o acesso aos comando através de movimentos da cabeça e olhos exigindo dois requisitos: facilidade de interação com o usuário, e mobilidade. Sendo assim este módulo exige o seguinte *hardware* (ilustrado na Figura 4.8) para funcionamento: 1 mini-PC, 1 *dongle WiFi*, 1 *Microsoft Kinect*, 2 *powerBank* (bateria).

Para processar as imagens e enviar os dados através da rede sem fio é necessário um dispositivo como um PC, porém, com baixo consumo de energia. Desta forma, foi utilizado o *Cubieboard* 2 (mini-PC) que possui capacidade de processamento próxima de um PC comum e consumo de 5 V, 2 A, conforme especificado na Seção 3.1.4.

Para o envio de dados para o Servidor Residencial através da rede sem fio, foi conectado ao mini-PC, o *dongle WiFi* da marca EDUP modelo EP-N8508GS, o mesmo utilizado no Servidor Residencial.

Figura 4.8 – Módulo Cliente *Kinect* (*hardware*)
(Fonte: Autor)

Os comandos serão obtidos através de imagens processadas e capturadas do usuário o que atende um dos requisitos do módulo a facilidade de interação com o usuário, para atender esta necessidade foi utilizado um dispositivo projetado para jogos do *Console XBOX*, o *Microsoft Kinect*. Este dispositivo foi conectado ao mini-PC através de uma porta USB, o que permite a transmissão de dados do *Kinect* para o mini-PC de forma serializada.

Para atender o requisito de mobilidade o fornecimento de energia não poderia ser obtido através de transformação de tensão da rede elétrica, pois isso fixaria o usuário em um ponto. Desta forma, utilizou-se 2 *powerBanks* neste módulo: 1 *powerBank* com saída de 5 V, 2 A e 10.000 mAh para fornecer energia ao mini-PC e 1 *powerBank* com saída de 12 V, 1 A e 6.900 mAh para fornecer energia ao *Kinect*. Este *hardware* garantirá que o dispositivo poderá ser utilizado em uma cadeira de rodas por um período aproximado de até 20 horas. Toda implementação física do Módulo Cliente *Kinect* está ilustrada na Figura 4.9.

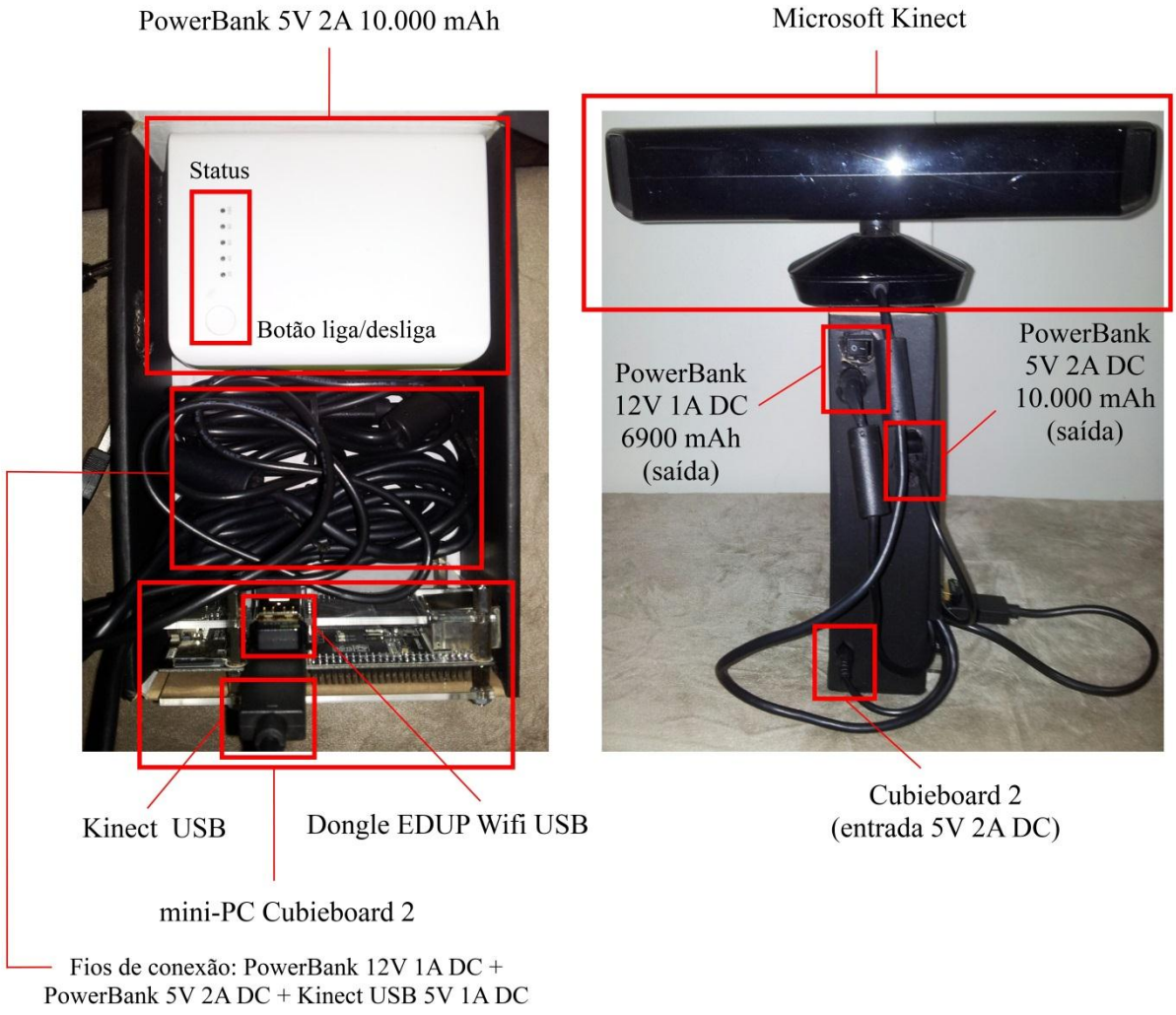

Figura 4.9 – Implementação física do Módulo Cliente *Kinect*
(Fonte: Autor)

O Módulo *Kinect* utiliza, ainda, *software* para coordenar o *hardware* implementado e a comunicação com a rede *wireless*. Dentre os *software* instalados e desenvolvidos, estão: 1 Sistema Operacional Linux, *Kinect SDK* e 1 aplicativo *desktop* para processamento das imagens e envio dos comandos para o Servidor Residencial através da rede *wireless*.

O Sistema Operacional (SO) instalado foi uma compilação do Sistema Operacional Linux para processadores ARM-hf, o Lubuntu 12.10 Desktop (CUBIEBOARD, 2013). Este SO possui o conjunto de comandos semelhantes à compilação do Linux para PC, porém, as duas compilações se diferem na hora de executar instruções no processador, pois cada processador possui um *set* (conjunto) de instruções específicos além de suportar operações diferentes e tratar operandos de forma distinta.

Como os *frameworks* utilizados no aplicativo *desktop* para o processamento de imagens e envio de comandos foi desenvolvido utilizando a linguagem Java e também a linguagem Python, foi necessário instalar uma máquina virtual Java no mini-PC, conforme especificado na Seção 4.3.1 e também o Python para Linux (composto pela *standard library* e também a biblioteca *OpenCV* conforme especificado na Seção 3.2.2.2).

Tendo o JDK instalado foi necessário instalar o *software* que dá suporte ao *Kinect* em ambientes Linux, o *Kinect SDK*. Em plataforma Linux existem duas formas de implementar o *Kinect SDK*: através do *Framework OpenNI + NITE* (Seção 3.2.5) ou através do *Framework libfreenect* (Seção 3.2.6).

Estes *frameworks* não possuem compilação disponível para Linux que utilizam processador ARM-hf, desta forma foi necessário criar uma compilação própria para cada um destes *frameworks* para o processador ARM-hf.

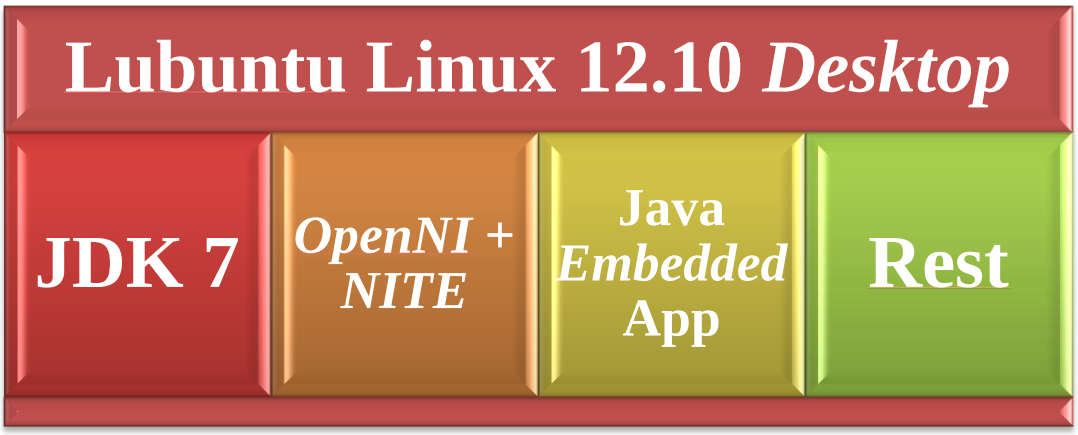

Figura 4.10 – Módulo *Kinect* Java (*software*)
(Fonte: Autor)

Com todo ambiente instalado e configurado foi desenvolvido um aplicativo em linguagem Java, o *KinectClient* Java (APÊNCICE E – Aplicativo *KinectClient* Java). Este aplicativo utiliza o *OpenNI + NITE* para receber as imagens através da USB e realizar o tratamento delas para identificação de gestos da mão e posterior tradução em comandos, todos os *softwares* e tecnologias utilizadas no Módulo Cliente *Kinect* Java são mostrados na Figura 4.10.

Conforme abordado na Seção 3.2.5 o *Framework OpenNI + NITE* possui vasta aplicação no tratamento e mapeamento de imagem, porém nos testes realizados na plataforma proposta (*Cubieboard* + Linux) obteve-se um desempenho aquém do esperado, o qual foi exemplificado pelo grande tempo de espera (entre 5 e 10 segundos) entre a imagem e definição do comando a ser enviado pela rede *wireless*. Além deste resultado, ocorreram constantes erros no tratamento de imagens e uma grande quantidade de interrupções no recebimento das imagens provenientes do *Kinect*.

Através destes resultados foi necessário o desenvolvimento de uma nova proposta de aplicativo *desktop* para o processamento de imagens, desta vez, por meio de um *framework* mais simples e leve, o *Framework libfreenect*. Tendo todo ambiente instalado e configurado foi desenvolvido um aplicativo em linguagem Python, o *KinectClient* Python (APÊNCICE F – Aplicativo *KinectClient* Python).

Este aplicativo utiliza o *libfreenect* para receber as imagens através da USB e realizar o tratamento delas para identificação de gestos da mão, posteriormente da cabeça e também movimentos dos olhos. Ao reconhecer tais movimentos ocorre a tradução em comandos para a automação. Todos os *softwares* e tecnologias utilizadas no Módulo Cliente *Kinect* Python são mostrados na Figura 4.11.

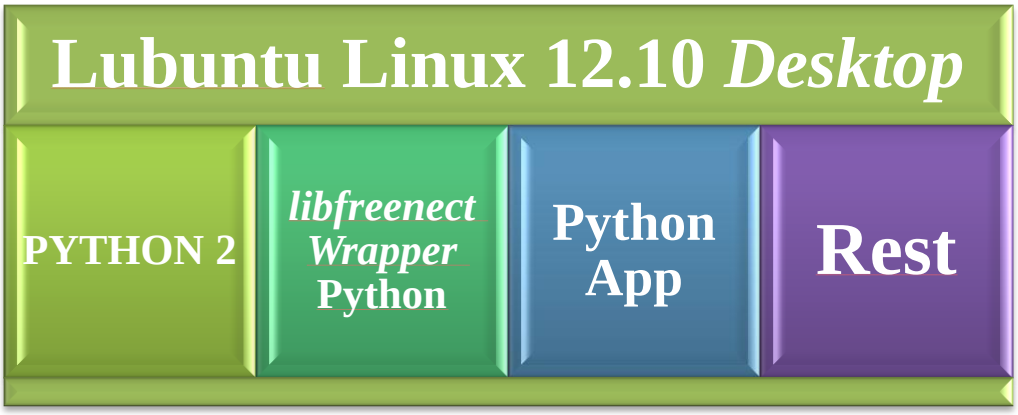

Figura 4.11 – Módulo *Kinect* Python (*software*)
(Fonte: Autor)

Nos testes de utilização deste aplicativo não foi observado grande tempo de espera (tempo de espera máximo de 4 segundos) e não ocorreu nenhuma interrupção inesperada no recebimento de imagens.

Além da identificação do comando, ambos aplicativos Java e Python realizam uma requisição HTTP especificando o comando para o *Web Service RESTFul* hospedado no Servidor residencial, este, por sua vez, realiza a comunicação com a rede PLC acionando o dispositivo eletrônico conforme explicado na Seção 4.3.1.

4.3.3.2 - Módulo *Smartphone*

O Módulo *Smartphone* é o Módulo Cliente responsável por utilizar um *hardware* comum para usuário como meio acionamento aos dispositivos conectados à rede PLC. Este Módulo atende aos deficientes físicos com deficiência em membros inferiores (paraplégicos), porque permite o acesso aos comando através de cliques com o dedo.

Para este escopo o único *hardware* necessário para o funcionamento do módulo é um *smartphone* que possua conexão a redes *wireless*. Alguns exemplos de modelos são mostrados na Figura 4.12.

Figura 4.12 – Módulo de *Smartphone* (*hardware*)
(Fonte: mobilemonitorsoftware.wordpress.com/2013/07/25/smartphones-spy, adaptada)

Para ilustrar a utilização de um *smartphone* como meio de acionar dispositivos automatizados, utilizou-se dispositivos com sistema operacional *Android* instalado. Desta forma, foi desenvolvido um aplicativo para *Android*, o *AndroidClient* (APÊNCICE G – Aplicativo *AndroidClient*). Este aplicativo é baseado na estrutura de lista, detalhes e fragmentos (técnicas de desenvolvimento para *Android* ilustradas na Figura 4.13), ele utiliza o *touchscreen* para a escolha dos comandos.

Ao selecionar o comando, o aplicativo *Android* realiza uma requisição HTTP especificando o comando escolhido para o *Web Service RESTFul* hospedado no Servidor residencial. Este, por sua vez, realiza a comunicação com a rede PLC acionando o dispositivo eletrônico conforme explicado na Seção 4.3.1.

Caso haja algum erro na requisição ao servidor, na conexão do Servidor Residencial com a interface *Arduino + Mamba* ou na identificação do endereço em um dos Módulos de Automação, o *Web Service* retorna estes estados que serão informados na tela do *smartphone* utilizado. A tela de escolha de comando e detalhamento dos estados estão presentes na Figura 4.14.

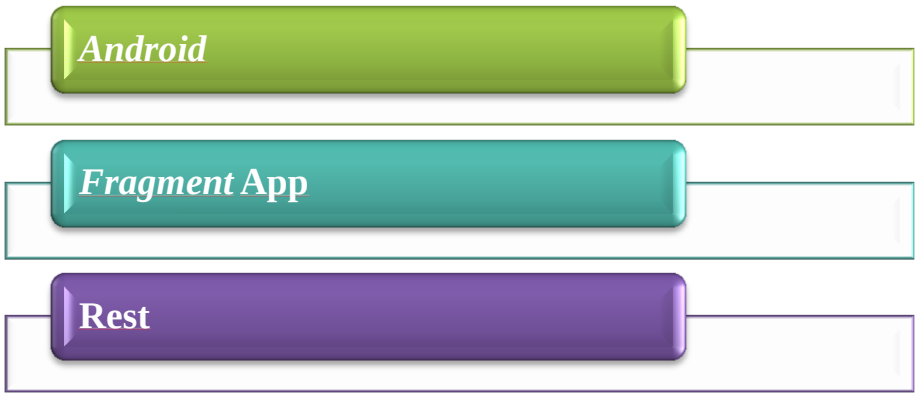

Figura 4.13 – Módulo *Smartphone* (*software*)
(Fonte: Autor)

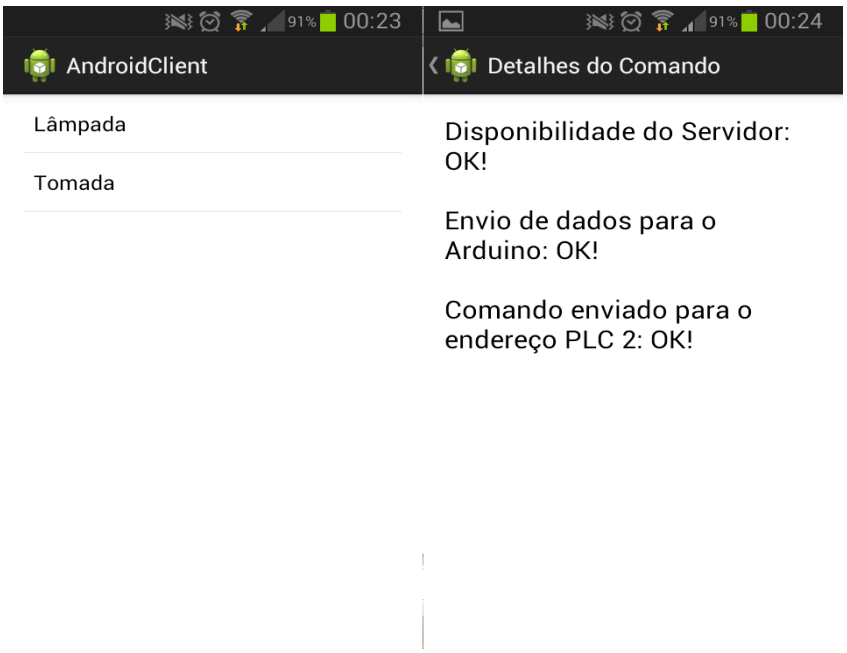

Figura 4.14 – *AndroidClient* (telas do aplicativo)
(Fonte: Autor)

Neste módulo também foi implementado a comunicação através da tecnologia NFC, abordada na Seção 3.1.6. Para implementação do NFC no Módulo Cliente *Smartphone* foi utilizado: 1 *Smartphone* com tecnologia NFC embarcada (não há dependência com o aplicativo *Android* desenvolvido) e 2 *tags* NFC (mostradas na Figura 4.15).

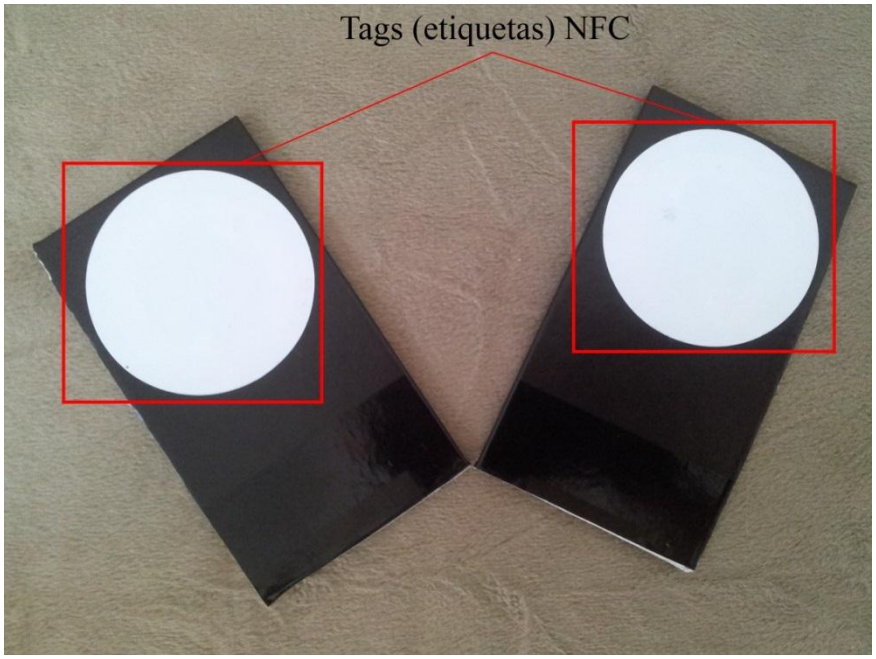

Figura 4.15 – *Tags* NFC utilizadas no projeto
(Fonte: Autor)

O *smartphone* com tecnologia NFC foi utilizado no modo de operação de escrita para gravar nas *Tags* NFC a requisição HTTP necessária para cada comando da automação. Ao terminar a gravação, as *tags* já estavam prontas para ser lidas por qualquer s*martphone* que possuísse NFC utilizando o modo de operação de leitura (modo padrão ao habilitar o NFC no s*martphone*) e tivesse conectado a rede *wireless* da residência.

Através deste simples procedimento foi obtido mais uma forma de utilização do Sistema de Automação Residencial para o usuário (paraplégico), a comunicação através da tecnologia NFC. Esta expansão ocorreu devido a generalização implementada pelo *Web Service* presente no Servidor Residencial (Seção 4.3.1).

Com isto, conclui-se as etapas de desenvolvimento e implementação, passando agora aos testes, utilização prática da solução, resultados obtidos, custos e avaliação global da Solução Proposta.

CAPÍTULO 5 - APLICAÇÃO PRÁTICA DA SOLUÇÃO PROPOSTA

5.1 - APRESENTAÇÃO DA ÁREA DE APLICAÇÃO DA SOLUÇÃO

A automação residencial é uma área de grande evolução, porém, a maioria dos projetos apresentam grande especificidade e complexidade de projeto, bem como a necessidade de implementação de nova infraestrutura, conforme detalhado no Capítulo 2.

A Solução proposta baseia-se em um projeto de automação em qualquer edificação sem a necessidade de especificidades no projeto ou infraestrutura dedicada. Desta forma, este projeto possui generalização no cenário de aplicação, além de possuir flexibilidade na interface com usuário por utilizar *software* livre e *Web Service* conforme explicitado no Capítulo 4.

Com isto, a Solução proposta abrange toda área de aplicação de qualquer sistema de Automação Residencial. Isto, porque possui um sistema genérico, de simples instalação e configuração e flexível na interação com o usuário.

5.2 - DESCRIÇÃO DA APLICAÇÃO DA SOLUÇÃO

Para generalização e facilidade de instalação, a Solução se beneficia da infraestrutura da rede elétrica para a transmissão de dados em qualquer edificação, com finalidade de automatizar o acionamento de eletrodomésticos e iluminação. No projeto, foram utilizadas três tomadas como nós da rede PLC, sendo um nó para o Servidor Residencial e dois nós para os Módulos de Automação, os quais estão conectados aos dispositivos eletrônicos a serem automatizados.

Para o usuário possuir acesso a automação através da rede PLC foi criado uma rede *wireless* através de um roteador (opcional, conforme explicado na Seção 4.3.1) com 3 nós: 1 nó para o Servidor Residencial e 2 nós para os Módulos Clientes. Sendo assim, o Servidor Residencial (Central) realiza todo o trabalho de receber os comandos enviados pelos Módulos Cliente e enviá-los para os Módulos de Automação através da Rede PLC.

Nos Módulos Cliente o foco foi a acessibilidade (Capítulo 2), pois o projeto destina-se a Automação Residencial para atender usuários com deficiência física. Desta forma, foram desenvolvidos dois Módulos Cliente: Módulo *Smartphone* e Módulo *Kinect*.

O Módulo *Smartphone* destina-se a deficientes que apresentam paraplegia (paralisação de membro inferiores), sendo assim, o usuário utiliza as mãos para acionar os dispositivos eletrônico através de aplicativo instalado em seu *smartphone* ou através da leitura de *tags*

NFC. Este Módulo se beneficia naturalmente da mobilidade que o *Gadget* proporciona (Seção 4.3.3.2).

O Módulo *Kinect* permite o acesso de deficientes que apresentem a tetraplegia (paralisação de membros inferiores e superiores). Este módulo utiliza o *Kinect* para reconhecimento de gestos através das imagens do movimento da cabeça e olhos como comandos de acionamento (Seção 4.3.3.1). Por utilizar baterias no fornecimento de energia, este módulo possui grande mobilidade permitindo o seu acoplamento a uma cadeira de rodas.

5.3 - RESULTADOS DA APLICAÇÃO DA SOLUÇÃO

A Solução Proposta permite a automação de dispositivos que se encontram em mesma fase (rede PLC), porém, limita a automação a circuitos que se encontram em fases diferentes. Devido a escolha de dispositivos baseando-se no custo/benefício, o modem PLC (*Shield Mamba*) apresentou uma taxa de transmissão de apenas 2048 bps, o suficiente para envio de mensagem endereçada (através da rede PLC) contendo o comando da automação.

A rede *wireless* compartilhada pelos Módulos Cliente e Servidor Residencial foi obtida através de um roteador *wireless*, o qual proporcionou uma taxa de transmissão de 54 Mbps. Esta taxa de transmissão atende acima das necessidades para utilização do *Web Service* hospedado no Servidor Residencial.

Foram realizados testes implementando uma rede *ad-hoc* através do Servidor Residencial em substituição a rede *wireless* através do roteador, porém, a mesma possui limitação de 4 dispositivos conectados em simultâneo, além de não ser reconhecida por dispositivos *Android* (Módulo *Smartphone*).

O Servidor Residencial apresentou uma excelente eficiência energética ao alcançar um consumo de apenas 15 Wh, o que permite que ele fique conectado a tomada sem apresentar o consumo que um PC comum apresentaria, cerca de 40 Wh.

O Módulo de Automação foi implementado utilizando o acionamento de apenas um dispositivo eletrônico (a Figura 5.1 ilustra o acionamento de uma lâmpada durante os testes realizados), porém, o *Shield Relay* v2.0 utilizado permite o acionamento de mais 3 tomadas, totalizando 4 tomadas por módulos.

Além disto, este módulo, possui mais 12 portas digitais (além das 4 utilizadas pelos relés, totalizando 16) e 6 analógicas que permitem a automação de mais dispositivos eletrônicos, bem como a implementação de sensores a serem utilizados na automação.

Figura 5.1 – Acionamento de uma lâmpada através do Módulo de Automação (Fonte: Autor)

O Módulo Cliente *Kinect* proporcionou o maior número de testes para sua implementação, com necessidade de compilação de bibliotecas para conexão do *Kinect* ao SO Linux para a arquitetura ARM-hf (Seção 4.3.3.1).

Neste módulo foi possível reconhecer gestos da cabeça e movimento dos olhos através das imagens capturadas pelo dispositivo *Kinect*. Porém, conforme abordado na Seção 4.3.3.1, só houveram resultados satisfatórios através do aplicativo *KinectClient* Python. A Figura 5.2 ilustra o teste realizado através deste aplicativo, em destaque (feito pelo próprio aplicativo *KinectClient* Python, para ilustrar o tratamento de imagem) está: a cabeça em vermelho e os olhos em verde.

Devido ao grande tempo de espera proporcionado pelo aplicativo *KinectClient* Java, não foi desenvolvido o código para reconhecimento de gestos da cabeça e olhos nesta linguagem, foi desenvolvido apenas o reconhecimento de gestos da mão.

As baterias de 10.000 mAh (fonte de alimentação para o mini-PC *Cubieboard* 2) e de 6900 mAh (fonte de alimentação para o *Microsoft Kinect*), mostradas na Figura 4.9, durante os testes, proporcionaram uma autonomia de 20 horas de funcionamento ininterruptas, não havendo a necessidade de conexão com a rede elétrica, atendendo aos requisitos de mobilidade exigidos por uma pessoa que utiliza cadeira de rodas.

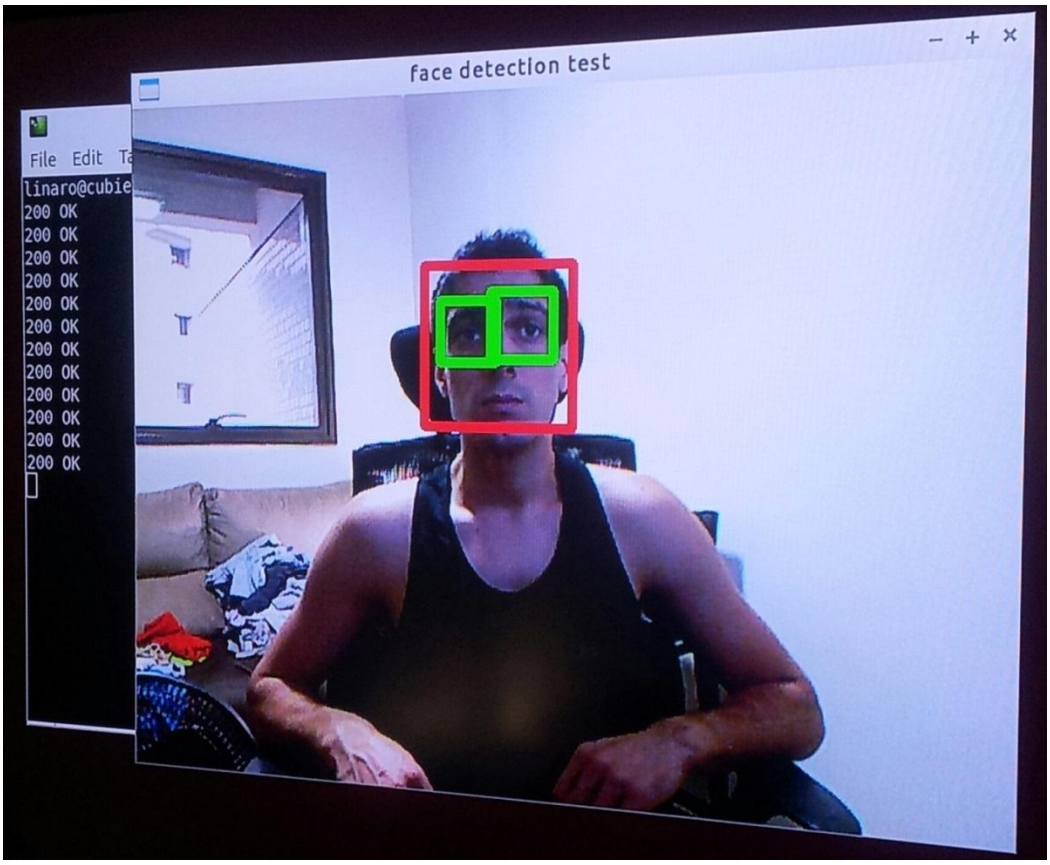

Figura 5.2 – Tratamento de imagem feito pelo aplicativo *KinectClient* Python (Fonte: Autor)

5.4 - CUSTOS DA SOLUÇÃO

Um projeto simples de Automação Residencial que proporciona apenas acionamento de dispositivos eletrônicos (liga/desliga) para dois pontos de automação apresentam um custo em média de R$ 3.000,00 (AURESIDE, 2013) e com grande complexidade de instalação. Foram levantados os custos com o desenvolvimento da solução aqui proposta, cujos resultados encontram-se nas Tabelas 5.1, 5.2, 5.3, 5.4 e 5.5.

Tabela 5.1 – Custos Servidor Residencial

Dispositivo	**Valor**
mini-PC *Cubieboard* 2	R$ 129,80
Modem PLC - *Shield Mamba*	R$ 107,80
Roteador *Wireless* D-Link DSL-2640T (opcional)	R$ 70,00
Microcontrolador *Arduino Uno*	R$ 69,00
Fonte *Kaiomy* 5 V 2 A DC	R$ 36,00
Fonte *Kaiomy* 5 V 1 A DC	R$ 21,00
Dongle WiFi EDUP EP-N8508GS	R$ 15,40
TOTAL	**R$ 449,00**

Fonte: Autor

Tabela 5.2 – Custos Módulo de Automação

Dispositivo	Valor
Modem PLC - *Shield Mamba*	R$ 107,80
Microcontrolador *Arduino Uno*	R$ 69,00
Shield Relay v2.0	R$ 44,00
Fonte *Kaiomy* 12 V 1 A DC	R$ 12,00
TOTAL	**R$ 232,80**

Fonte: Autor

Tabela 5.3 – Custos Módulo *Smartphone*

Dispositivo	Valor
Kit 2 *Tags NFC* adesivas	R$ 9,38
TOTAL	**R$ 9,38**

Fonte: Autor

Tabela 5.4 – Custos Módulo *Kinect*

Dispositivo	Valor
Microsoft Kinect	R$ 241,98
mini-PC *Cubieboard* 2	R$ 129,80
PowerBank 10.000 mAh 5V 2 A DC	R$ 65,78
PowerBank 6.900 mAh 12V 1 A DC	R$ 44,00
Dongle WiFi EDUP EP-N8508GS	R$ 15,40
TOTAL	**R$ 496,96**

Fonte: Autor

Tabela 5.5 – Custos Solução Proposta

Módulo	Quantidade	Valor Unitário	Valor
Servidor Residencial	1	R$ 449,00	R$ 449,00
Módulo de Automação	2	R$ 232,80	R$ 465,60
Módulo *Kinect*	1	R$ 496,96	R$ 496,96
Módulo *Smartphone*	1	R$ 9,38	R$ 9,38
TOTAL	**5**	**R$ 1.188,14**	**R$ 1.420,94**

Fonte: Autor

Foram abordados os custos de cada módulo do projeto: 1 Servidor Residencial, 2 Módulos de Automação e 2 Módulos Cliente.

Com base nas Tabelas 5.1, 5.2, 5.3, 5.4 e 5.5 o custo total do projeto fica no valor R$ 1.420,94, 47,36% da média de um projeto simples de automação (R$ 3.000,00). Porém, com o diferencial de fornecer um Módulo Cliente com o dispositivo *Kinect* já incluso. A maioria dos

Sistemas de Automação Residencial não fornecem Módulo Cliente (*hardware*) ou fornecem com alto custo adicional (AURESIDE, 2013).

Para a redução do custo do projeto, caso o cliente não necessite de sensores nos Módulos de Automação, não seria necessário a utilização de um microcontrolador *Arduino* em cada módulo, reduzindo em R$ 138,00 (2 x R$ 69,00) o custo do projeto.

Outra possibilidade de economia seria a implementação de apenas um relé por Módulo de Automação, o que reduziria em R$ 88,00 (2 x R$ 44,00). No total, para esta implementação mais simples, o valor passaria de R$ 1.420,94 para R$ 1.194,94 uma economia de R$ 226,00.

5.5 - AVALIAÇÃO GLOBAL DA SOLUÇÃO

A partir desta implementação pode-se concluir que a Solução Proposta possui alta aplicabilidade por ser adaptável a qualquer edificação que possua rede elétrica, sendo esta o único requisito para sua instalação. Porém, possui a limitação de funcionamento em apenas uma fase.

Como vantagens desta solução estão: simplicidade de instalação (não sendo necessário a implementação de nova infraestrutura na residência), baixo custo (aproximadamente a metade de um projeto simples de Automação Residencial), alta capacidade de expansão do projeto e facilidade na interação com usuário (reconhecimento de movimentos simples da cabeça e dos olhos através do Módulo Cliente *Kinect*).

A capacidade de expansão pode ser observada a partir de duas visões:

- Suporte permitido pelo microcontrolador em expandir a quantidade de dispositivos eletrônicos acionados e/ou sensores implementados em cada Módulo de Automação;
- Flexibilidade existente no *Web Service* hospedado no Servidor Residencial (Seção 4.3.1) que permite a sua utilização de qualquer dispositivo cliente, independente da arquitetura, linguagem de Programação ou plataforma utilizada, exemplificado pela utilização do aplicativo Python do Módulo *Kinect*, do aplicativo Java *Android* para o Módulo *Smartphone* e utilização da tecnologia de comunicação NFC.

Sendo assim, a Solução Proposta pode ser caracterizada como um projeto que une simplicidade, adaptabilidade, baixo custo, flexibilidade e inclusão utilizando apenas recursos de *software* livre.

CAPÍTULO 6 - CONCLUSÃO

6.1 - CONCLUSÕES

A acessibilidade envolve o *Design* Inclusivo, oferta de um leque variado de produtos e serviços que cubram as necessidades de diferentes populações (incluindo produtos e serviços de apoio), adaptação, meios alternativos de informação, comunicação, mobilidade e manipulação.

O conceito de acessibilidade está presente na vida de pessoas com necessidade especial: pessoas com algum tipo de deficiência, idosos e outras pessoas com alguma incapacidade. Esta população representa um enorme público alvo que almejam melhorias na qualidade de vida.

Esta crescente demanda tem colocado a acessibilidade em primeiro lugar quando o assunto é construção civil, seja ela residencial ou comercial. Tecnologias voltadas para automação residencial contribuem para o provimento de acessibilidade, pois proporcionam meios alternativos de comunicação, mobilidade e manipulação.

Com este enfoque, a Solução proposta apresentada no transcorrer deste trabalho, apresentou resultados que atendem ao Cenário apresentado:

- construções antigas;
- implementação de um sistema de Automação Residencial;
- prover acessibilidade;
- e boa relação custo/benefício.

Esta Solução aborda a transmissão de dados através da rede elétrica, a qual foi realizada com sucesso, o reconhecimento facial, através de APIs Java *open source* e APIs Python *open source*, o qual foi realizado com êxito. Esta funcionalidade permitiu o reconhecimento de comandos de deficientes que possuem dificuldade de locomoção.

A característica *open source* das bibliotecas, *device drivers* e APIs, permitiram as alterações necessárias quando da implementação do *Microsoft Kinect* em conjunto com o mini-PC *Cubieboard* (computador com processamento baseado na arquitetura ARM), são elas:

- recompilação de todos os *device drivers* e bibliotecas, com a finalidade de possibilitar a comunicação do *Microsoft Kinect* com o Sistema Operacional Linux para processadores ARM;
- alteração de instruções presentes nas APIs, as quais não podem ser executadas por processadores ARM.

Além disto, o projeto permite liberdade de desenvolvimento de aplicativos clientes, devido a utilização de um modelo arquitetural de serviços *web* REST. O que significa que qualquer *software* desenvolvido em qualquer linguagem de programação pode usufruir do sistema de automação residencial (como exemplo, no projeto foram implementados aplicativos clientes com linguagem Java, Python e leitura de *tag* NFC).

Esta característica apresenta um grande diferencial perante o mercado, já que a maioria dos Sistemas de Automação Residencial restringem o acesso aos sistemas aos *softwares* proprietários das empresas.

Em relação ao *hardware*, houve o cuidado de desenvolver todo o circuito composto por: microcontrolador, modem PLC e relés, dentro de uma caixa de tomada, fator que facilita a implementação do projeto em modelo comercial.

Com enfoque na acessibilidade, a implementação de *hardware* do Módulo Cliente *Kinect* visou a autonomia energética para que este Módulo pudesse ser utilizado em uma cadeira de rodas sem que haja a preocupação com fios ou necessidade de utilização de um computador portátil com um *notebook*.

Na implementação do *hardware* para o Servidor Residencial, foram utilizados conceitos de eficiência energética, visto que este módulo permanecerá ligado durante longos períodos de tempo e mobilidade deste módulo pela residência. Estes dois conceitos forma alcançados através da utilização do mini-PC *Cubieboard*.

Desta forma, a Solução proposta foi alcançada e provê acessibilidade através de um Sistema de Automação Residencial de baixo custo, fácil instalação, alta flexibilidade e grande facilidade, variedade e mobilidade na interface com usuário característica primordial para uma Solução que pretende atender deficientes físicos.

6.2 - SUGESTÕES PARA TRABALHOS FUTUROS

Como sugestão de trabalhos futuros, pode-se implementar um modelo que funcione em mais de uma fase elétrica, permitindo a aplicação do trabalho em residências com alimentação bifásica e trifásica.

Implementação de Módulos de Automação que implementem a leitura de *tags* NFC, isto permitiria que o usuário necessitasse apenas de um cartão com a *tag* NFC fixada para acionar um dispositivo eletrônico, expandindo o conceito de mobilidade no projeto.

Outra sugestão é a abordagem do protótipo de transmissão de dados através da rede elétrica como uma topologia de rede identificando as camadas do modelo OSI e TCP/IP (FOROUZAN, 2008) que são abordadas no protótipo de automação.

REFERÊNCIAS

ANDROID. Android Developer. **Near Field Communication**, 2013. Disponivel em: <http://developer.android.com/guide/topics/connectivity/nfc/index.html>. Acesso em: 20 Outubro 2013.

ARDUINO. Products: ArduinoBoardUno. **Arduino Uno**, 2013. Disponivel em: <http://www.arduino.cc>. Acesso em: 20 Agosto 2013.

AURESIDE. Quem Somos: Objetivos - Sistemas de Automação Residencial. **Site da Associação Brasileira de Automação Residencial**, 2013. Disponivel em: <http://www.aureside.org.br/quemsomos/default.asp?file=objetivos.asp>. Acesso em: 15 Setembro 2013.

CARMONA, J. V. C.; PELAES, E. G. Analysis and Performance of Traffic of Voice and Video in Network Indoor PLC. **IEEE Latin America Transactions**, v. X, n. 1, p. 1268-1273, Janeiro 2012.

CAVALCANTE, F. Z. M. S. Reconhecimento de movimentos humanos para imitação e controle de um robô humanoide. **Universidade de São Paulo**, São Carlos, Outubro 2012.

CHEN, J. Fluctuation Request: A Fast Retransmission Scheme in Power Line Communication Network. **IEEE ISPLC 2008**, v. IV, p. 36-42, Abril 2008.

COULOURIS, G. et al. **Sistemas Distribuídos:** conceitos e projeto. 5ª. ed. Porto Alegre: Bookman, 2013. 1048 p.

CUBIEBOARD. An introduction to cubieboard. **Ubuntu One**, 2013. Disponivel em: <http://www.ubuntuone.com/0zdRGF22MH4kCv5UdA6RmE/>. Acesso em: 23 Setembro 2013.

DUNN, F.; PARBERRY, I. **3D math primer for graphics and game development**. [S.l.]: AK Peters Ltd, 2011.

EDUP. Products: Home Plug Ethernet. **Site da Edup Corporation**, 2013. Disponivel em: <http://www.szedup.com/show.aspx?id=1674>. Acesso em: 20 Setembro 2013.

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. **University of California**, Irvine, 2000.

FOROUZAN, B. A. **Comunicação de Dados e Redes de Computadores**. 4ª. ed. São Paulo: McGraw-Hill, 2008. 1134 p.

GODINHO, F. Noções de Acessibilidade à Web. **Site acessibilidade.net**, 9 Janeiro 2010. Disponivel em: <http://www.acessibilidade.net/web/>. Acesso em: 10 Setembro 2013.

HAYKIN, S. **Sistemas de Comunicação:** Analógicos e Digitais. 1ª. ed. São Paulo: Bookman, 2007. 668 p.

IBGE. Características gerais da população, religião e pessoas com deficiência. **Censo Demográfico 2010**, 2010. Disponivel em: <http://www.ibge.gov.br/home/estatistica/populacao/censo2010/caracteristicas_religiao_deficiencia/default_caracteristicas_religiao_deficiencia.shtm>. Acesso em: 16 Julho 2013.

KUROSE, J.. **Redes de computadores e a Internet**. 5ª. ed. São Paulo: Pearson, 2010. 614 p.

KWON, H. et al. Generalized CSMA/CA for OFDMA Systems: Protocol Design, Throughput Analysis and Implementation Issues. **IEEE Transactions on Wireless Communications**, v. VIII, n. 8, p. 4176-4187, Agosto 2009.

LAMPE, L. Bursty Impulse Noise Detection by Compressed Sensing. **IEEE ISPLC 2011**, v. VII, p. 29-34, Abril 2011.

LUTZ, M. **Learning Python**. 5ª. ed. Sebastopol: O'Reilly, 2013. 1540 p.

MICROSOFT. Kinect for Windows. **microsoft.com**, 2013. Disponivel em: <http://www.microsoft.com/enus/kinectforwindows/>. Acesso em: 12 Julho 2013.

MOESLUND, T. B.; GRANUM, E. A survey of computer vision-based human motion capture. **Computer Vision and Image Understanding**, v. LXXXI, p. 231-268, 2001.

OPENCV. OpenCV API Reference. **OpenCV Documentation**, 2013. Disponivel em: <http://docs.opencv.org/modules/core/doc/intro.html#api-concepts>. Acesso em: 10 Novembro 2013.

OPENKINECT. OpenKinect Library. **openkinect.org**, 2013. Disponivel em: <http://openkinect.org/>. Acesso em: 10 Julho 2013.

OPENNI. OpenNI Framework. **openni.org**, 2013. Disponivel em: <http://www.openni.org/>. Acesso em: 15 Julho 2013.

POPPE, R. A survey on vision-based human action recognition. **Image and Vision Computing**, v. XXVIII, p. 976-990, 2010.

RICHARDSON, L.; RUBY, S. **RESTful Serviços Web**. 1ª. ed. Rio de Janeiro: Alta Books, 2007. 336 p.

SAAD, A. L. **Acessibilidade:** guia prático para o projeto de adaptações e de novas edificações. 1ª. ed. São Paulo: Pini, 2011. 83 p.

SCHWAGER, A.; STADELMEIER, L.; ZUMKELLER, M. Potential of Broadband Power Line Home Networking. **IEEE CCNC 2005**, p. 359-363, Janeiro 2005.

TANENBAUM, A. S. J.; WETHERALL, D. **Redes de Computadores**. 5ª. ed. São Paulo: Pearson, 2012. 582 p.

TEIXEIRA, T.; DUBLON, G.; SAVVIDES, A. A Survey of Human-Sensing: Methods for Detecting Presence, Count, Location, Track, and Identity. **ACM Computing Surveys**, v. V, 2010.

WORLD HEALTH ORGANIZATION; THE WORLD BANK. Disabilities and rehabilitation. **World Report on Disability**, 2011. Disponivel em: <http://www.who.int/disabilities/world_report/2011/en/index.html>. Acesso em: 15 Julho 2013.

ZAKHOUR, S. B.; KANNAN, S.; GALLARDO, R. **The Java Tutorial**. 5ª. ed. Michigan: Addison-Wesley, 2013. 713 p.

ZATTAR, H.; CORRÊA, P.; CARRIJO, G. Analysis, Measurement and Evaluation of Power Line Communication Network Applied for Popular Houses. **IEEE Latin America Transactions**, v. X, n. 1, p. 1283-1288, Janeiro 2012.

APÊNDICE A – Script *Web Server* automático

Script responsável por executar o aplicativo *Web Server Tomcat* 7 como serviço no sistema operacional Linux Lubuntu 13.06 Server. Controla as dependências e prioridade deste serviço além de especificar a linha de comando para inicia-lo (*start*), pará-lo (*stop*) e reiniciá-lo (*restart*).

/etc/init.d/tomcat7

```
#!/bin/bash

### BEGIN INIT INFO
# Provides:        tomcat7
# Required-Start:  $network
# Required-Stop:   $network
# Default-Start:   2 3 4 5
# Default-Stop:    0 1 6
# Short-Description: Start/Stop Tomcat server
### END INIT INFO

PATH=/sbin:/bin:/usr/sbin:/usr/bin

start() {
 sudo -E /usr/share/apache-tomcat-7.0.42/bin/startup.sh #START
}

stop() {
 sudo -E /usr/share/apache-tomcat-7.0.42/bin/shutdown.sh #STOP
}

case $1 in
  start|stop) $1;;    #RESTART
```

APÊNDICE B – *Web Service* RESTFul: RestWebServices

O *Web Service RESTFul* é composto por duas classes principais: a *CommandDetails.java* e a *SerialComm.java*. A classe *CommandDetails.java* é a única *servlet* (classe Java usada para expandir as capacidades do servidor) do *Web Service*. Esta classe é responsável por receber uma requisição *web* especificando o recurso a ser acessado, repassar para a classe *SerialComm.java* e retornar o estado completo deste recurso.

Já a classe *SerialComm.java* é responsável por duas ações:

- configurar e iniciar a comunicação serial com o microcontrolador *Arduino* (método *connect()*);
- tratar o parâmetro enviado pela *servlet* e encaminhá-lo através da porta serial para o microcontrolador *Arduino* (método *SerialWriter()*). Caso o envio para o *Arduino* tenha sucesso o método retorna 1, caso contrário 0.

CommandDetails.java

```
package br.com.plc;

import java.util.Enumeration;

import gnu.io.CommPortIdentifier;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;

@Path("/command")
public class CommandDetails {
	@SuppressWarnings("unchecked")
	@GET
	@Path("/{param}")
	@Produces("application/xml")
	public String getMsg(@PathParam("param") String state) throws Exception {
		Enumeration<CommPortIdentifier> portList =
CommPortIdentifier.getPortIdentifiers();
		String portName = "none";
		int result = 0;
		while (portList!= null && portList.hasMoreElements()) {
			CommPortIdentifier portId = (CommPortIdentifier)
portList.nextElement();
			if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
				portName = portId.getName();
				SerialComm com  = new SerialComm();
				com.connect(portName);
				result = com.SerialWriter(state);
				break;
			}
		}
```

```
            state = "<COM><portName>" + portName + "</portName><command>" + state
+ "</command><result>" + result + "</result></COM>";
            return state;
      }
}
```

SerialComm.java

```
package br.com.plc;

import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

import java.io.IOException;
import java.io.OutputStream;

public class SerialComm
{
      public SerialPort serialPort;

    public SerialComm()
    {
        super();
    }

    void connect ( String portName ) throws Exception
    {
        CommPortIdentifier portIdentifier =
CommPortIdentifier.getPortIdentifier(portName);
        if ( portIdentifier.isCurrentlyOwned() )
        {
             System.out.println("Erro: Porta em uso");
        }
        else
        {
            CommPort commPort =
portIdentifier.open(this.getClass().getName(),1000);

            if ( commPort instanceof SerialPort )
            {
                serialPort = (SerialPort) commPort;

serialPort.setSerialPortParams(9600,SerialPort.DATABITS_8,SerialPort.STOPBITS_1,Se
rialPort.PARITY_NONE);
                serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
            }
            else
            {
             System.out.println("Erro: Apenas porta serial pode ser utilizada.");
            }
        }
    }


    public int SerialWriter(String message ) throws IOException {
      OutputStream out = serialPort.getOutputStream();
      try {
            Thread.sleep(2000);
```

```
            out.write(message.getBytes());
            Thread.sleep(100);
            out.flush();
        }
        catch ( IOException | InterruptedException e ) {
            e.printStackTrace();
            return 0;
        }
        finally {
             out.close();
             serialPort.close();
        }
        return 1;
    }
}
```

APÊNDICE C – mamba_server

Código em linguagem C responsável por receber os dados da mensagem enviada (comando e endereço) através da porta serial (função *Serial.read()*) do microcontrolador *Arduino* e enviá-los para rede PLC em *brodcast* (função *plm1_send_data()*) para que o nó, correspondente ao endereço enviado, possa executar o comando especificado na mensagem.

***mamba_server*.ino**

```
#include <arduino.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "spi.h"
#include "ports.h"
#include "plm1.h"

#define MOSI          B,3
#define MISO          B,4
#define SCK           B,5


static volatile bool uart_tx_flag = false;
static volatile bool uart_rx_flag = false;
static uint8_t uart_rx_data;

uint8_t rx_packet[PLM_RX_MAX_PACKET_SIZE];
uint8_t rx_length = 0;
uint8_t tx_packet[PLM_TX_MAX_PACKET_SIZE];
uint8_t tx_length = 0;

uint8_t incomingByte = 0;

void setup()
{
  PIN_OUTPUT(MOSI);
  PIN_OUTPUT(SCK);
  PIN_OUTPUT(nPLM_RESET);
  PIN_OUTPUT(PLM_CS);
  PIN_INPUT(MISO);

  spi_init( SPI_SPEED_F16 );

  plm1_init();


  EIFR = 0xFF;
  EICRA = _BV( ISC00 ) | _BV( ISC01 );
  EIMSK = _BV( INT0 );
  Serial.begin(9600);
}

char *data = NULL;
```

```
void
loop( void )
{

    if (Serial.available() > 0) {
            incomingByte = Serial.read();
            plm1_send_data(&incomingByte, 1);
    }

    if( (rx_length = plm1_receive(rx_packet)) )
    {
       data = (char *)rx_packet;
       data[rx_length] = '\0';

       Serial.print(data);
    }

}

SIGNAL(INT0_vect)
{
  plm1_interrupt();
}
```

APÊNDICE D – mamba_relay

Código em linguagem C responsável por receber os dados da mensagem enviada (comando e endereço) através da rede PLC (função *plm1_receive()*), verificar se seu endereço corresponde com o endereço presente na mensagem (comando *if (strcmp(data, "ENDERECO\0") == 0)*) e, caso afirmativo, o programa verifica o estado do relé (*if (relay == 0)*) e alterna seu estado (através dos comandos *digitalWrite(relayNumber,HIGH)* e *digitalWrite(relayNumber,LOW)*). Caso não seja o nó endereçado o programa desconsidera a mensagem enviada.

mamba_relay.ino

```
#include <arduino.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "spi.h"
#include "ports.h"
#include "plm1.h"

#define MOSI          B,3
#define MISO          B,4
#define SCK           B,5


static volatile bool uart_tx_flag = false;
static volatile bool uart_rx_flag = false;
static uint8_t uart_rx_data;

uint8_t rx_packet[PLM_RX_MAX_PACKET_SIZE];
uint8_t rx_length = 0;
uint8_t tx_packet[PLM_TX_MAX_PACKET_SIZE];
uint8_t tx_length = 0;

uint8_t incomingByte = 0;

static int relayNumber = 4;

void setup()
{
  PIN_OUTPUT(MOSI);
  PIN_OUTPUT(SCK);
  PIN_OUTPUT(nPLM_RESET);
  PIN_OUTPUT(PLM_CS);
  PIN_INPUT(MISO);

  spi_init( SPI_SPEED_F16 );

  plm1_init();
```

```
  EIFR = 0xFF;
  EICRA = _BV( ISC00 ) | _BV( ISC01 );
  EIMSK = _BV( INT0 );

  Serial.begin(9600);

  pinMode(relayNumber, OUTPUT);
}
char *data = NULL;
int relay = 0;
void
loop( void )
{

    if (Serial.available() > 0) {
            incomingByte = Serial.read();

        plm1_send_data(&incomingByte, 1);
    }

    if( (rx_length = plm1_receive(rx_packet)) )
    {
       data = (char *)rx_packet;
       data[rx_length] = '\0';

       Serial.print(data);

       if (strcmp(data, "ENDERECO\0") == 0){
         if (relay == 0) {
           digitalWrite(relayNumber,HIGH);
           relay = 1;
         }
         else {
           digitalWrite(relayNumber,LOW);
           relay = 0;
         }
       }
    }

}

SIGNAL(INT0_vect)
{
  plm1_interrupt();
}
```

APÊNDICE E – Aplicativo *KinectClient* Java

Código em linguagem Java responsável por receber os dados do dispositivo *Kinect* (através dos objetos *Context context*, *SessionManager sessionMan* e *PositionInfo pi*). Estabelecer a identificação de gestos da mão (através do controle de eventos dos métodos: *initPushDetector()* – gesto de empurrar e *initSteadyDetector()* – gesto de parar a mão). Ao identificar qualquer um dos dois eventos o programa consome o serviço *web* (através da chamada do método *callWebService()*) para acionar os dispositivos eletrônicos encontrados na rede PLC.

Main.java

```
package br.com.kinect;

import java.io.IOException;
import java.io.InputStream;

import org.OpenNI.*;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.protocol.BasicHttpContext;
import org.apache.http.protocol.HttpContext;

import com.primesense.NITE.*;

import br.com.kinect.nite.PositionInfo;

@SuppressWarnings("deprecation")
public class Main
{
  private Context context;
  private SessionManager sessionMan;

  private boolean isRunning = true;
  private PositionInfo pi = null;


  public Main()
  {
    try {
      configOpenNI();
      configNITE();

      System.out.println();
      while (isRunning) {
        context.waitAnyUpdateAll();
        sessionMan.update(context);
      }
      context.release();
    }
```

```
  catch (GeneralException e) {
    e.printStackTrace();
  }
}



private void configOpenNI()
{
  try {
    context = new Context();

    License licence = new License("PrimeSense", "0KOIk2JeIBYClPWVnMoRKn5cdY4=");
    context.addLicense(licence);

    HandsGenerator handsGen = HandsGenerator.create(context);    // OpenNI
    handsGen.SetSmoothing(0.1f);
    setHandEvents(handsGen);

    GestureGenerator gestureGen = GestureGenerator.create(context);
    setGestureEvents(gestureGen);

    context.startGeneratingAll();
    System.out.println("Started context generating...");
  }
  catch (GeneralException e) {
    e.printStackTrace();
    System.exit(1);
  }
}



private void configNITE()
{
  try {
    sessionMan = new SessionManager(context, "Click", "RaiseHand");   // NITE
    setSessionEvents(sessionMan);

    PushDetector pd = initPushDetector();
    sessionMan.addListener(pd); //LÂMPADA

    SteadyDetector sdd  = initSteadyDetector();
    sessionMan.addListener(sdd); //TOMADA
  }
  catch (GeneralException e) {
    e.printStackTrace();
    System.exit(1);
  }
}

private void setHandEvents(HandsGenerator handsGen)
{
  try {
    handsGen.getHandCreateEvent().addObserver(
                        new IObserver<ActiveHandEventArgs>() {
      public void update(IObservable<ActiveHandEventArgs> observable,
                         ActiveHandEventArgs args)
      {
        int id = args.getId();
        Point3D pt = args.getPosition();
```

```
          float time = args.getTime();
          System.out.printf("Hand %d located at (%.0f, %.0f, %.0f), at %.0f
secs\n",
              id, pt.getX(), pt.getY(), pt.getZ(), time);
        }
      });

      handsGen.getHandDestroyEvent().addObserver(
                          new IObserver<InactiveHandEventArgs>() {
        public void update(IObservable<InactiveHandEventArgs> observable,
                           InactiveHandEventArgs args)
        {
          int id = args.getId();
          float time = args.getTime();
          System.out.printf("Hand %d destroyed at %.0f secs \n", id, time);
        }
      });
    }
    catch (StatusException e) {
      e.printStackTrace();
    }
  }



  private void setGestureEvents(GestureGenerator gestureGen)
  {
    try {
      gestureGen.getGestureRecognizedEvent().addObserver(
                          new IObserver<GestureRecognizedEventArgs>() {
        public void update(IObservable<GestureRecognizedEventArgs> observable,
                           GestureRecognizedEventArgs args)
        {
          String gestureName = args.getGesture();
          Point3D idPt = args.getIdPosition();
          Point3D endPt = args.getEndPosition();
          System.out.printf("Gesture \"%s\" recognized at (%.0f, %.0f, %.0f);
ended at (%.0f, %.0f, %.0f)\n",
              gestureName, idPt.getX(), idPt.getY(), idPt.getZ(),
                           endPt.getX(), endPt.getY(), endPt.getZ() );
        }
      });
    }
    catch (StatusException e) {
      e.printStackTrace();
    }
  }

  private void setSessionEvents(SessionManager sessionMan)
  {
    try {
      sessionMan.getSessionFocusProgressEvent().addObserver(
                          new IObserver<StringPointValueEventArgs>() {
        public void update(IObservable<StringPointValueEventArgs> observable,
                           StringPointValueEventArgs args)
        {
          Point3D focusPt = args.getPoint();
          float progress = args.getValue();
          String focusName = args.getName();
          System.out.printf("Session focused at (%.0f, %.0f, %.0f) on %s [progress
%.2f]\n",
```

```
                focusPt.getX(), focusPt.getY(), focusPt.getZ(), focusName,
progress);
          }
        });

        sessionMan.getSessionStartEvent().addObserver( new
IObserver<PointEventArgs>() {
          public void update(IObservable<PointEventArgs> observable, PointEventArgs
args)
          {
            Point3D focusPt = args.getPoint();
            System.out.printf("Session started at (%.0f, %.0f, %.0f)\n",
                                        focusPt.getX(), focusPt.getY(),
focusPt.getZ());
          }
        });


        sessionMan.getSessionEndEvent().addObserver( new IObserver<NullEventArgs>()
{
          public void update(IObservable<NullEventArgs> observable, NullEventArgs
args)
          {
             System.out.println("Session ended");
             isRunning = false;
          }
        });
      }
      catch (StatusException e) {
        e.printStackTrace();
      }
    }

    private PushDetector initPushDetector()
    {
      PushDetector pushDetector = null;
      try {
        pushDetector = new PushDetector();

        float minVel = pushDetector.getPushImmediateMinimumVelocity();
        float duration = pushDetector.getPushImmediateDuration();
        float angleZ = pushDetector.getPushMaximumAngleBetweenImmediateAndZ();

        System.out.printf("Push settings -- min velocity: %.1f m/s; min duration:
%.1f ms; max angle to z-axis: %.1f degs \n",
                                minVel, duration, angleZ);

        pushDetector.getPushEvent().addObserver( new
IObserver<VelocityAngleEventArgs>() {
          public void update(IObservable<VelocityAngleEventArgs> observable,
                             VelocityAngleEventArgs args)
          {
            System.out.printf("Push: velocity %.1f m/s, angle %.1f degs \n",
                                                 args.getVelocity(),
args.getAngle());
            System.out.println("   " + pi);

            System.out.println(callWebService(3));
          }
        });
      }
```

```
    catch (GeneralException e) {
      e.printStackTrace();
    }
    return pushDetector;
  }

  private SteadyDetector initSteadyDetector()
  {
    SteadyDetector steadyDetector = null;
    try {
      steadyDetector = new SteadyDetector();
      System.out.println("Steady settings -- min duration: " +
                     steadyDetector.getDetectionDuration() + " ms");
      System.out.printf("                    max movement: %.3f mm\n",
                       steadyDetector.getMaxDeviationForSteady());

      steadyDetector.getSteadyEvent().addObserver( new
IObserver<IdValueEventArgs>() {
        public void update(IObservable<IdValueEventArgs> observable,
                           IdValueEventArgs args)
        {
          System.out.printf("Hand %d is steady: movement %.3f\n",
                                      args.getId(), args.getValue());
          System.out.println("  " + pi);

          System.out.println(callWebService(2));
        }
      });
    }
    catch (GeneralException e) {
      e.printStackTrace();
    }
    return steadyDetector;
  }

public static String callWebService(int command) {
        @SuppressWarnings("resource")
        HttpClient httpClient = new DefaultHttpClient();
        HttpContext localContext = new BasicHttpContext();
        HttpGet httpGet = new
HttpGet("http://192.168.1.2/RestWebServices/plc/command/" + command);
        String text = null;
        try {
            HttpResponse response = httpClient.execute(httpGet, localContext);
            HttpEntity entity = response.getEntity();
            text = getASCIIContentFromEntity(entity);
        } catch (Exception e) {
            return e.getLocalizedMessage();
        }
        return text;
  }

  protected static String getASCIIContentFromEntity(HttpEntity entity) throws
IllegalStateException, IOException {
      InputStream in = entity.getContent();
      StringBuffer out = new StringBuffer();
      int n = 1;
      while (n>0) {
            byte[] b = new byte[4096];
            n =  in.read(b);
            if (n>0) out.append(new String(b, 0, n));
```

```
        }
        return out.toString();
        }

  public static void main(String args[])
  {  new Main();  }

}
```

APÊNDICE F – Aplicativo *KinectClient* Python

Código em linguagem Python responsável por receber os dados do dispositivo *Kinect* (através do método *freenect.sync_get_video()*). Estabelecer a identificação de gestos da cabeça e dos olhos do usuário (através da função *DetectFace()*). Ao identificar qualquer um dos dois gestos, cabeça ou olhos (controlados pelas variáveis globais *evento* e *eventoOlho*), o programa consome o serviço *web* (através da método = *httplib.HTTPConnection().request()*) para acionar os dispositivos eletrônicos encontrados na rede PLC.

***detectFace*.py**

```
#!/usr/bin/python

import cv
import time
import Image
import freenect
import httplib

evento = 1
eventoOlho = 1

def DetectFace(image, faceCascade, eyeCascade):

    #image = kinect.getImage()
    #image = kinect.getDepth()

    global evento
    global eventoOlho

    min_size = (20,20)
    image_scale = 2
    haar_scale = 1.1
    min_neighbors = 3
    haar_flags = 0

    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    smallImage = cv.CreateImage(
            (
                cv.Round(image.width / image_scale),
                cv.Round(image.height / image_scale)
            ), 8 ,1)

    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)

    cv.Resize(grayscale, smallImage, cv.CV_INTER_LINEAR)
```

```
    cv.EqualizeHist(smallImage, smallImage)


    if faces:
        eventoAtual = 1
        for ((x, y, w, h), n) in faces:
            # bounding box of each face and convert it to two CvPoints
            pt1 = (int(x * image_scale), int(y * image_scale))
            pt2 = (int((x + w) * image_scale), int((y + h) * image_scale))
            cv.Rectangle(image, pt1, pt2, cv.RGB(255, 0, 0), 5, 8, 0)
    else:
        eventoAtual = 0

    if eyes:
            eventoOlhoAtual = 1
        for ((x, y, w, h), n) in eyes:
            # bounding box of each eye and convert it to two CvPoints
            pt1 = (int(x * image_scale), int(y * image_scale))
            pt2 = (int((x + w) * image_scale), int((y + h) * image_scale))
            cv.Rectangle(image, pt1, pt2, cv.RGB(0, 255, 0), 5, 8, 0)

    else:
        eventoOlhoAtual = 0

    if (evento != eventoAtual and eventoAtual == 0):
        cv.Rectangle(image, (0, 0), (20, 20), cv.RGB(0, 0, 255), 5, 8, 0)
           h1 = httplib.HTTPConnection('192.168.1.2')
        h1.request('GET', '/RestWebServices/plc/command/3')
        r1 = h1.getresponse()
        print r1.status, r1.reason
        h1.close()
    else:
        if (eventoOlho != eventoOlhoAtual and eventoOlhoAtual == 0):
            cv.Rectangle(image, (200, 0), (220, 20), cv.RGB(0, 0, 255), 5, 8, 0)
               h1 = httplib.HTTPConnection('192.168.1.2')
              h1.request('GET', '/RestWebServices/plc/command/2')
               r1 = h1.getresponse()
            print r1.status, r1.reason
            h1.close()

    evento = eventoAtual
    eventoOlho = eventoOlhoAtual

    return image

def video_cv(video):
    video = video[:, :, ::-1]  # RGB -> BGR
    image = cv.CreateImageHeader((video.shape[1], video.shape[0]),
                                 cv.IPL_DEPTH_8U,
                                 3)
    cv.SetData(image, video.tostring(),
```

```
            video.dtype.itemsize * 3 * video.shape[1])
    return image

#----------
# M A I N
#----------

while (cv.WaitKey(15)==-1):
    #img = cv.QueryFrame(capture)
    img = video_cv(freenect.sync_get_video(0)[0])
    image = DetectFace(img, haarFace, haarEyes)
```

APÊNDICE G – Aplicativo *AndroidClient*

Código em linguagem Java responsável por tratar as interações do usuário com o *touchscreen* de um *smartphone*. A interação ocorre através do clique do dedo do usuário em um menu de contexto que apresenta dois itens: Lâmpada e Tomada. Ao selecionar um dos itens (através do método *onOptionsItemSelected()*) é realizada uma requisição *web* para consumir o serviço *web* (através da sobreposição do método *doInBackground()* da classe *LongRunningGetIO*) para acionar os dispositivos eletrônicos encontrados na rede PLC.

O estado do recurso solicitado ao serviço *web* é apresentado na tela do *smartphone* (através do tratamento dos dados, provenientes da resposta do serviço *web*, implementado pela sobreposição do método *onCreateView()* da classe *ItemDetailFragment.java*).

ItemDetailActivity.java

```
package br.com.plc.androidclient;

import java.io.IOException;
import java.io.InputStream;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.protocol.BasicHttpContext;
import org.apache.http.protocol.HttpContext;

import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.NavUtils;
import android.view.MenuItem;

public class ItemDetailActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_item_detail);

        getActionBar().setDisplayHomeAsUpEnabled(true);
        if (savedInstanceState == null) {
            Bundle arguments = new Bundle();
            arguments.putString(ItemDetailFragment.ARG_ITEM_ID,

getIntent().getStringExtra(ItemDetailFragment.ARG_ITEM_ID));
            ItemDetailFragment fragment = new ItemDetailFragment();
            fragment.setArguments(arguments);
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.item_detail_container, fragment)
                    .commit();
```

```
        }
        new
LongRunningGetIO(this.getIntent().getStringExtra(ItemDetailFragment.ARG_ITE
M_ID)).execute();
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                NavUtils.navigateUpTo(this, new Intent(this,
ItemListActivity.class));
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    private class LongRunningGetIO extends AsyncTask <Void, Void, String> {
      protected String getASCIIContentFromEntity(HttpEntity entity) throws
IllegalStateException, IOException {
            InputStream in = entity.getContent();
            StringBuffer out = new StringBuffer();
            int n = 1;
            while (n>0) {
                  byte[] b = new byte[4096];
                  n =  in.read(b);
                  if (n>0) out.append(new String(b, 0, n));
            }
            return out.toString();
      }

          private String command;

          private LongRunningGetIO(String command) {
            this.command = command;
          }

          @Override
          protected String doInBackground(Void... params) {
            HttpClient httpClient = new DefaultHttpClient();
            HttpContext localContext = new BasicHttpContext();
            HttpGet httpGet = new
HttpGet("http://192.168.1.2/RestWebServices/plc/command/" + command);
            String text = null;
            try {
                  HttpResponse response = httpClient.execute(httpGet,
localContext);
                  HttpEntity entity = response.getEntity();
                  text = getASCIIContentFromEntity(entity);
            } catch (Exception e) {
                  return e.getLocalizedMessage();
            }
            return text;
          }

            protected void onPostExecute(String results) {
                  if (results!=null) {
                  Bundle arguments = new Bundle();
                  arguments.putString(ItemDetailFragment.ARG_ITEM_ID,
                              results);
                  ItemDetailFragment fragment = new ItemDetailFragment();
                  fragment.setArguments(arguments);
                  getSupportFragmentManager().beginTransaction()
```

```
                        .add(R.id.item_detail_container, fragment)
                        .commit();
                }
            }
        }
}
```

***ItemDetailFragment*.java**

```
package br.com.plc.androidclient;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import br.com.plc.androidclient.dummy.DummyContent;

public class ItemDetailFragment extends Fragment {
    public static final String ARG_ITEM_ID = "item_id";

    private DummyContent.DummyItem mItem;

    public ItemDetailFragment() {
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getArguments().containsKey(ARG_ITEM_ID)) {
            mItem =
DummyContent.ITEM_MAP.get(getArguments().getString(ARG_ITEM_ID));
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_item_detail,
container, false);
        String t1 = this.getArguments().getString(ARG_ITEM_ID);
        if (t1 != null && t1.length() > 2) {
            String txt = "Disponibilidade do Servidor: OK!\n\n";
            if (t1.indexOf("refused") != -1) {
                  txt = "Disponibilidade do Servidor: FALHOU!\n\n";
            }
            else {
                  int porta = t1.indexOf("<portName>/dev/ttyACM");
                  if (porta != -1) {
                              txt += "Envio de dados para o Arduino:
OK!\n\n";
                  }
                  else {
                        txt += "Envio de dados para o Arduino:
FALHOU!\n\n";
                  }

                  int cmd2 = t1.indexOf("<command>2");
                  int cmd3 = t1.indexOf("<command>3");
```

```
                    if (cmd2 != -1 || cmd3 != -1) {
                                txt += "Comando enviado para o endereço PLC "+
(cmd2 != -1 ? 2 : 3) +": OK!\n";
                    }
                    else {
                          txt += "Comando enviado para nenhum endereço PLC:
FALHOU!\n";
                    }
              }
              ((TextView)
rootView.findViewById(R.id.item_detail)).setText(txt);
         }
         return rootView;
     }
}
```