TRABALHO DE GRADUAÇÃO

IHM IPHONE PARA AUTOMAÇÃO DE SISTEMAS DE CLIMATIZAÇÃO PREDIAL UTILIZANDO INTERFACES WI-FI, RF E IR

Daniel de Miranda Vasconcellos Vilela

Brasília, outubro de 2012

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

IHM IPHONE PARA AUTOMAÇÃO DE SISTEMAS DE CLIMATIZAÇÃO PREDIAL UTILIZANDO INTERFACES WI-FI, RF E IR

Daniel de Miranda Vasconcellos Vilela

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Dr. Adolfo Bauchspiess, ENE/UnB _____
Orientador

Prof. Dr. Antonio Padilha Lanari Bó, ENE/UnB _____
Examinador

Prof. Dr. Eduardo Stockler Tognetti, ENE/UnB _____
Examinador

FICHA CATALOGRÁFICA

VILELA, DANIEL DE MIRANDA VASCONCELLOS
IHM IPHONE PARA AUTOMAÇÃO DE SISTEMAS DE CLIMATIZAÇÃO PREDIAL UTILIZANDO INTERFACES WI-FI, RF E IR
[Distrito Federal] 2012.
xi, 81p., 210 x 297 mm (FT/UnB, Engenheiro, Engenharia Mecatrônica, 2012).
Trabalho de Conclusão de Curso - Universidade de Brasília, Faculdade de Tecnologia.

1. iPhone	2. IHM
3. Infra-vermelho	4. Ar-condicionado
I. Mecatrônica/FT/UnB	II. IHM iPhone para Automação de Sistemas de Climatização Predial Utilizando Interfaces Wi-Fi, RF e IR

REFERÊNCIA BIBLIOGRÁFICA

VILELA, D.M.V. (2012). IHM IPHONE PARA AUTOMAÇÃO DE SISTEMAS DE CLIMATIZAÇÃO PREDIAL UTILIZANDO INTERFACES WI-FI, RF E IR, Trabalho de Conclusão de Curso em Engenharia Mecatrônica, Publicação FT.TG-nº 07/2012, Universidade de Brasília, Brasília, DF, 81p.

CESSÃO DE DIREITOS

AUTOR: Daniel de Miranda Vasconcellos Vilela e
TÍTULO: IHM IPHONE PARA AUTOMAÇÃO DE SISTEMAS DE CLIMATIZAÇÃO PREDIAL UTILIZANDO INTERFACES WI-FI, RF E IR.
GRAU: Engenheiro de Controle e Automação ANO: 2012

Daniel de Miranda Vasconcellos Vilela
SQN 211 Bloco J Apto 208
Asa Norte
CEP 70863-100 - Brasília - DF - Brasil

Dedicatória

Dedico este trabalho àqueles que sempre estiveram por perto para me ajudar nos momentos de alegria ou de tensão, fáceis ou difíceis no decorrer do curso. Aos meus pais e meus amigos, um abraço.

Daniel de Miranda Vasconcellos Vilela

Agradecimentos

Agradeço este trabalho primeiramente aos meus pais, que sempre me guiaram e me deram suporte quando precisei. Em segundo lugar agradeço aos meus amigos que sempre estiveram lá por mim. Em terceiro ao professor Adolfo por ter me orientado e ajudado nesse trabalho final culminando em minha formação. E por fim aos outros professores que me ajudaram nessa árdua jornada de cinco anos.

Daniel de Miranda Vasconcellos Vilela

RESUMO

Para um ambiente de automação predial de teste do Laboratório de Automação e Robótica (LARA) é criado um aplicativo supervisório para o smartphone iPhone, da marca Apple, que permite controlar e ver o estado de cada aparelho de ar-condicionado do laboratório partindo da visualização de sua planta. O aplicativo irá enviar mensagens via Wi-Fi para um servidor feito em Python localizado em um computador remoto, onde esse por sua vez irá repassar o comando serialmente para um módulo X-Bee conectado diretamente a esse computador, onde então essa mensagem é distribuída para cada outro módulo X-Bee conectados aos aparelhos de ar-condicionado onde então é enviado o comando IR de liga ou desliga. Cada vez que um comando é enviado pelo usuário utilizando o iPhone, o servidor atualiza um relatório com as informações de data e hora do comando. O aplicativo supervisório pode enviar um comando de criação de gráfico de consumo de um dos aparelhos também, onde o servidor ao receber esse comando cria um gráfico de consumo mensal baseado no relatório criado do aparelho de ar-condicionado referente.

Palavras-chave: iPhone, IHM, Infra-vermelho, Ar-condicionado.

ABSTRACT

For a test environment of building automation of Automation and Robotics Laboratory (LARA), is created a supervisory application based on the smartphone iPhone, Apple-branded, which allows an user to control and view the status of each air-conditioning device on the laboratory based on the laboratory‘s plan. The application will send messages via Wi-Fi to a server created in Python located on a remote computer, where this in turn will pass the command serially to an X-Bee module directly connected to that computer, where that message is then distributed to every other X-Bee module connected to the air-conditioner which is then sent the IR command that turns the device on or off. Each time a command is sent by the user using the iPhone, the server updates a report with information about date and time of the command. The supervisory application can send a command to create an energy consumption graphic also where when the server receives this command it creates a graph of monthly consumption based on the air-conditioning reports created before.

Keywords: iPhone, HMI, Infrared, Air-Conditioner.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

A/D	Analógico/Digital
AC	Ar-Condicionado
API	Application Programming Interface
DSL	Domain Specific Language
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
I/O	Input/Output
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
IP	Internet Protocol
IR	Infra-Red (Infra-Vermelho)
IRC	Internet Relay Chat
LARA	Laboratório de Automação e Robótica
LED	Light Emitting Diode
LCD	Liquid Crystal Display
LPRF	Low Power Radio Frequency
LSB	Least Significant Bit
MSB	Most Significant Bit
MVC	Model-View-Controller
PC	Personal Computer
PWM	Pulse Width Modulation
PAN ID	Personal Area Network Identification
RAM	Random Access Memory
RF	Rádio Frequência
RX	Receptor
SSH	Secure Shell
TCP	Transmission Control Protocol
TX	Transmissor
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UnB	Universidade de Brasília
USB	Universal Serial Bus

1 INTRODUÇÃO

Este capítulo apresenta a principal motivação do trabalho. Os objetivos são aqui apresentados, assim como a descrição do projeto e a forma como o manuscrito é apresentado.

1.1 CONTEXTUALIZAÇÃO

1.1.1 Automação Predial

O conceito de automação surgiu na indústria com o objetivo de substituir a mão-de-obra humana por máquinas e sistemas de controle. O papel destas máquinas e sistemas era supervisionar e otimizar o controle dos processos, visando aumentar a produtividade e a qualidade dos produtos. A partir dos anos 80, a automação passou a ser utilizada em prédios e residências, proporcionando diversos benefícios, tais como: conforto pessoal, segurança e uso racional de energia. Com a automação predial surgiu também a idéia de edifício inteligente. De acordo com o *Intelligent Buildings Institute* (IBI), um edifício inteligente é aquele que oferece um ambiente produtivo e que é economicamente racional, através de otimizações dos seus quatro elementos básicos - estrutura, sistemas, serviços e gestão - e das relações entre eles. Os edifícios inteligentes ajudam os seus proprietários, gestores e ocupantes a atingir os seus objetivos sob as perspectivas de custo, conforto, adequação, segurança, flexibilidade e valor comercial [21].

Pode-se afirmar que um edifício inteligente é aquele concebido e construído de forma a oferecer uma grande flexibilidade de utilização, dispondo da capacidade de evoluir, de se adaptar às necessidades das organizações e de oferecer, a cada momento, o suporte mais adequado à sua atividade. Por outro lado, deve possuir sistemas de automação, de computação e de comunicações que possibilitem, de um modo integrado e coerente, gerir de forma eficaz os recursos disponíveis, objetivando aumentos de produtividade, redução de consumo e oferecendo elevado grau de conforto e segurança aos indivíduos que nele trabalham [22].

Em sistemas de climatização prediais, conforto é um dos principais objetivos. Considerando isso, automações que sejam implementadas para um edifício inteligente devem permitir flexibilidade e possíveis aprimoramentos futuros, além de racionalização do uso da energia.

Como visto em [19] para implementar sistemas de automação em edifícios em construção pode-se utilizar cabeamento estruturado, porém cabear um edifício após sua construção tem um custo elevado. Por isso, nesses casos é recomendado utilizar uma rede sem fio para a comunicação entre os dispositivos desse sistema.

1.1.2 *Smartphones*

Os sistemas de automação residencial presentes atualmente no mercado oferecem como interfaces de controle páginas Web que podem ser acessadas de qualquer computador pessoal. Com a disseminação dos telefones celulares em todo mundo, tornou-se possıvel que eles também tivessem acesso a essas páginas

Figura 1.1: iPhone 4S.

Web, porém, com uma interface mais simples por conta das limitações de hardware e software desses telefones. O surgimento dos *smartphones* possibilitou uma melhor utilização de dispositivos móveis como interfaces de controle para sistemas de automação residencial. Esses telefones possuem recursos como maior quantidade de memória e poder de processamento, aliados a sistemas operacionais mais sofisticados, se comparados aos celulares comuns [20].

Atualmente, os *smartphones* vem sendo bastante utilizados para comunicações com dispositivos externos, hardwares projetados para prover praticidade nas atividades diárias e rotineiras. Com apenas um toque pode-se ligar ou desligar luzes, criar uma cena em uma casa, começar um filme ou ligar um aparelho de ar-condicionado.

Cada dia o mercado cria novas necessidades e faz-se necessário o desenvolvimento de novas tecnologias para manter o mercado. Além disso, essas tecnologias fornecem também subsídios para aspectos mais relevantes como racionalização de energia e segurança.

Este projeto entra então nessa categoria. Fornece subsídios para racionalização de energia controlando aparelhos de ar-condicionado por *smartphone*, permitindo acesso à distância pelo usuário, sendo bem prático do ponto de vista do usuário, pouco invasivo ao aparelho e bastante flexível do ponto de vista técnico e de projeto, devido a forma que foi feito o controle do aparelho de ar-condicionado.

1.2 OBJETIVOS DO PROJETO

O projeto tem como objetivos genéricos criar um supervisório simplificado em *smartphone* para a rede de aparelhos de climatização e fazer a monitoração do consumo de energia do laboratório.

De forma mais específica, utilizando a rede de três aparelhos de ar-condicionado do LARA, o projeto objetiva utilizar interfaces RF e IR juntamente com o aplicativo supervisório para iPhone para controlar os aparelhos remotamente e manter um relatório de controle.

Além disso, o aplicativo utiliza um servidor em um computador remoto para auxiliar em sua tarefa. O iPhone juntamente com este servidor são capazes de monitorar o consumo dos aparelhos de ar-condicionado do LARA com relatórios e gráficos de consumo de energia.

1.3 JUSTIFICATIVA

A intenção deste projeto é criar um sistema de automação para uma rede de aparelhos de ar-condicionado onde fossem utilizadas tecnologias mais recentes e menos invasivas do ponto de vista de implementação, assim como permitissem maiores flexibilidades para implementações futuras. Além disso, a implementação deve permitir visualização da utilização dos aparelhos de ar-condicionado para poder implementar políticas de racionalização de energia.

Diversos trabalhos feitos no LARA já criaram dispositivos capazes de controlar os aparelhos de ar-condicionado do laboratório. Porém, a maioria deles se baseiam em usar um tiristor de forma invasiva nos fios do aparelho. Além disso, o controle do aparelho acaba por ser um liga-desliga, sem controle efetivo da temperatura pelo ar-condicionado e possivelmente diminuindo a expectativa de vida do aparelho se mal implementado.

A forma proposta agora é utilizar um transmissor infra-vermelho para esse controle, reutilizando os comandos dos próprio controle remoto do aparelho. Claramente é uma forma não invasiva, onde além de reaproveitar a malha de controle do próprio aparelho de ar-condicionado, permite maiores ajustes (os mesmos que o controle remoto possui). Além disso permite uma grande flexibilidade, reaproveitando a implementação feita para um aparelho, para vários.

Outro justificativa para o projeto é a forma de envio dos comandos ao aparelho de ar-condicionado pelo usuário. Aqui é implementado o controle via smartphone, especificamente, o iPhone, aproveitando sua grande quantidade de vendas atingindo então grande número de usuários e o Wi-Fi existente no laboratório LARA.

Os relatórios emitidos pelo projeto juntamente com os gráficos de consumo criados fornecem subsídios para racionar energia de acordo com a utilização dos aparelhos de ar-condicionado pelo usuário. Com eles pode-se ver por exemplo se o aparelho ficou ligado uma noite inteira desnecessariamente entre diversos outros gastos com energia referentes aos aparelhos que normalmente passam despercebidos.

1.4 VISÃO GERAL DO PROJETO

O projeto consiste em construir um supervisório para *smartphone* onde será permitido ligar ou desligar os aparelhos de ar-condicionado do LARA, mostrando o estado dos aparelhos em cada cliente smartphone conectado a um servidor em um computador remoto. Além disso, o supervisório permite gerar gráficos de consumo baseados na potência nominal dos aparelhos.

A comunicação do *smartphone* com o servidor é feita via Wi-Fi. Em seguida, o servidor analisa a mensagem enviada, descobrindo se deve mudar o estado de um dos aparelhos de ar-condicionado ou criar o gráfico de consumo mensal. Caso a intenção seja a primeira, o servidor cria um relatório adicionando a um arquivo de texto qual aparelho foi selecionado, assim como seu estado e horário de mudança de estado. Se a intenção for a segunda, o servidor irá analisar o relatório com todos os horários e criar um gráfico de consumo mensal para o usuário.

Para o acionamento dos aparelhos de ar-condicionado foi escolhida a interface por infra-vermelho (IR). E a distribuição dos emissores é feita por arduinos conectados a *shields* X-Bee com módulos X-Bee Series 1 da Max-Stream, já existentes no LARA. Esses módulos consistem em comunicação via rádio-frequência (RF) de baixa potência (LPRF) bastante eficaz que permitem ainda serem atualizados para X-Bee Pro obtendo maior alcance e possibilidade de roteamento via ZigBee.

O smartphone escolhido para o projeto foi o iPhone, da marca Apple (figura 1.1), por ser de alta qualidade, confiável e ser um dos smartphones mais vendidos no mundo. Além disso, ele já é utilizado vastamente em automação predial e residencial.

Um outro recurso do projeto é a grande capacidade de melhorias no futuro. Os protocolos utilizados, os dispositivos utilizados e outras escolhas de projeto permitem uma flexibilidade para versões mais novas que possibilitam outras novas idéias para trabalhos de graduação.

A figura 1.2 apresenta um esquemático do projeto para melhor visualização.

1.5 ESTRUTURA DO MANUSCRITO

No capítulo 2 são explicados os conceitos de comunicação via infra-vermelho, ou IR, definindo como os bytes são transmitidos, os componentes utilizados e a definição de todo o protocolo e funcionamento dessa tecnologia.

Depois, no capítulo 3 é mostrado o aplicativo LARA Control utilizado como sendo a interface do projeto com o usuário e o protocolo de mensagens utilizados para comunicação com o servidor remoto.

Então no capítulo 4 descreve a implementação do servidor em um computador remoto feito em Python. É explicado também a base do programa e suas funcionalidades, como geração dos gráficos e relatórios.

Em seguida, no capítulo 5 são apresentados os módulos sem-fio, sejam eles os módulos conectados diretamente aos ACs, seja ele o módulo base de distribuição de mensagens conectado no computador remoto. O funcionamento da comunicação e o protocolo utilizado também são aqui apresentados.

Para mostrar os resultado e a análise do conteúdo do projeto apresentado nos capítulos anteriores tem-se o capítulo 6.

Por fim o capítulo 7 apresenta as conclusões do projeto como também propostas de trabalhos futuros.

Os anexos contém material complementar, como códigos utilizados.

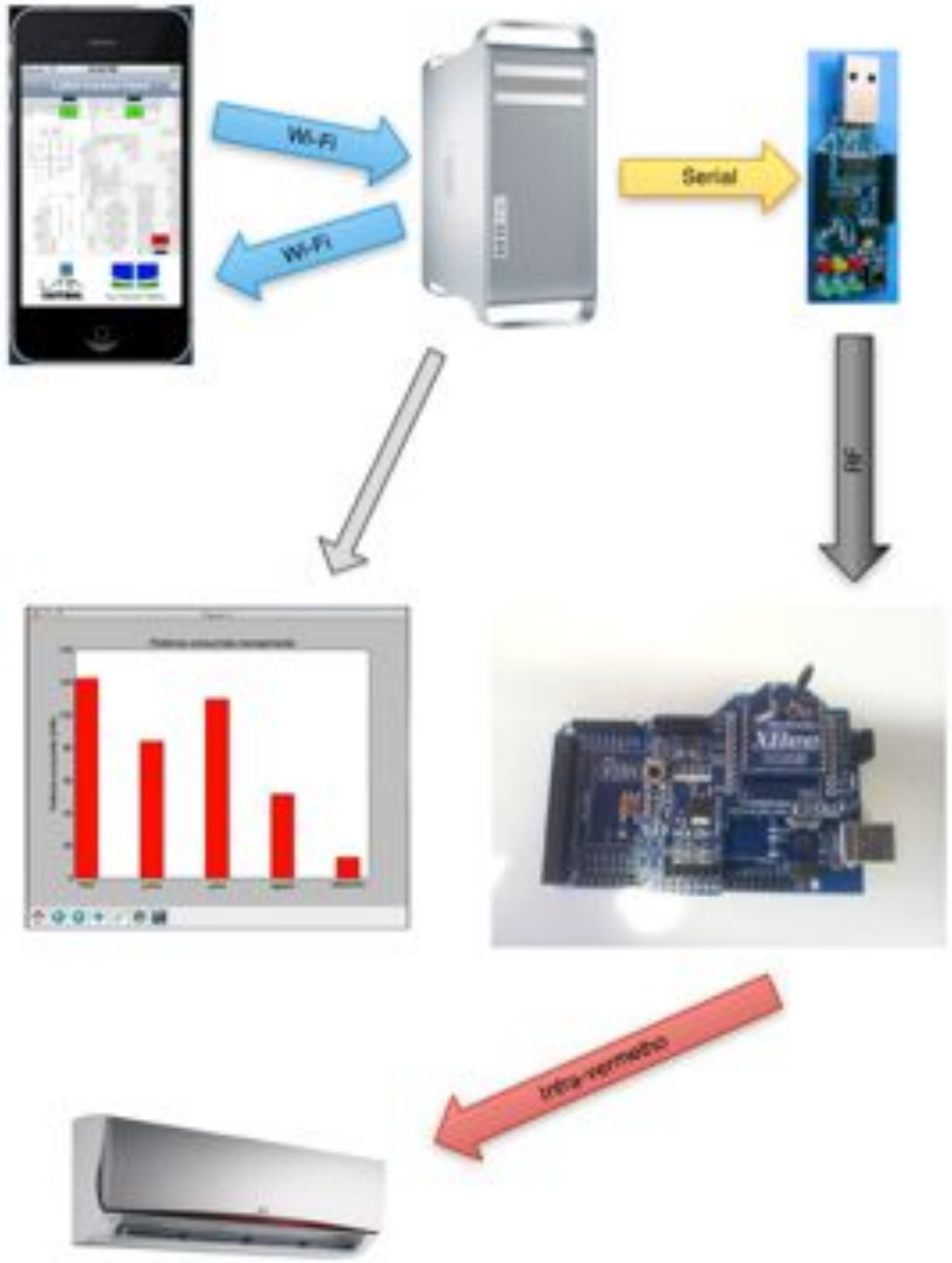


Figura 1.2: Esquemático mostrando as etapas do projeto. Aplicativo iPhone enviando mensagem via Wi-Fi para o servidor, podendo gerar gráficos de consumo ou controlar o ar-condicionado através de interfaces RF seguida de IR.

2 COMUNICAÇÃO VIA INFRA-VERMELHO

Este capítulo aborda o funcionamento do protocolo infra-vermelho, passando por fundamentais conceitos e também a forma como foi implementado no presente trabalho.

2.1 INTRODUÇÃO

Uma forma muito barata de se controlar um dispositivo à distância é via luz infra-vermelha (IR). Hoje em dia praticamente todos os equipamentos de áudio e vídeo podem ser controlados por essa tecnologia. Além disso, por ser também prático, flexível e nada invasivo ao dispositivo que se queira controlar, foi utilizada essa tecnologia nesse trabalho.

Vários trabalhos realizados no LARA já utilizaram tiristores para fazer o controle de liga e desliga dos aparelhos de ar-condicionado [23]. Outro motivo para a escolha de controle por IR é utilizar uma tecnologia diferente e que aproveite a malha de controle do próprio aparelho, assim como suas outras funções, como ajuste de temperatura e timers.

Neste capítulo serão explicados os conceitos desse tipo de comunicação, mostrando alguns exemplos de protocolos e a forma como foi implementado no projeto.

2.2 MODULAÇÃO POR PWM

2.2.1 Motivação

A luz infra-vermelha é invisível aos olhos humanos devido a seu comprimento de onda em torno de 950 nm[11], abaixo do espectro humano visível. E existem diversas fontes que emitem essa radiação, como o sol, velas, ou qualquer outro objeto ou pessoa emitindo calor, comprometendo o funcionamento do dispositivo que a utiliza. Por esse motivo, utiliza-se modulação do sinal IR para que a mensagem possa chegar ao receptor evitando-se ruídos.

2.2.2 Conceito

Modulação é o processo de variar uma ou mais propriedades de uma onda de alta frequência chamada de portadora, onde o sinal modulado contém informações a serem transmitidas [12]. O transmissor adiciona a informação na portadora de tal forma que poderá ser recuperada na outra parte através de um processo reverso chamado demodulação.

No caso de IR, utiliza-se modulação por PWM (*Pulse Width Modulation*) [13], que é a modulação envolvendo a fração efetiva do ciclo (*duty cycle*) da onda. Temos como frequência nominal 38kHz, a

Figura 2.1: Exemplo de controle remoto IR de um ar condicionado.

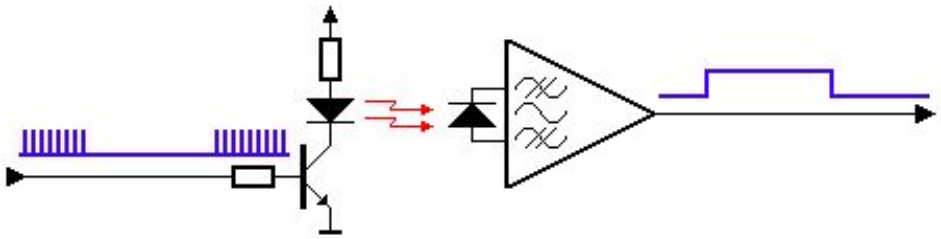


Figura 2.2: Exemplo de modulação e marcas e espaços.

mais utilizada nos controles de IR, apesar de também existirem diversos outros tipos de frequência, de acordo com cada fabricante, como 30 kHz, 33 kHz, 36 kHz, 40 kHz e até 56 kHz. O receptor IR é então configurado na frequência desejada como será explicado em 2.3.

2.2.3 Marcas e Espaços

2.2.3.1 Aspectos Gerais

Marcas e espaços são as formas efetivas de comunicação IR, ou seja, utilizado a modulação por PWM. Espaço é o sinal padrão, que é o estado em *off* do transmissor. O LED IR permanece apagado durante esse tempo. Marca é o sinal modulado, onde o LED IR é pulsado sendo ligado e desligado numa frequência particular como explicado em 2.2.2.

Apesar de um LED IR não poder ser visto acendendo aos olhos humanos, pode-se facilmente vê-lo ”piscando”por uma câmera digital, como a de um celular.

Apesar de parecer contra-intuitivo, ao receber esses sinais no receptor, marcas aparecem como um nível baixo na saída e espaços aparecem como um nível alto. A figura 2.2 ajuda a ilustrar esse fato.

Assim é possível se comunicar evitando ruídos excessivos. Tendo apenas a distância como um fator limitante.

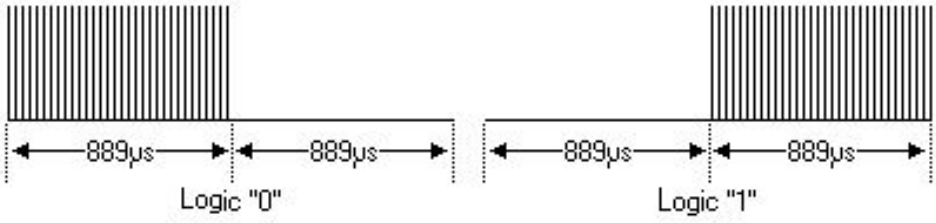


Figura 2.3: Protocolo RC-5 exemplificado.

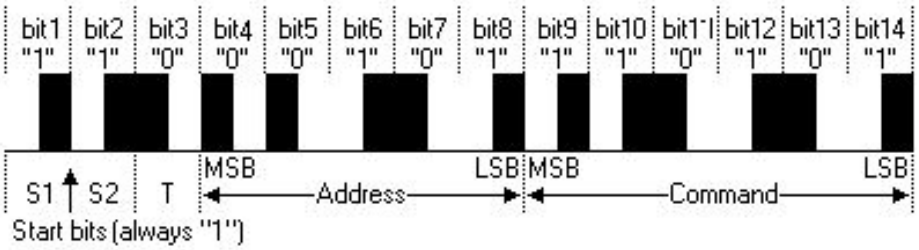


Figura 2.4: Protocolo RC-5 detalhado.

2.2.3.2 Protocolos IR

Deve-se notar que marcas e espaços não são os 0's e 1's lógicos que devem ser transmitidos. A relação entre marcas e espaços, definidas pelo protocolo usado é que determinam os sinais lógicos.

Existem diversos tipos de protocolos, alguns amplamente utilizados como os philips RC5 e RC6, o protocolo NEC e outros como o SONY SIRC, que inclusive exige que cada mensagem IR seja enviada três vezes para se obter o comando desejado. Porém alguns protocolos não são abertos, como o do ar-condicionado Springer no LARA.

Os detalhes desses protocolos genéricos não cabem no escopo desse trabalho, porém, para ajudar a obter um entendimento sobre o assunto, é válido explicar superficialmente alguns exemplos.

O RC-5 da Philips utiliza uma frequência de PWM de 36 kHz e cada 0 e 1 tem período fixo de 1,778ms, sendo sempre uma dupla começando por marca seguida de espaço ou espaço seguido de marca, cada um com 889us A figura 2.3 demonstra esse detalhe.

A sequência desses pares, com 2 bits de início representam o protocolo, onde 1 bit representa tecla de repetição, outros 5 bits para o endereço do dispositivo, e 6 bits para o comando, totalizando 14 bits e 25ms de duração da mensagem. A figura 2.4 ilustra o protocolo RC-5 um pouco mais detalhado.

Outro exemplo é o protocolo NEC que possui frequência de PWM de 38 kHz, 8 bits de endereço e 8 bits para o comando. No caso, utiliza-se o tamanho dos espaços para representar os bits, sendo sempre um bit começando com uma marca e o tamanho do espaço determinando se é 0 ou 1. A figura 2.5 ilustra essa idéia.

Uma coisa interessante desse protocolo é que ele transmite o endereço e o comando duas vezes para

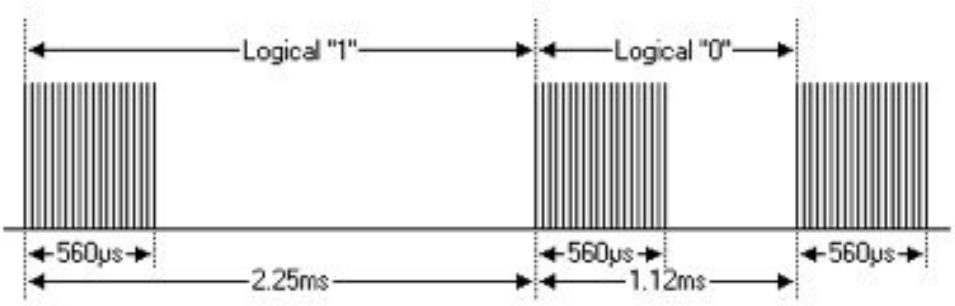


Figura 2.5: Protocolo NEC exemplificado.

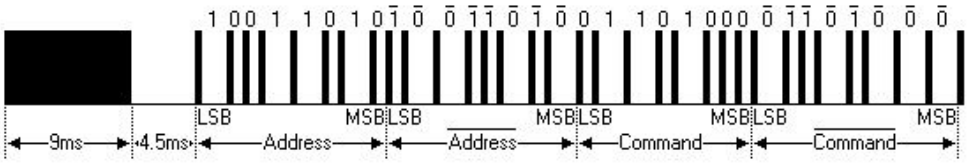


Figura 2.6: Protocolo NEC detalhado.

garantir que a mensagem seja entregue corretamente. Onde o segundo endereço e comando são invertidos. O protocolo NEC também emite no início uma marca de 9ms chamado de "*AGC burst*", que era utilizado para setar o ganho dos receptores IR antigos. A figura 2.6 ilustra esse protocolo de forma mais detalhada.

Existem alguns detalhes adicionais não mencionados aqui por fugirem ao escopo do projeto, como formas de repetição de comandos, formato NEC extendido, entre outros. Porém, agora têm-se o conceito de como funcionam os protocolos IR.

2.3 RECEPTOR IR

Os receptores IR são dispositivos que permitem a demodulação do sinal IR enviado pelo emissor. São circuitos integrados ligados a um fotodiodo que permitem transformar a modulação PWM emitidas pelo emissor e transformar as marcas em nível baixo e os espaços em nível alto. A figura 2.7 mostra o receptor IR utilizado e a figura 2.8 mostra o diagrama de blocos de um receptor IR.

O sinal IR recebido entra no lado esquerdo do diagrama. Esse sinal é então amplificado e limitado nos dois primeiros estágios. Pode-se perceber que apenas o sinal AC entra no filtro passa banda (B.P.F), o qual é configurado para operar na frequência de modulação do sinal. Então o sinal passa pelo demodulador e então pelo integrador e comparador, onde a função desses três blocos é detectar a presença da frequência de modulação. Caso esteja presente, a saída do comparador fica em nível baixo.

Existem diversos tipos de receptores, variando suas frequências entre 30 kHz e 56 kHz. O receptor utilizado deve ser de frequência próxima ao da frequência de PWM criada pelo emissor. Para esse projeto foi utilizado o receptor PNA4602 (agora já descontinuado, mas como opção equivalente existe o GP1UX311QS ou o TSOP38238) de 38 kHz (figura 2.7), que foi apenas utilizado para descobrir o proto-

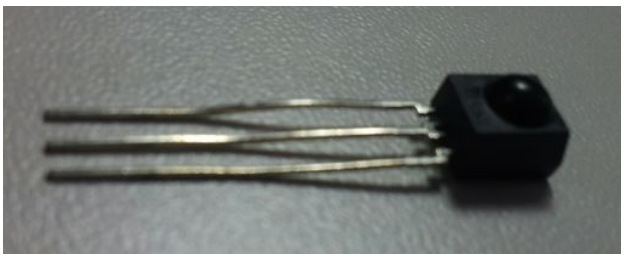


Figura 2.7: Receptor IR utilizado.

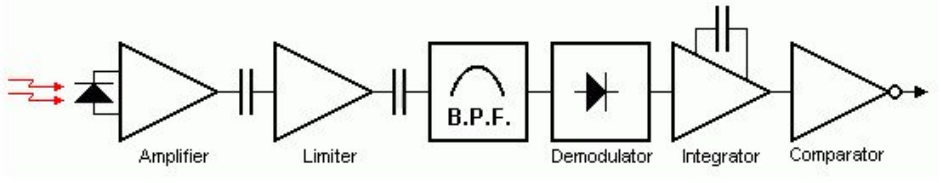


Figura 2.8: Diagrama de blocos do receptor.

colo do controle remoto do aparelho de ar-condicionado.

Na figura 2.9 no gráfico da esquerda pode-se ver as características do filtro passa-banda, onde o pico da frequência de detecção é de 38 kHz porém irá funcionar também com outras frequências próximas, na faixa de 35 kHz até 41 kHz, porém a distâncias menores apenas.

Na mesma figura no gráfico da direita pode-se perceber que a sensibilidade espectral de pico é de 940nm, porém qualquer comprimento de onda do transmissor na faixa de 850nm até 1100nm irá ser detectado, porém não tão bem como um na faixa entre 900nm e 1000nm. Eis a importância de achar um LED IR equivalente.

2.4 TRANSMISSOR IR

Os transmissores IR são LED's comuns mas que emitem luz na frequência referente ao infra-vermelho, com comprimento de onda próximos a 950nm, invisível aos olhos humanos. A figura 2.10 mostra o transmissor IR utilizado.

Uma câmera comum é capaz de observar o LED acendendo, como ilustrado na figura 2.11.

2.4.1 Protocolo IR do Ar-Condicionado Springer (LARA)

Para poder controlar então o aparelho de ar-condicionado via IR, é necessário o protocolo utilizado. Porém, por serem dispositivos de certa forma mais complexos, seus controles remotos não seguem nenhum protocolo aberto padrão, e mais que isso, cada botão não representa um código fixo.

Os controles remotos de ACs possuem um visor LCD mostrando as configurações atuais do aparelho. Pode-se notar que a comunicação IR entre controle e aparelho não possui mecanismo de *feedback*, ou seja,

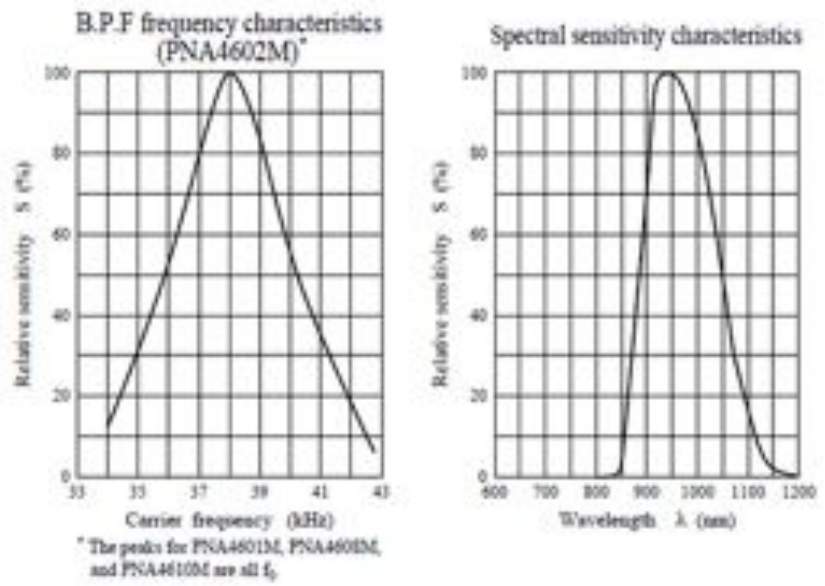


Figura 2.9: Gráfico de características do receptor.

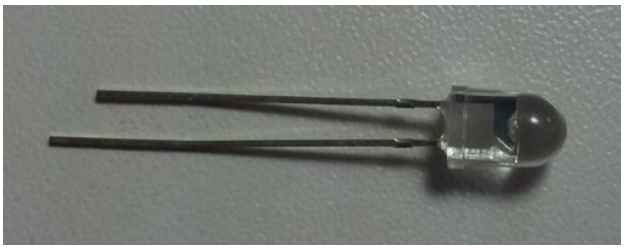


Figura 2.10: Transmissor IR utilizado.

o que é transmitido se não chegou, não chegará, mas a configuração no controle irá mudar.

Cada vez que um botão é apertado, diversos valores são enviados, como velocidade do ar, temperatura, se o aparelho está ligado ou desligado, modo de funcionamento e em alguns modelos até o horário (para timers).

Sendo assim, o funcionamento se baseia no seguinte fato. Quando algum botão é apertado no controle remoto do AC, a configuração mantida no controle muda, então é criado um código IR a partir dessas configurações e então esse comando é enviado via IR. Ao receber o comando o aparelho de ar-condicionado se atualiza com as configurações recebidas.

Como um LED IR funciona também como um fotodiodo, podemos usá-lo como receptor dos sinais de um controle remoto. Assim, com um osciloscópio é possível ver todo o sinal enviado, assim como descobrir a frequência de modulação PWM utilizada. Na próxima seção é explicado esse procedimento e como o código IR pôde ser replicado.

Figura 2.11: Transmissor IR aceso visto por uma câmera.

Figura 2.12: Arduino UNO.

2.5 IR E ARDUINO

2.5.1 Contexto e Motivação

Arduino é uma plataforma de prototipagem de eletrônicos open-source flexível baseada em hardware e software de fácil utilização e conhecido mundialmente.

Existem diversas versões do arduino, todas elas baseadas em microcontroladores da Atmel. Para o presente projeto foi utilizado o Arduino UNO (figura 2.12) [16], baseado em um microcontrolador ATMEGA328, onde já existe suporte para fonte, conexão usb, pinos de entrada e saída, botão de reset e todo o circuito necessário para o funcionamento do microcontrolador além de LEDs utilitários.

A programação é feita em uma linguagem DSL (*Domain Specific Language)* inspirada em C e C++ onde já existem diversas bibliotecas que podem ser utilizadas, sendo uma delas de IR. Essa é a principal motivação para a utilização de arduino no projeto, visto que alguns alunos do LARA já tentaram antes fazer comunicação via IR mas não obtiveram êxito. Uma biblioteca já bastante utilizada e testada é o ideal então para fazer o projeto.

A biblioteca é a IRremote, disponível gratuitamente e de código livre [1]. Permite receber e transmitir mensagens via IR seguindo diversos protocolos como por exemplo RC-5, RC-6, Sony Sirc, NEC e o código, ou conjunto de marcas e espaços crus (ou *raw*).

O protocolo IR do controle remoto do ar-condicionado Springer é fechado e não segue nenhum dos

Figura 2.13: LED IR como receptor.

padrões disponíveis na biblioteca. Então deve-se replicar o comando *raw*.

2.5.2 Descobrindo a frequência de modulação PWM

Para poder utilizar os conceitos anteriores sobre IR, é necessário descobrir a frequência de modulação do PWM.

Utilizando-se de um osciloscópio e um LED IR é possível obter essa informação, tendo como conhecimento que um LED também é um fotodiodo, ou seja, além de funcionar como um emissor, pode ser utilizado como um receptor. Basta então conectar o LED IR no osciloscópio como na figura 2.13.

Ao apertar então um botão no controle remoto do ar-condicionado, este enviará o sinal que deve ser analisado e evidenciado para o reenvio de acordo com a necessidade do usuário. O osciloscópio permite ver todo o código enviado. Ao ampliar o sinal em uma marca é possível ver a frequência PWM utilizada, como na figura 2.14.

Pode-se perceber de acordo com a figura 2.14 (c) que a frequência de modulação é de aproximadamente 38 kHz. Como visto anteriormente, o receptor IR utilizado de frequência de pico de 38 kHZ pode ser então utilizado com praticamente eficiência máxima.

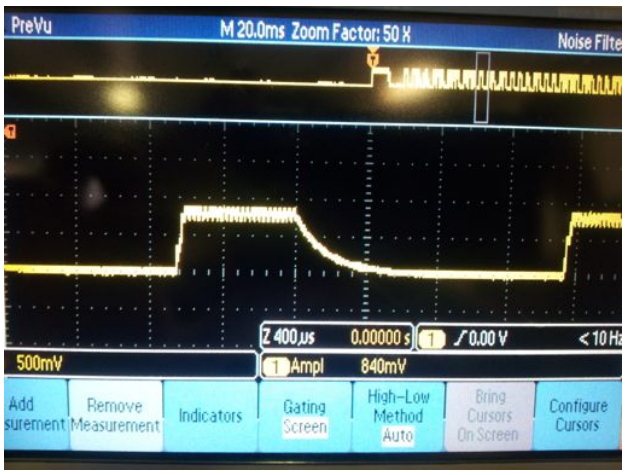
2.5.3 Utilizando a biblioteca IRremote

2.5.3.1 Hardware

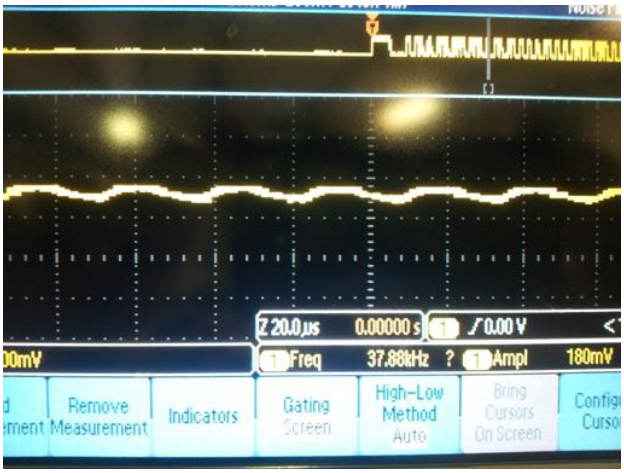
Para ligar o receptor é bem simples, bastando apenas ligar um pino em VCC, o outro em GND e o terceiro no pino de captação dos sinais, lembrando que o sinal ao passar pelo receptor é transformado em um nível baixo para as marcas e em um nível alto para os espaços.

E para o transmissor é preciso apenas conectá-lo juntamente com um resistor de 100 ohms.

(a) Sinal do osciloscópio



(b) Sinal do osciloscópio com zoom



(c) Frequência descoberta

Figura 2.14: Sinal captado no osciloscópio.

A figura 2.15 mostra os esquemáticos do hardware utilizado para o transmissor e o receptor.

Para uma versão de testes e reconhecimento do código IR, foi colocado também um botão para quando recebesse os sinais crus, pudesse apenas apertar o botão para reenviá-los. Foi utilizado também um *protoshield*, que é um ”*plug-in*”para o arduino, onde coloca-se um módulo externo do tamanho do arduino, encaixando ele nos pinos, permitindo expandir suas funções. No caso do protoshield é apenas uma protoboard facilitando fazer os contatos e adiciona portabilidade ao módulo.

A figura 2.16 mostra o esquemático do botão e a figura 2.17 mostra todo o hardware utilizado.

2.5.3.2 Visão Geral do Software

A programação é feita em um IDE Java também chamado Arduino, onde o código é feito em cima de duas funções: setup() e loop(). A primeira serve para setar registradores, iniciar serial e qualquer outra configuração necessária. A segunda mantém um loop como é esperado de um código feito para microcontroladores. A figura 2.18 mostra o IDE Arduino.

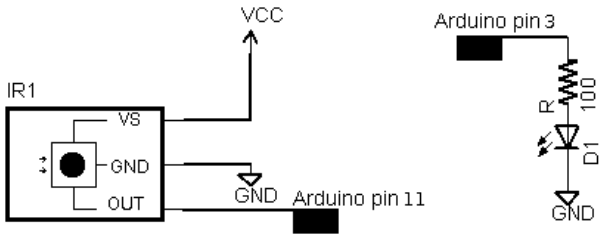


Figura 2.15: Esquemático do hardware utilizado.

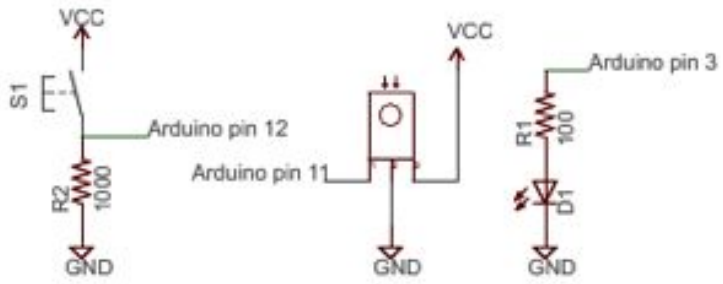


Figura 2.16: Esquemático do hardware utilizado com botão.

Apesar da biblioteca fornecer subsídios para emitir e receber sinais IR, ela é feita para códigos mais simples, então pequenas modificações tiveram que ser feitas para funcionar com o sinal do condicionador de ar, já que, por conter diversas informações referentes à configuração atual do aparelho, o código é muito maior que o normal.

Uma das modificações feitas foi aumentar o buffer de recepção (RAWBUF) para 256 bytes. Outra modificação de grande importância foi que a biblioteca normalmente implementa um tempo adicional às marcas e retira dos espaços o mesmo tempo (100us), pois parte do princípio que o receptor IR possui intrinsecamente esse erro. Porém, foi observado que o receptor utilizado só funcionava corretamente para captar o protocolo se essa definição no código fosse anulada (igual a 0), possivelmente por diferenças entre os receptores existentes no mercado.

2.5.3.3 Descobrindo o Protocolo IR

Como foi dito, o protocolo de um controle remoto de ar-condicionado é um pouco mais complexo, incluindo informações de modo de operação, velocidade do ar, temperatura e dependendo do modelo até horário.

Quando um botão é clicado, um código é criado com todas essas informações e então é enviado ao aparelho de ar-condicionado, o qual se atualiza. É por isso que esses controles remotos possuem um visor LCD mostrando as configurações atuais do ar-condicionado mesmo não mantendo um mecanismo de feedback do sinal IR.

Tendo isso em mente, como o objetivo principal era ligar e desligar o aparelho de ar-condicionado, dois códigos foram recebidos e gravados em forma de tempo das marcas e espaços (o mesmo controle remoto

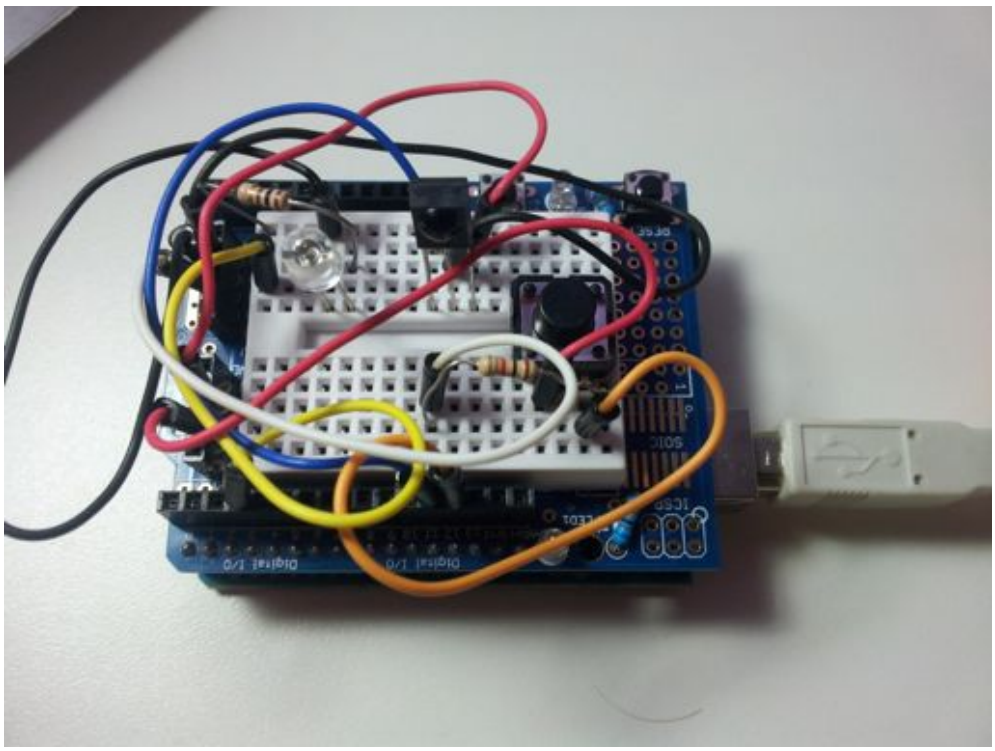


Figura 2.17: Arduino com protoshield e conexões.

Tabela 2.1: Configuração do ar-condicionado ao receber o código.

Command	Mode	Fan Speed	Temp	Clock
ON	AUTO	AUTO	21	00:00 (desativado)
OFF	AUTO	AUTO	21	00:00 (desativado)

controla todos os aparelhos de ar-condicionado do LARA), sendo a configuração de acordo com a tabela 2.1.

Os tempos gravados das marcas e espaços em microsegundos estão no anexo I.

2.5.3.4 Utilizando a IRremote

A biblioteca possui funções para mandar comandos de vários protocolos, como já foi dito, em hexadecimal, podendo ser ampliados de acordo com a necessidade do usuário. Também possui uma função para mandar o código *raw*, ou seja, de acordo com a temporização de marcas e espaços guardados em um buffer, a qual é utilizada nesse projeto.

A biblioteca do arduino IRremote vem com códigos exemplos comentados que podem ser utilizados para aprender a utilizá-la. Seguindo a lógica deles é possível fazer o código que o programador quiser. Para o caso de descobrir o código IR do controle remoto foi utilizado o exemplo IRrecord, onde permite receber o código, mostrá-lo no monitor serial em forma de tempo de marcas e espaços e então um botão

Figura 2.18: GUI de programação do Arduino.

pode reenviá-lo para ser testado.

Em seguida foi criado o código para enviar *raw* com um buffer completo com as marcas e espaços que foram conseguidos no passo anterior. Assim, combinando com o a informação enviada pelo servidor (capítulo 4) via serial, é possível distribuir aos aparelhos o código correto de controle dos aparelhos. O capítulo 5 irá explicar a distribuição dos emissores IR e como é feito o envio do código.

3 APLICAÇÃO IPHONE

Este capítulo explica o aplicativo LARA Control, utilizado como cliente para o servidor que controla o ar-condicionado, assim como sua linguagem de programação, padrões de programação e funcionamento.

3.1 INTRODUÇÃO

Como um dos smartphones mais vendidos do mundo, o iPhone é uma ótima plataforma para fazer um aplicativo de controle de dispositivos domésticos. A indústria de automação residencial atualmente utiliza vastamente tablets e smartphones para o controle de seus equipamentos. É simples, intuitivo e muitas pessoas já possuem esse tipo de tecnologia.

No iPhone existe a interface WiFi que será utilizada nesse trabalho para se comunicar com o servidor em um computador remoto. O LARA possui a sua própria rede, e normalmente qualquer laboratório também possui a sua, ou seja, as pessoas com smartphones sempre estarão conectadas, fazendo dessa interface uma ótima escolha para o objetivo desse trabalho.

O aplicativo "LARA Control"representa toda a interface com o usuário do projeto. Ele é o supervisório que se conecta com o servidor no computador remoto incluindo botões direto na planta do laboratório que permitem ligar ou desligar os aparelhos de ar-condicionado no LARA. Além disso, ele mostra o consumo no instante baseado na potência nominal dos aparelhos e envia para o servidor o momento em que os aparelhos foram ligados ou desligados, permitindo ser criado um relatório no servidor que posteriormente poderá ser utilizado para criação de um gráfico de consumo de energia mensal.

A figura 3.1 mostra o ícone do LARA Control no smartphone e a tela de *Splash Screen*, a qual aparece quando o aplicativo é iniciado.

Infelizmente para se criar um aplicativo para iPhone é necessário um computador da marca Apple, com o sistema operacional deles, Mac OS X.

3.2 PROGRAMAÇÃO DE UM APLICATIVO PARA IPHONE

3.2.1 Objective-C

A programação de um aplicativo para iPhone é toda feita numa linguagem elegante chamada Objective-C, de alto nível, orientada a objetos, que adiciona um recurso chamado transmissão de mensagens para o C. É muito bem documentada no site da Apple [2].

O Objective-C é uma camada construída sobre a linguagem C e constitui-se num superconjunto estrito de C. Ou seja, é possível compilar qualquer programa em C com um compilador Objective-C.

(a) Tela inicial com ícone

(b) Tela de *"Splash Screen"*

Figura 3.1: Tela com o ícone do LARA Control e tela de Splash Screen.

O IDE utilizado é o X-Code 4, gratuido na App Store (loja virtual de aplicativos da apple, presente no sistema operacional Mac OS X). A figura 3.2 mostra uma tela do X-Code em seu *assistant editor* com um pouco de código em objective-C.

3.2.2 Model-View-Controller (MVC)

Além da utilização de Objective-C para a criação de aplicativos, a Apple implementa também uma arquitetura, ou padrão de programação chamado Model-View-Controller, ou MVC, bastante conhecido em orientação a objetos, que isola a lógica da aplicação da interface do usuário (inserir e exibir dados), permitindo desenvolver, editar e testar separadamente cada parte.

Ele se baseia em designar para cada objeto um entre três papéis: ou ele é um *model*, ou um *view*, ou um *controller*. O padrão não define apenas o objetivo do objeto no aplicativo, mas também como os objetos comunicam entre si.

O *model* encapsula os dados de um aplicativo e define a lógica e computação que manipula e processa esses dados. Por exemplo, um *model* pode ser a parte que faz os cálculos de uma calculadora, ou os números de uma agenda. Como o LARA Control apresenta apenas uma interface para o usuário, onde o servidor guarda os dados e envia ao aplicativo quando necessário, o LARA Control não possui um arquivo *model*.

Um objeto *view* é aquele que o usuário pode ver em um aplicativo. Ele se desenha na tela e pode responder a ações de usuário, como um toque na tela. O objetivo principal de um *view* é mostrar os dados

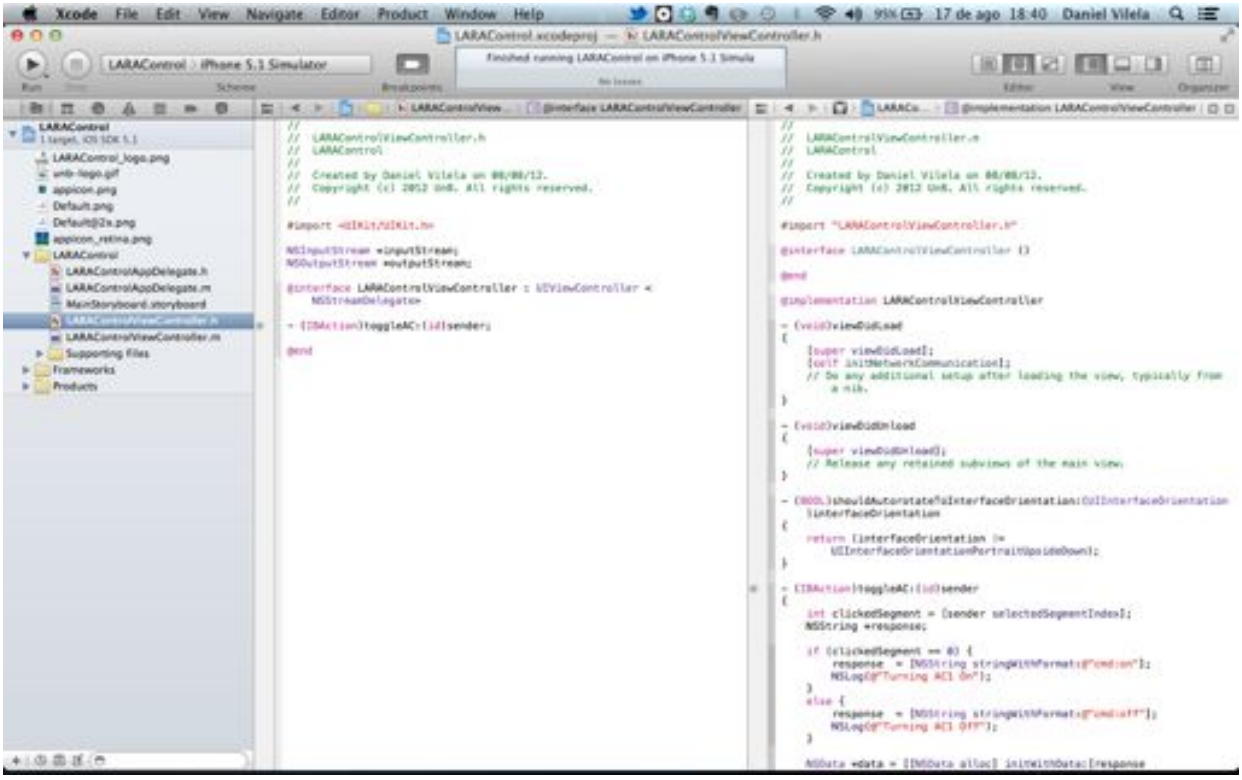


Figura 3.2: Tela do X-Code 4 e exemplo de Objective-C.

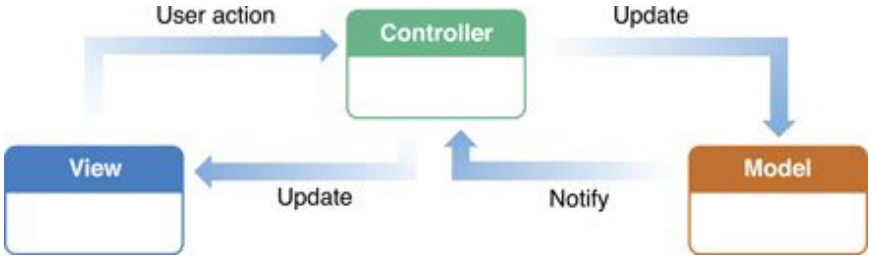


Figura 3.3: Ilustração do padrão de programação MVC.

de um *model* na tela e permitir a edição desses dados. Excluindo isso, um objeto MVC *view* normalmente é desacoplado de objetos *model*.

O *controller* atua como um intermediário entre objetos do tipo *model* e objetos do tipo *view*. É um conduíte onde os objetos view aprendem sobre mudança em objetos model e vice-versa. Um controller interpreta as ações feitas em objetos view e comunica novos dados ou dados editados para a camada model. E quando um objeto model muda, o controller comunica esses novos dados para que o view possa exibi-los.

A figura 3.3 ilustra bem esses detalhes.

Tabela 3.1: Arquivos criados automaticamente em um projeto.

LARAControlAppDelegate.h
LARAControlAppDelegate.m
MainStoryBoard.storyboard
LARAControlViewController.h
LARAControlViewController.m

3.3 INICIANDO UM PROGRAMA

Ao criar um projeto no X-Code, é automaticamente criado cinco arquivos. No caso do projeto desse trabalho chamado LARA Control, foram criados os arquivos de acordo com a tabela 3.1.

Os arquivos de extensão ".h"são os arquivos de cabeçalho enquando os de extensão ".m"representam a implementação do código em Objective-C. Já o arquivo ".storyboard"é um arquivo de edição de uma *view*, onde é visualmente editável, arrastando objetos referentes à tela.

Os arquivos "ViewController"representam o *controller* do MVC como falado anteriormente. Nele se encontram as funções de controle entre a View e o Model. No caso, como o aplicativo não possui dados, então não possui mais arquivos funcionando como *model*.

Os arquivos "AppDelegate"contém um conjunto de funções que executam funções referentes a ações do aplicativo, como se ele fechou, se iniciou ou se entrou para *background*.

O arquivo "storyboard"contém um arquivo de objetos, transições e conexões que definem a interface de um aplicativo.

O programa então pode ser criado em cima desses cinco arquivos, completando suas funções já existentes, criando o layout da tela, com botões, tabelas e diversos outros objetos já previamente prontos e criando sus próprias funções, delegando serviços, entre outros.

3.4 LARA CONTROL

3.4.1 Views do Aplicativo

3.4.1.1 View de Seleção do IP do Servidor

A interface com o usuário é apresentada em forma de views. Cada *view* é um retângulo de tamanho variável representando algo que está sendo mostrado na tela. É possível ter uma hierarquia de *views*, como em uma pilha, onde a mais no topo é a que o usuário está vendo naquele instante. Para o LARA Control foram criadas três views que serão explicadas adiante.

A primeira *view* (após a tela de Splash) é onde o usuário digita o IP do computador remoto em que o servidor rodando se encontra. Para utilizar o simulador do iOS no computador utiliza-se localhost (127.0.0.1).

Figura 3.4: Primeira *view* do aplicativo.

Ao clicar na caixa de texto o teclado aparece automaticamente para o usuário digitar o IP. Ao retornar, o teclado também sai da tela automaticamente. Em seguida o usuário clica no botão de conexão onde uma nova *view* com o supervisório mostrando a planta do LARA aparece.

A figura 3.4 ilustra a primeira *view* do aplicativo.

3.4.1.2 View do Supervisório

Em seguida aparece a *view* do supervisório. Ela consta de um botão de *info* no canto superior direito (explicado mais adiante), o título em si, a planta do LARA com os botões de acionamento dos aparelhos de AC e as logos do LARA e da UnB.

Figura 3.5: Segunda *view* do aplicativo.

A figura 3.5 ilustra a segunda *view* do aplicativo.

Ao entrar na tela do supervisório, o servidor manda informações de estados dos aparelhos para o aplicativo onde esse se atualiza automaticamente. Além disso, como pode-se ter diversos clientes iPhone conectados ao servidor, sempre que um deles muda o estado de algum ar-condicinado, essa mensagem é enviada a todos os clientes iPhone os quais se atualizam corretamente. Assim é possível manter de fato o controle dos aparelhos e a criação correta dos relatórios.

Para mudar-se o estado de um dos aparelhos basta tocar no botão de on/off na frente deles na tela do supervisório. Ao fazer isso, o aplicativo manda uma mensagem de comando para o servidor, juntamente com o horário exato da mudança de estado para que o servidor guarde o relatório em um arquivo no computador remoto.

3.4.1.3 *View* de criação de gráfico

O aplicativo consiste também em uma interface para enviar comandos de criação de gráficos de consumo mensal e também mostra o consumo instantâneo do aparelho escolhido baseado na potência nominal de cada aparelho.

Para essa terceira view basta-se apenas segurar algum dos três botões referentes a mudança de estados dos ACs e ela irá aparecer.

Ao contrário das outras duas views, essa não ocupa a tela inteira do iPhone, permitindo ainda a visualização do supervisório. Essa view foi criada de forma a aparecer uma simples caixa de texto com dois botões, e um enunciado, ou *label*, além de informar a potência nominal gasta pelos aparelhos em uso.

O primeiro botão manda um comando de criação de gráfico de consumo mensal referente àquele aparelho de ar-condicionado para o servidor, onde no computador remoto então irá aparecer tal gráfico.

O segundo apenas retira essa view da frente da view do supervisório voltando o aplicativo para seu funcionamento normal.

Acima dos botões é apresentado uma pergunta sobre o que deve ser feito para deixar o aplicativo intuitivo e também mostra a potência dos ACs, referente a quantidade deles que estão em uso no momento e a potência nominal de cada um.

A figura 3.6 ilustra o aparecimento da terceira view.

3.4.1.4 Botão de info

Existe um quarto botão no aplicativo, presente na view do supervisório que ao apertá-lo não aparece exatamente uma *view*, mas um aviso de alerta. Avisos de alerta são contido em uma classe de Objective-C que apresentam um texto qualquer em forma de alerta e botões.

No caso do botão de info utilizado, ao apertá-lo aparece para o usuário um aviso mostrando como é a utilização do LARA Control e recursos existentes. O botão no final do alerta (que na verdade é informativo, sendo alerta apenas o nome da classe) simplesmente serve para retirar o aviso e voltar a utilização normal do aplicativo supervisório.

A figura 3.7 ilustra esse aviso informativo.

3.4.2 View Controller

O aplicativo conta com diversas funcionalidades, onde cada uma delas é realizada a partir de um método. Existem métodos de controle das *views* decidindo qual delas aparece para o usuário, métodos relacionados ao acionamento dos botões de controle dos ACs, tanto apenas apertar o botão como segurá-lo por um curto espaço de tempo, métodos para descobrir o horário do acionamento dos botões para ser enviado ao servidor, entre outros.

Cada vez que um botão é apertado, uma *view* é mudada ou uma função é chamada, o *view controller*

Figura 3.6: Terceira *view* do aplicativo.

que está realizando essas ações.

Quando o aplicativo inicia, após a tela de Splash, o *view controller* chama a primeira *view* do aplicativo, onde é escolhido o endereço IP do servidor. Ao clicar no botão da tela, o *view controller*, além de chamar a *view* do supervisório, chama a função de inicialização de conexão com o servidor remoto. A partir daí, qualquer ação do usuário em cima do supervisório ou atualização do supervisório via mensagem do servidor, é função do *view controller* tratar e chamar a função correta de acordo com o solicitado.

Objective-C para programação de iPhone já implementa diversas classes e métodos que facilitam a criação de um aplicativo e comunicação com o objeto *view controller*.

Cada objeto em uma *view* é uma instância de uma classe já implementada pelo Objective-C. Então todos os botões, *views*, labels e alertas controlados pelo view controller são objetos com diversos atributos

Figura 3.7: Tela de alerta ao clicar no botão de info.

e métodos que auxiliam na criação do software.

O IDE auxilia bastante na criação do aplicativo, pois apresenta uma interface GUI de clicar e arrastar os objetos para uma tela hipotética de iPhone para fácil visualização em tempo real de como o aplicativo está ficando. Esse é o mencionado storyboard.

Além do storyboard, o X-Code implementa uma interface de ligação de conectores. Ou seja, quando um objeto é conectado a um outro objeto ou a uma função, como a função que irá ser chamada quando um botão for apertado, o IDE permite fazer essa conexão a nível de GUI e não apenas linhas de código. Essa forma de implementação além de auxiliar bastante na criação do aplicativo, permite uma produtividade grande e boa visualização das conexões entre objetos.

Os anexos II e III possuem o código feito em Objective-C para melhor entendimento do que foi expli-

Tabela 3.2: Mensagens do cliente iPhone

Comandos	**Parâmetros**						
AC1	ON OFF	sec	min	hour	day	month	year
AC2	ON OFF	sec	min	hour	day	month	year
AC3	ON OFF	sec	min	hour	day	month	year
GRAPH	AC1 AC2 AC3	00	00	00	00	00	00

cado.

3.4.3 Protocolo de mensagens iPhone - Servidor

Quando se fala de comunicação, pensa-se logo em linguagem. A forma que nos comunicamos com o computador é com as linguagens de programação. A forma que comunicamos entre nós no Brasil é a língua portuguesa. O Wi-Fi representa um ”caminho”entre o iPhone e o computador servidor, agora o conteúdo, ou seja, a linguagem, é feita pelo protocolo.

Neste projeto foi escolhido o protocolo via separação de strings de comandos e parâmetros a partir de um sinal de dois pontos (”:”), onde mais a esquerda ficaria o comando e a partir do segundo campo em diante, os parâmetros. Essa combinação de comandos e parâmetros representa a mensagem, onde não importa o tamanho de cada parâmetro e sim o conteúdo, que é uma string qualquer indicando o parâmetro ou comando. O caractere de dois pontos foi escolhido pois além de separar visualmente bem as strings, Python e Objective-C apresentam métodos nativos para separação de strings a partir de um caractere.

Como o protocolo escolhido se baseia em achar o caractere de separação, não é necessário um limite de caracteres em cada campo, ou seja, em cada string do protocolo. Porém, o número de parâmetros deve ser sempre o mesmo para que não ocorra um overflow no servidor.

Baseando-se nas funcionalidades do aplicativo, foram decididos os comandos e parâmetros válidos. Deve-se ter um comando para ligar ou desligar cada aparelho de ar-condicionado do LARA. Esse mesmo comando deve ser capaz de enviar o horário exato de acionamento do aparelho, adicionando mais alguns parâmetros. Por fim deve ser possível enviar o comando de criação de gráfico, também do aparelho especificado.

Visto isso, a tabela 3.2 mostra todo os comandos possíveis juntamente com seus parâmetros válidos.

Observando-se a tabela, percebe-se então que cada comando tem exatamente sete parâmetros, ou seja, cada mensagem possui oito strings. No caso de mandar mensagem de gráfico, são completados os parâmetros com zeros apenas para manter a padronização e não causar problemas no servidor ao receber a

mensagem.

No caso dos comando de ligar ou desligar algum AC, parâmetros referentes ao horário também são enviados. Para obter informações detalhadas sobre o uso dos ACs no laboratório, foram adicionados parâmetros que vão desde segundo até o ano em questão. Assim, pode-se obter relatórios e gráficos bem detalhados que fornecem informações realmente úteis quando fazendo decisões de racionalização.

Para o comando de criar o gráfico, é mandado apenas um parâmetro informativo (os outros são *dummy*), onde pode-se ver criado no computador remoto contendo o servidor o gráfico referente ao consumo mensal do aparelho de ar-condicionado escolhido. Maiores informações são dadas no capítulo 4.

4 SERVIDOR EM PYTHON

Este capítulo explica a utilização de python para criar o servidor. Explica também as bibliotecas utilizadas e conceitos necessários para o entendimento do código.

4.1 POR QUÊ PYTHON?

Python [3] é uma linguagem de programação de alto nível, interpretada, orientada a objetos e de tipagem dinâmica. É bastante conhecida e bem documentada, assim como funciona em todas as plataformas, o que é de extrema importância, já que a criação de um aplicativo para iphone exige um sistema operacional da apple, mas o servidor deve funcionar em qualquer computador para que o projeto possa ser prático.

Além disso, Python possui diversos recursos poderosos como módulos e *frameworks* desenvolvidos por terceiros. Nesse trabalho especificamente, o framework *twisted* para desenvolvimento de aplicações de rede foi de extrema importância para a criação do servidor, como será visto em 4.3.1. Além desse framework, outras bibliotecas foram utilizadas para criação dos gráficos, comunicação serial e lidar com datas e horários.

O IDE utilizado para a programação do código foi o TextWrangler 4.0.1, disponível apenas para mac. A interpretação do código é feita em linha de comando via terminal. Maiores informações disponíveis no site oficial do Python. A figura 4.1 ilustra o IDE utilizado com exemplo de programa em Python.

4.2 CONCEITOS UTILIZADOS NO SERVIDOR

4.2.1 Sockets

Um socket serve para transferir dados de maneira bidirecional. É utilizado em redes de computadores para comunicação entre dois programas. É uma abstração computacional aonde cada lado da comunicação possui dois elementos: um endereço IP e uma porta. O primeiro identifica um computador e o segundo é conectado a um processo. Dessa forma, é possível identificar unicamente um aplicativo ou servidor na rede de comunicação.

Existem diversos tipos de sockets, que diferem na maneira como os dados são transferidos, ou seja, por seu protocolo. Os tipos mais comuns são TCP e UDP, que são protocolos da camada de transporte, aonde o primeiro garante que os dados cheguem corretamente e é o que será utilizado nesse trabalho.

A figura 4.2 ilustra o que foi dito, mostrando cada extremo da comunicação. No caso o iPhone cliente irá se conectar a um IP escolhido já conhecido do computador remoto contendo o servidor, e eles irão se comunicar através da porta 80 que é a porta padrão de comunicação HTTP. A comunicação então é feita via internet (Wi-Fi do iPhone).

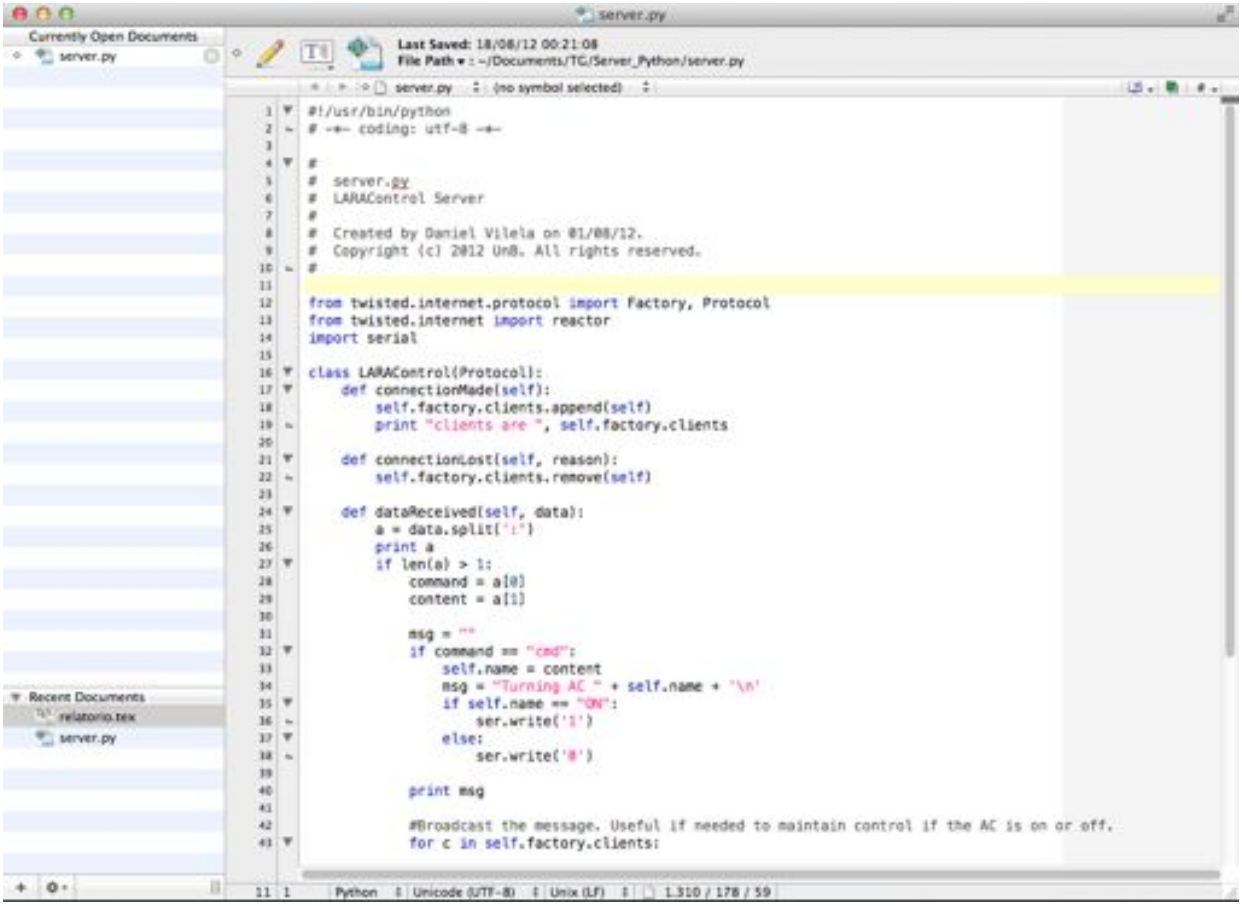


Figura 4.1: IDE TextWrangler no mac.

4.2.2 Clientes e Servidores

Um servidor é um processo que ”escuta”conexões de clientes. Cada instância de um cliente pode enviar requisições de dados para um servidor conectado e esperar pela resposta. Então, o servidor pode aceitar tal requisição, processá-la e retornar o resultado ao cliente se necessário.

O modelo cliente-servidor é largamente utilizado em diversas aplicações, onde sua arquitetura é praticamente a mesma em seus usos.

Apesar de grande parte das vezes os clientes e servidores se comunicarem via uma rede de computador em máquinas separadas, eles podem residir em um mesmo sistema.

Uma vez que uma conexão é estabelecida, o cliente e o servidor podem começar a enviar e receber dados pelo seus sockets. O socket fecha no momento que o cliente termina a comunicação ou quando o servidor para. Qualquer tentativa de uso do socket depois disso resulta em erro.

No caso desse trabalho, o código em Python aqui implementado funciona como servidor TCP, onde ele escuta por conexões de clientes, acompanha quais clientes estão conectados, os quais são identificados pelo seu socket e seu nome, e despacha eventos de acordo com o necessário.

O cliente solicita a conexão com o servidor, e então podem trocar dados. O aplicativo iPhone no caso está funcionando como cliente, podendo ter diversos clientes conectados ao servidor.

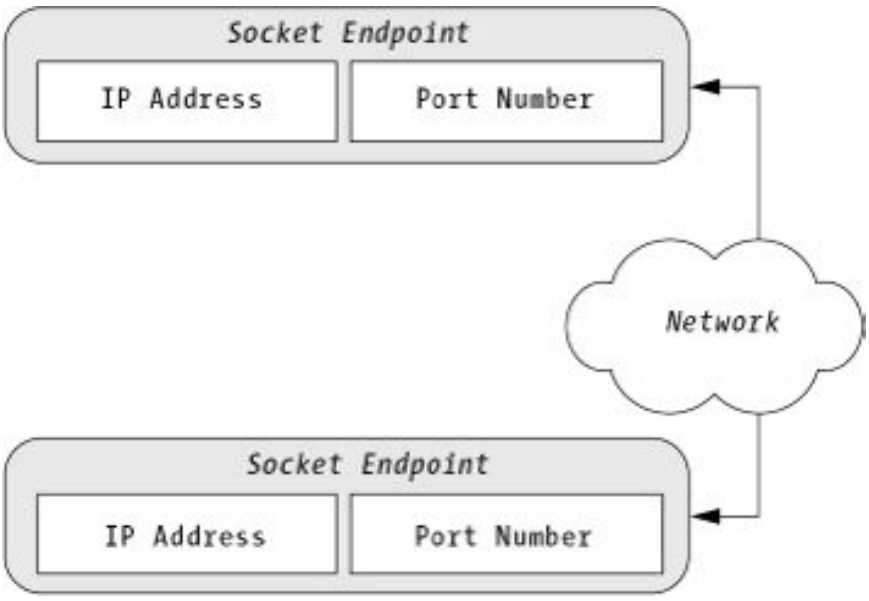


Figura 4.2: Sockets.

A figura 4.3 mostra o servidor funcionando no terminal com um cliente iPhone conectado. Pode-se perceber o servidor mostrando sua tela inicial indicando o seu funcionamento, então um cliente entra e ele mostra a lista de clientes conectados (no caso um apenas). O cliente é referenciado a um nome que é o que aparece na tela.

4.3 BIBLIOTECAS E FRAMEWORKS UTILIZADOS

4.3.1 Twisted e o padrão de programação Reactor

Twisted [4] é uma *engine* direcionada a eventos que facilita aplicações que utilizam TCP, UDP, SSH, IRC ou FTP. Ela vem com diversas classes que lidam com protocolos de rede facilitando o código e ajudando a focar especificamente na aplicação, ou seja, a troca de mensagens.

Esse framework é implementado em torno de um padrão de programação chamado *reactor pattern*. Ele começa um loop, espera por eventos e reage a esses eventos, como na figura 4.4.

Esse padrão tem como objetivo lidar com pedidos de serviços entregues concorrentemente por uma ou mais entradas e então demultiplexar os pedidos e entregá-los sincronamente para os serviços que lidam com os pedidos associados.

Um benefício claro do padrão reactor é que ele separa completamente o código da aplicação da implementação do reactor, permitindo que o os componentes da aplicação possam ser divididos em partes modulares e reutilizáveis.

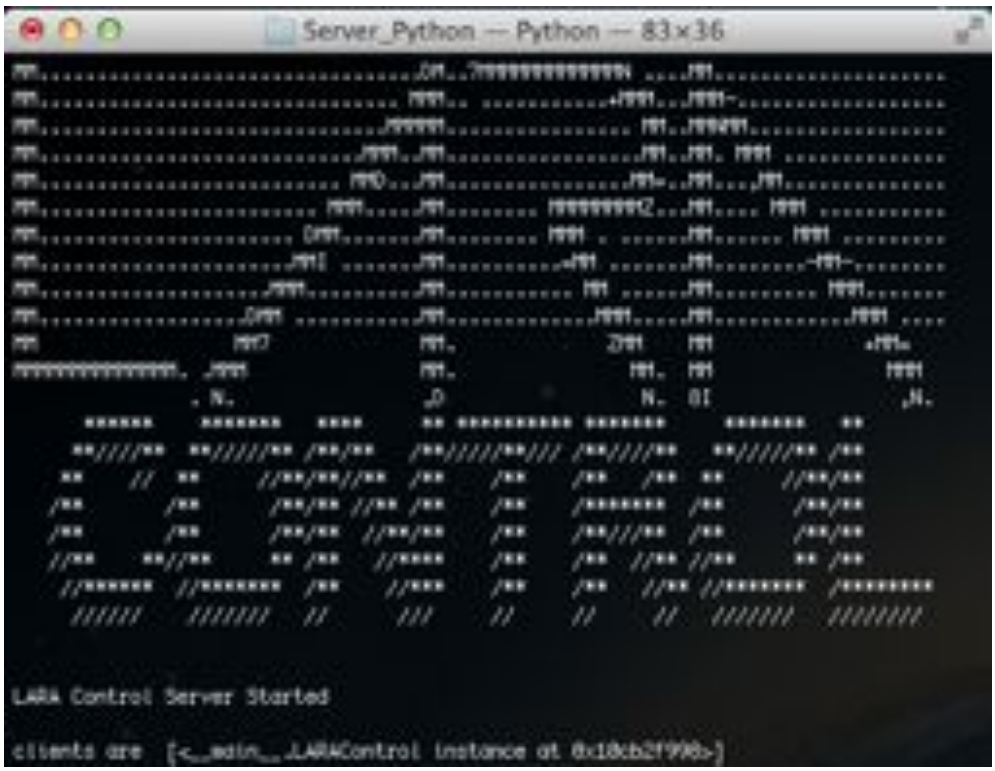


Figura 4.3: Servidor LARA Control rodando com cliente conectado.

4.3.2 Python Serial

Para poder se comunicar com o módulos externos que distribui as mensagens aos módulos X-Bee, o servidor implementa uma interface serial. Em Python existe uma biblioteca chamada serial [5] que auxilia nas funções dessa tarefa.

No caso, a interface é na realidade USB, porém no módulo de distribuição dos sinais X-Bee existe um chip FTDI que cria uma porta serial virtual (COM virtual no caso do Windows) que permite a comunicação serial como se estivesse utilizando interface RS-232 padrão.

A utilização dessa biblioteca via Python é bem simples. Sua programação é parecida com a utilização da linguagem arduino mas podendo utilizar diversas funcionalidades do Python como tipagem dinãmica dos dados, tratamento de exceções, entre outros.

Então, sempre que o usuário apertar em seu iPhone algum botão de controle de AC do aplicativo LARA Control, o servidor irá destrinchar essa mensagem e repassá-la serialmente ao módulo externo de distribuição de mensagens, onde ela será repassada via RF para os módulos arduinos contendo interface IR que finalmente irão controlar o aparelho de ar-condicinado selecionado ligando-o ou desligando-o.

O protocolo recebido pelo iPhone é diferente daquele enviado serialmente. Neste capítulo ainda será explicado o protocolo criado.

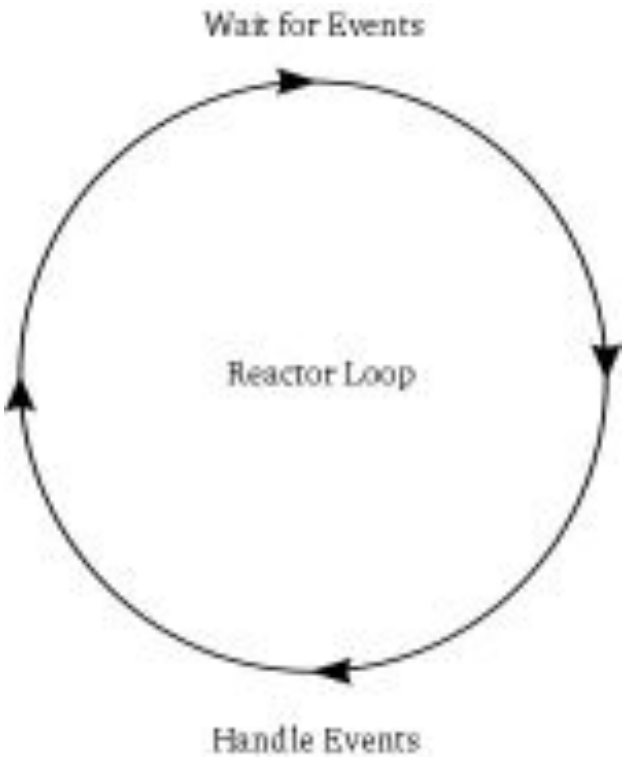


Figura 4.4: Ilustração do padrão de programação Reactor.

4.3.3 Matplotlib e NumPy

Matplotlib [6] é uma biblioteca gratuita open-source para plotar gráficos 2D na linguagem Python que utiliza outra biblioteca em conjunto para auxílio em seu uso. Essa extensão matemática utilizada é feita pela biblioteca NumPy (Numerical Python) [7]. Com essas bibliotecas é possível plotar gráficos de barra, histogramas e diversos outros tipos de gráficos com um pequeno número de linhas de código.

Ela provê uma API orientada a objetos que permite que a plotagem de gráficos sejam embarcadas em aplicações utilizando toolkits GUI genéricas, como wxPython, Qt ou GTK. Ela possui também uma utilização parecida com a do MATLAB, com interface de uso com design bem parecido.

A NumPy é uma biblioteca para Python que permite o suporte a grandes vetores e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com essas estruturas.

Além disso, como Python é uma linguagem interpretada, algoritmos matemáticos muitas vezes rodam substancialmente mais devagar do que os implementados em um código compilado equivalente. NumPy tenta então solucionar esse problema para algoritmos numéricos que operam eficientemente sobre vetores e matrizes multidimensionais, conseguindo deixar a sua eficiência perto de um código feito em C.

A combinação dessas duas bibliotecas permite obter uma funcionalidade comparável ao MATLAB devido as duas linguagens serem interpretadas e permitirem operações baseadas não só em escalares mas também vetores e matrizes. Obviamente o MATLAB implementa substancialmente mais funcionalidades, porém a interface utilizada pela matplotlib é parecida e bem familiar para quem já utiliza MATLAB. Além disso, outras bibliotecas para python podem oferecer a funcionalidade necessária dependendo da aplicação.

4.3.4 Datetime

A biblioteca datetime [8] adiciona um módulo para manipulações simples ou complexas de datas e horas em Python. Enquanto datas e horas aritméticas são suportadas, o foco da biblioteca é na eficiência de extração de atributos para formatação e manipulação de saída.

Ou seja, considerando que o servidor cria relatórios com dados de data e hora e extrai desses relatórios gráficos em barra com informações de consumo a partir do tempo em que o aparelho de ar-condicionado se manteve ligado, a biblioteca apresenta extrema ajuda para manipulação desses dados contidos nos relatórios.

Como os gráficos de consumo se baseiam em extrair o tempo que os ACs ficaram ligados, a simples utilização de certas funções dessa biblioteca permitem extrair um dado no modelo de horário convencial (como por exemplo ”12:34:26”para 12 horas, 34 minutos e 26 segundos) como também a diferença entre dois horários no formato que quiser (apenas segundos, minutos, horas, entre outros tipos).

Essa biblioteca é muito poderosa é possui diversas outras funcionalidades, porém essas não são necessárias ao projeto e portanto não sao tratadas aqui.

4.4 FUNCIONAMENTO DO SERVIDOR

4.4.1 Aspectos Gerais

O servidor apresenta uma interface com os clientes apenas, ou seja, os clientes iPhone conectados ao servidor terão total acesso às suas funcionalidades. Porém, o usuário em si não tem interface com o servidor, apenas quando um gráfico é criado ele aparecerá em uma nova janela com comandos de salvar, zoom, entre outros.

A sua funcionalidade então é baseada na troca de mensagens. Receber mensagens dos smartphones e mandar mensagens de volta para todos os aparelhos clientes com conexão com o servidor e enviar mensagens para o módulo externo de distribuição de mensagens. E cada mensagem possui um protocolo. Os protocolos utilizados em cada interface (módulo externo e iPhone) são diferentes.

O protocolo referente ao Wi-Fi e comunicação com o iPhone foi detalhado no capítulo 3. Já o protocolo serial de comunicação com o módulo X-Bee externo será detalhado no capítulo 5.

4.4.2 Recebimento de mensagens

O servidor sempre que receber uma mensagem (apenas do iPhone isso é possível), deve executar uma ação equivalente. Então, sempre que algo chegar ele irá procurar pelos caracteres ”:”contido nas mensagens, que são os divisores de strings, separando strings de comandos de strings de parâmetros. Em seguida, as strings são analisadas e o comando é executado.

Diversos comandos são utilizados no projeto. Sejam eles ligar ou desligar cada um dos aparelhos de ar-condicionado como também criar gráficos para cada um dos aparelhos.

Tabela 4.1: Tabela mostrando o comando de update do servidor para os smartphones.

comando	**AC1**	**AC2**	**AC3**
UPDATE	ON	ON	ON
	OFF	OFF	OFF

Além disso, cada comando também recebe parâmetros. Os parâmetros referentes a controle de AC acompanham seis outros parâmetros que são referentes ao horário em que o estado do ar-condicionado foi mudado. São eles: segundos, minutos, horas, dia, mês e ano, nessa ordem.

Cada vez que um comando desses é recebido, o servidor cria um relatório (ou adiciona a informação no relatório) de datas e horários que o aparelho de ar-condicionado foi ligado ou desligado. É criado separadamente um relatório para cada um dos três aparelhos, onde assim permite a criação de gráfico de consumo para cada aparelho e permitindo dessa forma um controle melhor de consumo dos ambientes do LARA.

Em relação aos parâmetros do comando de criação de gráfico, eles são do tipo *dummy*, ou seja, não trazem informação alguma podendo ser descartados (são apenas zeros). Eles são enviados de qualquer forma para manter o padrão do protocolo podendo assim o servidor manter um controle mais preciso e simples, permitindo também modificações futuras de forma fácil.

4.4.3 Envio de mensagens

4.4.3.1 Envio para Iphone

Para o projeto ser completo, o servidor precisa também poder enviar mensagens. Sejam elas via Wi-Fi para notificação aos smartphones de mudança de estado dos ACs, sejam elas de envio via serial para comunicação com o módulo externo X-Bee.

Considerando um supervisório iPhone robusto e funcional, deve ser possível saber quando o estado de um AC mudou para poder gerar o relatório de uso correto e também apresentar informação correta ao usuário que está utilizando o supervisório.

Pensando nisso foi criado o comando de *update* para ser enviado a todos os clientes iPhone conectados ao servidor para atualização do supervisório.

A tabela 4.1 mostra o comando e seus parâmetros utilizados.

O nome do comando é ”UPDATE”de fato, mandado literalmente, e cada campo do parâmetro representa respectivamente o estado de cada aparelho de ar-condicionado.

Esse comando é enviado de forma broadcast, ou seja, para todos os aparelhos conectados com o servidor. Cada aparelho então se atualiza mudando os estados se necessário de cada aparelho de ar-condicionado na planta contida no supervisório.

Tabela 4.2: Protocolo de comunicação entre servidor e módulos X-Bee.

Reservados	**AC3**	**AC2**	**AC1**	**Ação**
00	11	00	00	Liga AC3
00	10	00	00	Desliga AC3
00	00	11	00	Liga AC2
00	00	10	00	Desliga AC2
00	00	00	11	Liga AC1
00	00	00	10	Desliga AC1

4.4.3.2 Envio para Módulo Externo X-Bee

Para completar a função da mensagem enviada pelo iPhone, é necessário que ela chegue até os módulos conectados diretamente aos ACs com interface IR.

Considerando que o módulo externo de comunicação X-Bee conectado ao computador em que se encontra o servidor é apenas um replicador, ou seja, reenvia aos módulos X-Bee com conexão direta aos ACs todas as informações que chegam via serial, o protocolo escolhido deve ser um em que os arduinos com shields X-Bee aceitem.

Considerando que cada arduino não implementa uma biblioteca poderosa de comunicação serial como o Python, foi escolhido um protocolo mais simples, mais baixo nível, utilizando máscara de bits.

Então, o protocolo deve carregar as informações de estado de cada aparelho de ar-condicionado. Além disso, como será visto no capítulo 5, o módulo externo envia de modo broadcast, ou seja, envia a todos os arduinos as informações que recebe via serial. Portanto o protocolo deve ter informações de todos os ACs codificadas e os arduinos devem decodificar essa mensagem posteriormente.

O protocolo foi criado da seguinte maneira, um byte é enviado contendo todas as informações necessárias. Como são três aparelhos que devem ser controlados, três bits foram escolhidos para informação de comando, ou seja, se aquele aparelho deve ou não executar um comando. E outros três bits foram utilizados para o estados dos aparelhos, sendo que um ”1”lógico é utilizado para representa um comando de ligar o aparelho (”ON”) e de forma análoga um ”0”lógico serve para desligar o aparelho (”OFF”). A tabela 4.2 ilustra melhor o protocolo.

Só é possível uma ação ser executada de cada vez, de acordo com a tabela. Percebe-se também que um comando só será executado se tiver um bit ”1”no bit da esquerda referente a cada AC. E o segundo dirá então o futuro estado do AC.

Com esse protocolo fica fácil para o arduino saber se deve ser executado algum comando ou não. Basta apenas aplicar um *shift* de bits e uma máscara e então o resto do código pode ser o mesmo para todos os arduinos.

Tabela 4.3: Métodos principais a nível de implementação do Servidor.

connectionMade(self)
connectionLost(self, reason)
dataReceived(self, data)
message(self, message)

4.4.4 Aspectos de Implementação

O programa conta com a definição de uma classe LARAControl, que é uma subclasse da classe Protocol do framework twisted.

Nela é feita o *override* de métodos da classe Protocol para agirem de acordo com a necessidade deste servidor. São quatro esses métodos de acordo com a tabela 4.3.

O primeiro método é chamado quando um cliente se conecta, onde então o servidor os guarda em uma lista e escreve na tela todos os clientes que estão conectados.

O segundo método é quando um cliente perde a conexão ou a fecha. O servidor apenas deve retirar esse cliente de sua lista de clientes.

O terceiro método é chamado quando o servidor recebe uma mensagem de um cliente, que é de fato onde tudo que é visível para o usuário acontece. Nesse método que as strings do protocolo são separadas e então é procurado qual o comando utilizado. Então a mensagem é enviada de forma serial e o relatório ou o gráfico é criado.

Por fim o último método serve para fazer um broadcast das mensagens que chegaram de um cliente para todos os outros clientes, ou seja, é a mensagem de ”UPDATE”, assim, todos os smartphones conectados com o servidor podem se manter atualizados dos estados dos ACs.

Além dessa classe, diversas outras funções foram criadas para auxiliar na condução do programa. Mas tudo que acontece está centrado na classe principal detalhada acima. O framework começa a rodar e a dar caminho para todas as mensagens que chegam. A partir daí as funções criadas são chamadas de acordo com a necessidade.

O código implementado encontra-se no anexo IV.

5 DISTRIBUIÇÃO DE COMANDOS POR X-BEE

Este capítulo explica o papel e o funcionamento dos módulos X-Bee utilizados no projeto e como eles foram utilizados em conjunto com os arduinos.

5.1 INTRODUÇÃO

5.1.1 Arduino MEGA

Como foi dito no capítulo 2, arduino é uma plataforma de prototipagem altamente conhecida, baseada nos micro-controladores da Atmel AVR, onde também é utilizada para criar diversos projetos onde posteriormente pode-se criar uma placa externa com o microcontrolador gravado com o código em arduino. Ou seja, além de ser muito fácil e bom para prototipagens, nada impede que ele seja usado também em projetos fixos.

A figura 5.1 ilustra o Arduino MEGA [17].

Além disso, arduino permite a adição de *shields*, que são outras placas que se encaixam no arduino adicionando-lhe funcionalidades. No caso deste projeto foram utilizados *shields* X-Bee [15] que permitem a conexão do arduino com um módulo X-Bee de forma fácil. Ou seja, todo o circuito para a utilização dos módulos X-Bee já estão contidos no *shield*. A figura 5.2 ilustra o shield X-Bee e a figura 5.3 ilustra o arduino montado com o shield.

O LARA já possui dez arduinos MEGA com *shields* X-Bee. O Arduino MEGA é uma placa baseada no microcontrolador ATmega1280, possuindo 128kb de memória flash e 8kb de memória RAM, 54 pinos digitais de entrada e saída onde 14 deles podem ser utilizados como saídas PWM, possui 16 entradas analógicas, 4 UARTs, um cristal de 16 MHz, uma conexão USB, um conector de força, um conector ICSP e um botão de reset. A placa possui todo o circuito para o correto funcionamento do micro-controlador. E também é compatível com os *shields* X-Bee.

Ele foi utilizado pois possui completa compatibilidade com a biblioteca IRremote, shield X-Bee disponível no LARA e tem tudo o que é preciso para que o projeto funcione corretamente.

Para o projeto então foram utilizados três arduinos com seus respectivos X-Bee *shields*, um para cada aparelho de ar-condicionado.

5.1.2 X-Bee

X-Bee é o nome dado para a família de dispositivos de rádio da Digi International, onde os primeiros rádios foram introduzidos pela MaxStream e são baseados no padrão IEEE 802.15.4 que especifica a camada física e efetua o controle de acesso para redes sem fio pessoais de baixas taxas de transmissão. É utilizado em comunicações ponto-a-ponto e ponto-a-multiponto como pode ser visto em [14].

Existem diversas versões de módulos X-Bee, algumas só implementam o simples rádio e alguns con-

Figura 5.1: Ilustração do Arduino MEGA.

troles, outras possuem conversores A/D, pinos de I/O e algumas ainda permitem ser programados com códigos de usuários.

A versão disponível no LARA de módulos X-Bee é a 802.15.4 Series 1. Existe uma versão análoga sendo o X-Bee PRO que possui maior potência e consequentemente maior alcance. Felizmente para o tamanho do LARA os módulos existentes possuem alcance suficiente.

Existe ainda uma outra versão que deverá chegar em breve ao LARA, que são os X-Bee Series 2 que podem implementar redes do tipo *mesh* utilizando o protocolo ZigBee PRO, que possui roteamento e diversas outras funções interessantes. Também são disponíveis nos modelos PRO de maior potência.

Aplicações diversas utilizam X-Bee, como por exemplo em uma fazenda de gados, é possível instalar uma rede para monitorar sensores, instalando-os em vários locais, obtendo dessa forma informações de uma vasta área da fazenda, como nível de água dos açudes, rios e bebedouros, detecção de arames rompidos na cerca, saber a localização onde os animais permanecessem mais tempo pastando, controlar a irrigação do pasto e controlar cancelas.

A figura 5.4 ilustra um módulo X-Bee da MaxStream utilizado e os tipos de antenas existentes nos módulos do LARA. Na esquerda está o módulo com antena do tipo chip e na direita com antena do tipo chicote. A segunda antena apresenta uma vantagem em relação a primeira pois ela pode direcionar a antena para a direção onde o sinal se encontra mais forte.

5.2 X-CTU E CONFIGURAÇÃO DE HARDWARE

A Digi International fornece um programa gratuito para programação e debug dos módulos [9]. O programa permite programar os módulos, testar distância, ver pacotes recebidos, ver configuração atual, atualizar o firmware do módulo, entre outras funções. O software é disponível apenas para Windows.

A figura 5.5 ilustra o programa X-CTU. Pode-se ver pelas abas as opções possíveis, como configuração

Figura 5.2: Shield X-Bee.

de comunicação em ”PC Settings”, programação dos módulos em ”terminal”, leitura da configuração atual e atualização de firmware em ”modem configuration”e teste de distância em ”range test”.

Além da utilização da X-CTU para programação dos módulos, o shield X-Bee deve ser configurado também por *jumpers* na placa. Existem dois jumpers que podem estar na conexão USB ou X-Bee.

Quando eles estão na posição USB (escrito na placa), o micro-controlador pode ser programado tranquilamente. Já quando na posição X-Bee (escrito na placa), o micro-controlador começa a se comunicar com o shield, sendo assim a configuração utilizada no projeto. Apenas para gravar o código no arduino que os jumpers foram colocados na posição USB.

A explicação da utilização dos jumpers se encontra na conexão dos pinos DIN e DOUT do módulos X-Bee conectados aos RX e TX do micro-controlador ou ao chip FTDI presente no arduino (comunicação com o computador via USB).

A programação dos módulos via X-CTU então pode normalmente ser feita de duas formas. Uma utilizando o adaptador CON-USBBEE ROGERCOM explicado na seção do módulo de distribuição de mensagens ou utilizando-se o shield com os jumpers na posição USB, porém com o micro-controlador fora do arduino. Como o arduino MEGA possui um micro-controlador smd soldado na placa, a única forma de programar os módulos é utilizando o adaptador.

Existem outros adaptadores similares no mercado, porém o que se encontra no LARA é o mencionado acima.

Figura 5.3: Arduino MEGA montado com shield X-Bee.

5.3 PROGRAMAÇÃO DOS MÓDULOS

Para se poder utilizar os módulos, é preciso que eles sejam programados anteriormente. Isso é feito por comando AT atráves do terminal do programa X-CTU.

O módulo atende a diversos comandos diferentes, porém, para se obter uma comunicação não é necessário setar todas as configurações possíveis.

Para o projeto, as configurações necessárias foram feitas seguindo a tabela 5.1.

Todos os comandos AT são precedidos das letras ”AT”antes do comando. Então, para utilizar os

Tabela 5.1: Comandos AT utilizados na configuração dos módulos.

Comando AT	Parâmetro Base	Parâmetro End-Device	Descrição
RE	-	-	Restaura configurações de fábrica
AP	0	0	Modo transparente
CH	0C	0C	Canal de comunicação
MY	1000	1001	Endereço do módulo
DL	1001	1000	Endereço de destino
ID	1111	1111	PAN ID
WR	-	-	Grava configuração na flash

Figura 5.4: Módulo X-Bee Series 1 da MaxStream.

comandos da tabela deve-se utilizar esse prefixo na hora da configuração dos módulos.

O módulo de distribuição de mensagens é o módulo base da tabela e fica conectado ao computador com o servidor. Devido ao fato de a comunicação ser broadcast, os módulos End-Device possuem o mesmo endereço, então recebem a mesma configuração.

Os módulos X-Bee permitem funcionar em três modos diferentes, que é o comando ”AP”da tabela. No modo ”AP=0”, o X-Bee funciona em modo transparente, ou seja, tudo que o arduino enviar na serial é enviado via X-Bee e de forma reversa tudo que chegar via RF ao X-Bee chega serialmente ao micro-controlador (TX e RX do X-Bee estão ligados diretamente aos do micro-controlador).

Os outros dois modos de funcionamento são referentes ao API mode, onde pode-se enviar comandos para o X-Bee e os pacotes possuem um formato. Tudo que for diferente desse formato é descartado silenciosamento. Como para o projeto o modo transparente é o suficiente, ele foi utilizado.

Para a comunicação funcionar, o canal de comunicação e o PAN ID (Personal Area Network ID) dos módulos devem ser os mesmos, explicitados na tabela.

O comando ”MY”representa o endereço dos módulos e o comando ”DL”o endereço de destino, ou seja, tudo que o módulo enviar será enviado para o módulo com endereço ”MY”igual ao ”DL”do que enviou. Portanto os módulos End-Device conectados aos ACs possuem o mesmo endereço que é equivalente ao endereço de destino do módulo base.

Por fim executa-se o comando de gravar as configurações na flash do módulo. Caso contrário ao parar de fornecer energia ao módulo, sua configuração seria perdida.

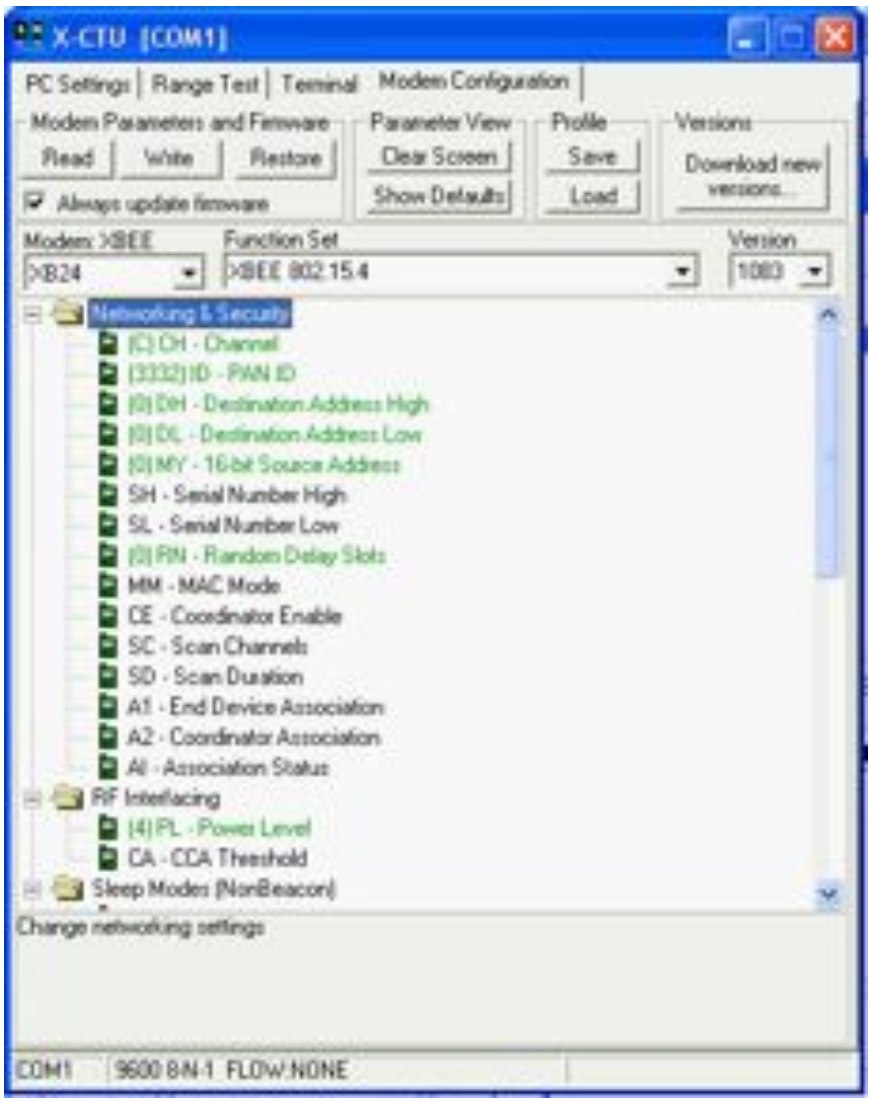


Figura 5.5: X-CTU.

5.4 MÓDULO DE DISTRIBUIÇÃO DE MENSAGENS

Para o módulo conectado ao PC que se comunica diretamente com o servidor, utilizou-se o adaptador CON-USBBEE ROGERCOM [18] do LARA o qual também é utilizado para gravar as configurações dos módulos.

Ele permite conexão direta com o PC criando uma porta serial virtual a partir da USB (FTDI Chip), portanto, pode-se enviar serialmente os comandos de controle dos ACs para a plaquinha e ela irá enviar via X-Bee para os outros módulos.

A figura 5.6 ilustra o adaptador utilizado. Pode-ser notar que o adaptador também utiliza um módulo X-Bee além de possuir botão de reset e alguns LEDs indicativos. A figura 5.7 apresenta a funcionalidade dos LEDs. A figura é auto-explicativa.

Figura 5.6: Adaptador CON-USBBEE ROGERCOM.

5.5 PROGRAMAÇÃO DO ARDUINO

A programação do arduino então considera uma simples comunicação serial. Os módulos X-Bee já possuem mecanismo de garantia de entrega de pacote transparente ao usuário onde parâmetros podem ser configurados, como número de tentativas (*retries*) que o módulo deve tentar até considerar o pacote perdido, entre outras. Mas as configurações de fábrica já funcionam muito bem.

Existe também uma biblioteca de X-Bee para arduino disponível online [10], mas ela só é realmente útil quando utilizando um X-Bee Series 2 que utiliza redes mesh ZigBee. Como o módulo disponível no projeto não possui esse recurso, a programação serial é suficiente.

Então, sempre que um comando é solicitado de um iPhone, todos os módulos receberão esse comando. Uma manipulação de bits foi utilizada para tratar esses comandos como explicado no capítulo 4.

Cada arduino possui uma diretriz ”define”que define qual AC ele está controlando. A partir dessa definição é fixado o número de bits que deve ser realizado o *shift*, o qual serve para deixar o comando referente ao AC nos dois primeiros bits. Então o byte pode ser tratado e o código igual para todos os arduinos.

Em seguida então é aplicada uma máscara que depois do shift faz um ”e”lógico que zera todos os bits do byte exceto os dois primeiros. A partir daí é só tratar esses dois bits que conterão o comando, se existente, de controle do aparelho de ar-condicionado.

Caso ele deva ser controlado, um buffer guarda os valores de marcas e espaços (capítulo 2) dos comandos de ligar e desligar do AC, para que então o LED infra-vermelho seja pulsado de acordo com o

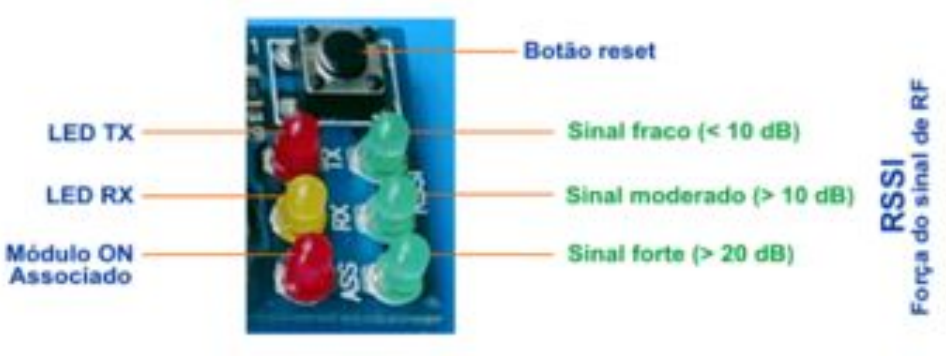


Figura 5.7: Funcionalidade dos LEDs do adaptador.

protocolo IR do aparelho.

O anexo V mostra o código criado para o arduino para melhor entendimento do que foi explicado.

5.6 HARDWARE COMPLETO

Para completar o hardware, foram utilizados alguns componentes extras de acordo com a necessidade do projeto.

Os arduinos devem possuir independência de alimentação, portanto foram utilizadas bateria 9V que são capazes de durar muito tempo considerando a baixa potência do arduino e a utilização esporádica do rádio.

Para conexão das baterias foi-se utilizado conectores compatíveis com as baterias e do tipo plug do lado do arduino. A figura 5.8 ilustra esses componentes.

Para a interface IR, deve-se ter um LED IR conectado a um resistor como especificado no capítulo 2. A conexão deles é entre o pino 9 e o GND do arduino. A figura 5.9 ilustra a solda feita para facilitar a montagem.

O hardware funcional então fica de acordo com a figura 5.10. Pode-se notar a fita isolante recobrindo a bateria e a prendendo no arduino para deixar o hardware estável. Também foi aproveitado o espaço aberto entre o arduino MEGA e o espaço não utilizado pelo shield para colocar a bateria.

Além da parte funcional, foi implementado também um suporte para o módulo ficar preso na parede ou no aparelho de ar-condicionado para diminuir um ponto de falha referente a distância do emissor IR e o receptor do aparelho. Independente disso, notou-se que a distância em que o projeto se mantia funcional era consideravelmente grande.

Para evitar prejudicar os arduinos do LARA para que possam ser utilizados novamente em outros projetos, utilizou-se fita isolante, seguida de fita dupla-face de forte fixação para prender um pedaço de isopor que isola o arduino e suas soldas inferiores do ambiente em que ele irá ser preso, seja parede, seja um ar-condicionado. A figura 5.11 ilustra esses detalhes.

Figura 5.8: Baterias e plug.

Por fim, a figura 5.12 ilustra todo o hardware utilizado no projeto.

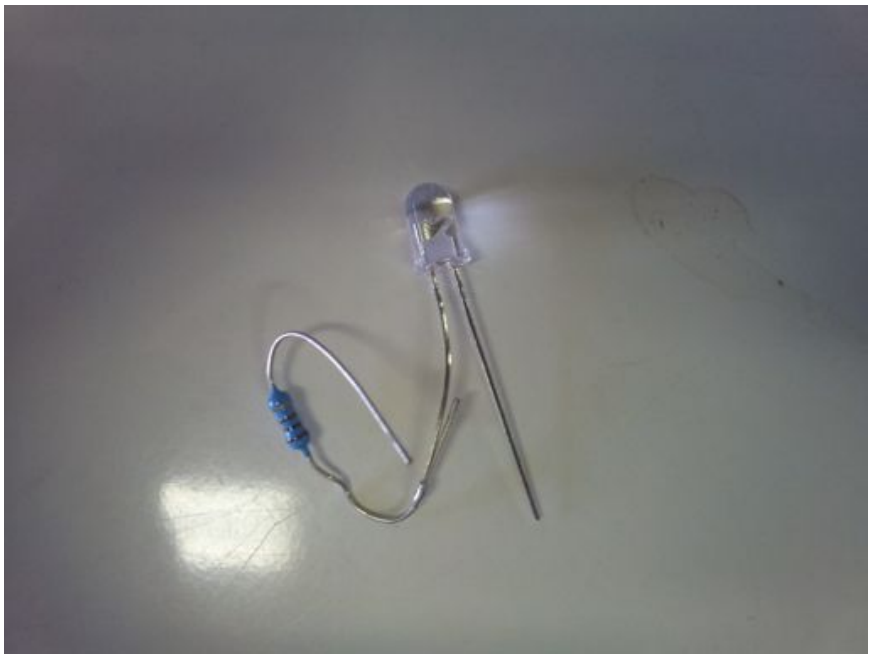


Figura 5.9: Solda feita do LED IR com o resistor de 100 ohms.

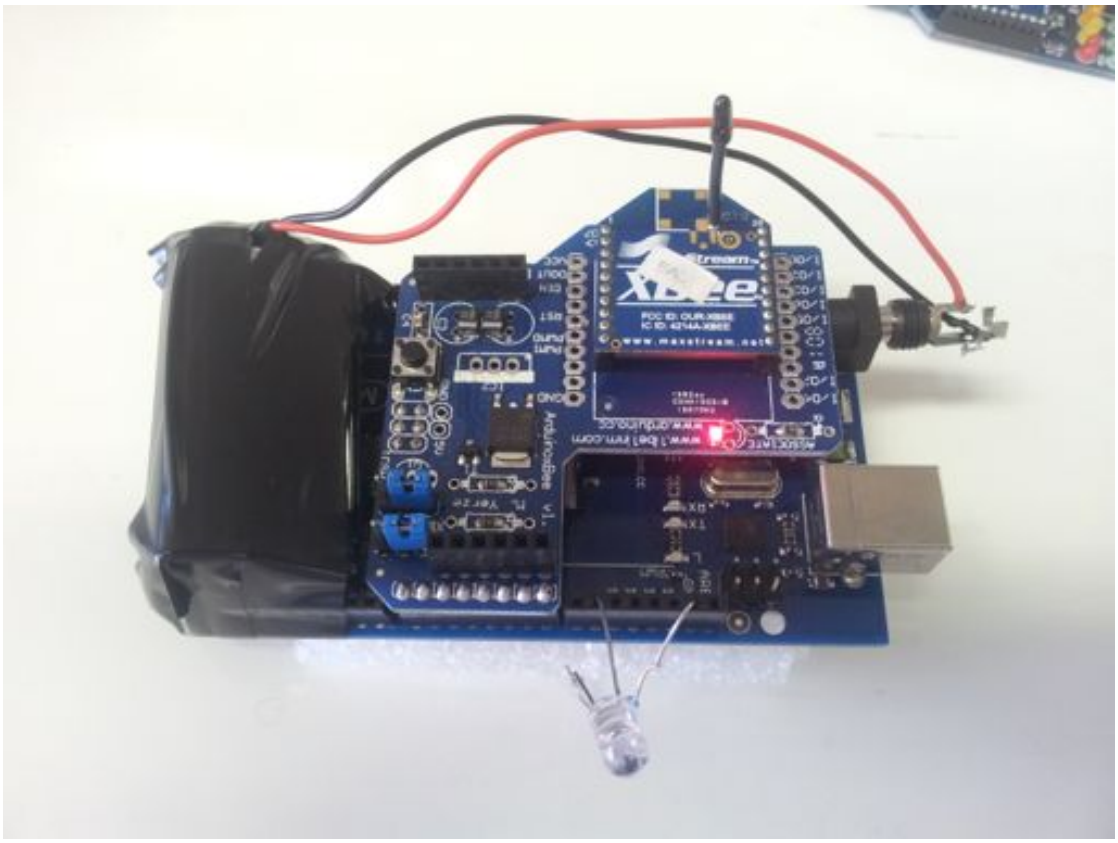


Figura 5.10: Hardware completo de um módulo de acionamento IR.

(a) Fita-isolante para não prejudicar o arduino.

(b) Isopor com fita dupla face para fixação do módulo.

Figura 5.11: Ilustrando detalhes de fita isolante, dupla-face e isopor.

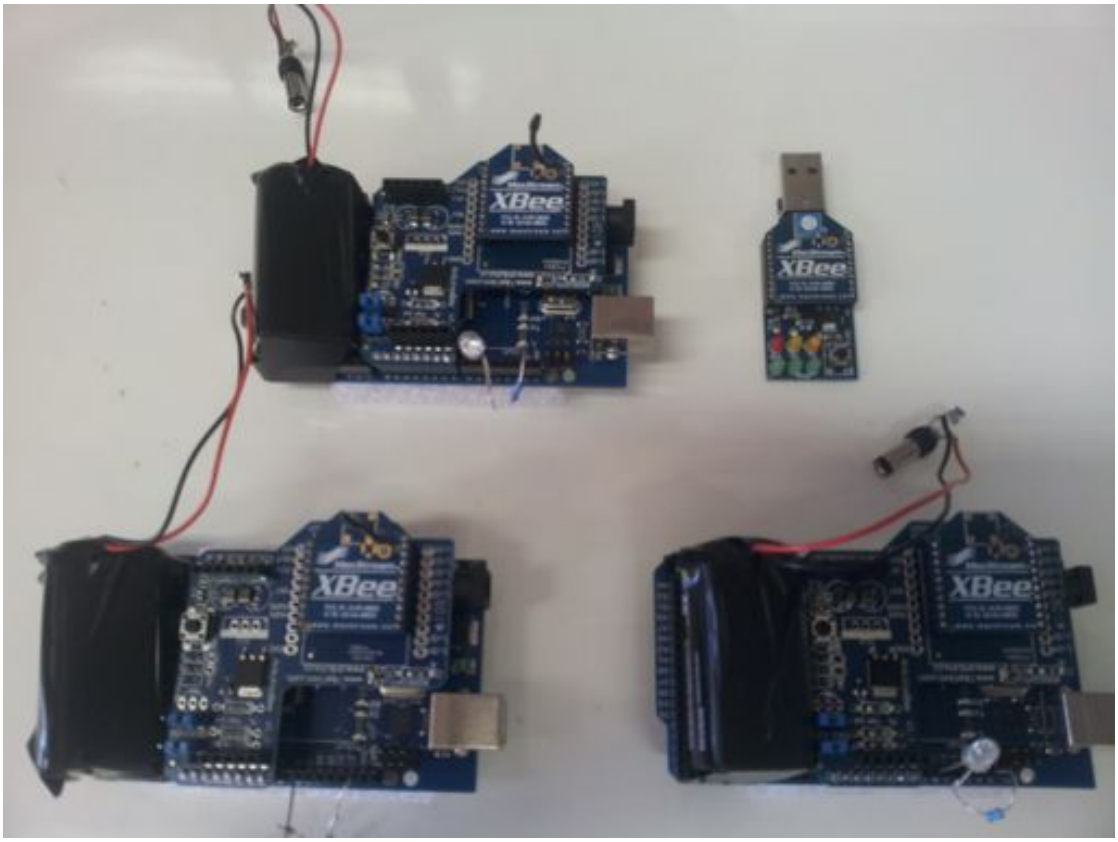


Figura 5.12: Hardware utilizado no projeto: três módulos de acionamento IR e um módulo de interface XBee-PC.

6 ANÁLISE E RESULTADOS

Este capítulo apresenta uma análise dos resultados do projeto após sua conclusão, clarificando-os e apontando possíveis problemas, assim como sucessos do projeto.

6.1 ANÁLISE E RESULTADOS INDIVIDUAIS

6.1.1 Infra-Vermelho

No capítulo 2 foi visto todo o protocolo de comunicação utilizados em controles remoto de forma geral e clarificado os conceitos dessa tecnologia.

Foi tentado então replicar os comandos de um controle remoto de ar-condicionado da marca Springer que são os três aparelhos controlados do LARA.

No começo, facilmente foi possível reenviar comandos de controles remotos mais comuns, que utilizam protocolos da Philips (RC5, RC6), Sony (SIRC), entre outros. Porém, quando tentou-se reenviar comandos do controle do AC, diversos problemas apareceram.

Os problemas são baseados no fato de que o protocolo utilizado nos controles de AC são bem mais complexos, onde carregam diversos tipos de informações e não apenas um comando. Pode-se perceber que esse tipo de controle possui um visor LCD e que nem todo botão resulta em uma ação no aparelho. O que acontece é que cada botão que é apertado, a configuração interna do controle remoto referente ao aparelho de AC é mudada e então é criado um comando a partir dessas novas configurações. Aí então o comando IR é enviado. A figura 6.1 ilustra esse tipo de controle remoto e um aparelho de ar-condicionado do tipo split controlado por ele (Springer e Carrier são essencialmente a mesma marca).

Como o padrão do protocolo desse tipo de controle remoto é fechado, teve-se a idéia então de apenas replicar os comandos. Então fixou-se uma temperatura, desativou-se o relógio, deixou os modos também fixados e começaram-se os testes.

Em princípio não houve sucesso, o protocolo é bem grande, cerca de 99 bytes e pequenas oscilações no tempo de marcas e espaços poderiam ser replicadas nos bytes mais a frente no protocolo e acabar impedindo o comando de ser enviado corretamente.

Alguma alterações na biblioteca IRremote foram então realizadas para tentar desviar dos problemas. A primeira modificação foi referente ao tamanho do buffer de recepção do código a ser replicado. Como o buffer padrão da biblioteca é muito pequeno, o código não era inteiramente recebido e obviamente assim o comando fica impossível de ser enviado corretamente.

A segunda modificação foi em relação ao atraso e avanço que a biblioteca nativamente colocava nas marcas e espaços recebidos. Porém, existe diversos tipos de receptores no mercado, e como o utilizado não existe mais para venda (inclusive foi difícil encontrar o seu datasheet), existia grande possibilidade do problema residir nesse detalhe.

(a) Controle remoto de um AC convencional.

(b) Aparelho de ar-condicionado convencional split.

Figura 6.1: Ar condicionado Split e controle remoto.

Passou-se tentou a tentar valores arbitrários até que de fato conseguiu-se receber o código correto. Curiosamente, outros aparelhos de ar-condicionado possuia controles remotos que esses valores deveriam ser ainda diferentes.

No final, o código foi recebido e guardado em um buffer para que o arduino possuisse o código fixo em sua memória. Funcionou de forma correta, sem problemas e ainda a uma distância considerável, cerca de dois terços da distância que o controle remoto original do aparelho pudesse alcançar.

6.1.2 Aplicativo iPhone

O capítulo 3 teve o intuito de explicar a criação do aplicativo supervisório para iPhone de controle dos aparelhos de ar-condicionado e comunicação com o servidor em um controle remoto.

Por ser uma linguagem totalmente nova, teve que ser aprendida. Felizmente a sintaxe é parecida com C que já é bastante conhecida.

O site oficial de desenvolvimento da Apple fornece documentação forte e detalhada sobre o desenvolvimento de aplicativos para iPhone, incluindo vídeos e tutoriais. Diversos erros e problemas foram encontrados no desenvolvimento do aplicativo, porém foi possível arrumar cada um deles.

No final obteve-se um aplicativo supervisório que pudesse comunicar com um servidor remoto em qualquer outro computador bastante apenas que tivesse o seu IP e que o programa servidor estivesse rodando nesse computador.

Então o supervisório começava a rodar, tendo botões de controle dos ACs, incluindo possibilitada de apertar por um longo tempo e executar uma diferente função. Botão de informação exibindo um alerta

indicando o uso do aplicativo também foi implementado. Indicativos de crédito do criador do aplicativo (o autor desse trabalho) também era mostrado.

Pequenos detalhes tiverem que ser arrumados para que o aplicativo pudesse ser, caso necessário, comercial, e também que contribuisse para sua estética, como o ícone do aplicativo e o logo escolhido do LARA Control.

Foram utilizados o Adobe Illustrator e o Adobe Photoshop (ambos CS6) para criação da logo e dos ícones.

No fim foi criado um supervisório que possibilitava executar todos os objetivos iniciais do projeto e até um pouco mais, mostrando informações de potência atual dos ACs somados.

O supervisório funcionou como o esperado sem maiores problemas. Inclusive o aplicativo é relativamente simples se considerar o hardware (iPhone) utilizado. Assim diminuem as chances de erro relacionadas a processamento.

6.1.3 Servidor em Python

No capítulo 4 foi introduzido o servidor criado em Python que gerencia todas as funcionalidades do projeto. Ele é o mecanismo central do trabalho.

Toda mensagem enviada por qualquer iphone conectado ao servidor deveria passar por esse e então executar sua ação, seja ela passar a mensagem para a frente para os módulos X-Bee, seja ela criar o gráfico.

Cada mensagem passada pelo servidor é impressa na tela para que o usuário possa saber o que está acontecendo. Tanto o comando como os parâmetros são impressos separados.

O servidor também, para manter a robustez, implementa um mecanismo de tratamento de erros (tratamento de exceções de linguagem de orientação a objetos). Então, caso algo não possa ser feito, como parâmetro enviado incorretamente, ou dispositivo serial não encontrado, o servidor imprime na tela uma mensagem de erro, não executa o comando e continua o seu funcionamento normalmente sem travar até que o problema possa ser tratado.

A figura 6.2 ilustra um desses erros sendo tratados e o servidor se mantendo rodando.

Foi observado que o servidor executa suas funcionalidades sem problemas, mantendo a lista de todos os clientes iPhone conectados e gerenciando todos eles, assim como atualizando eles sempre que necessário.

O servidor também cria os gráficos dos relatórios de consumo mensal tranquilamente.

6.1.4 Distribuição de Comandos por X-Bee

O capítulo 5 explica o funcionamento dos módulos assim como suas utilizações no projeto. Os módulos já foram utilizados em diversos outros trabalhos do LARA, mas pela primeira vez foram utilizados com shields X-Bee para arduino.

Esperava-se que os módulos implementassem rede ZigBee mesh ao pensar em utilizá-los no projeto

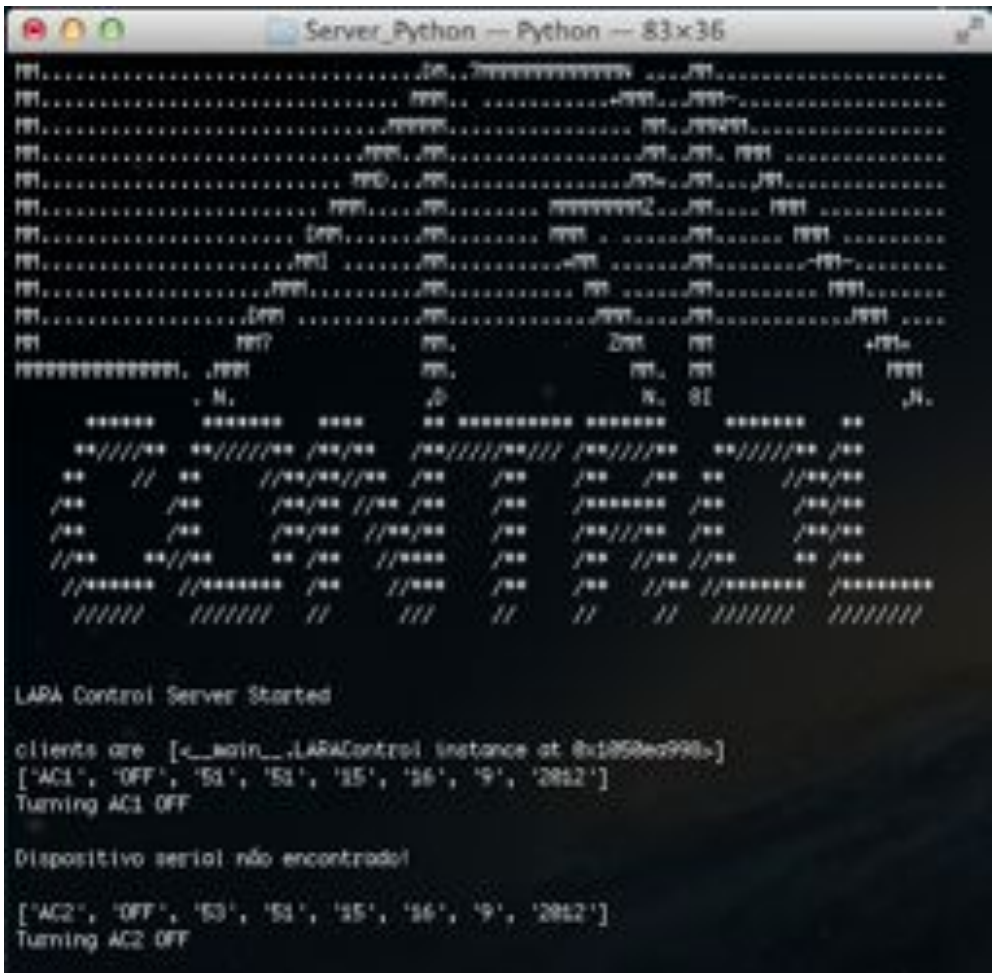


Figura 6.2: Servidor mostrando erro ao tentar enviar mensagem via serial.

pela primeira vez, porém, por serem de Series 1, isso acabou não sendo possível.

Independentemente, os módulos mostraram possuir um alcance razoável, suficiente para o projeto. Caso isso não fosse viável em primeira mão, ainda era possível utilizar um quarto arduino com X-Bee para servir de replicador de sinais, ou seja, apesar de não rotear, ele extende o alcance padrão do módulo base reemitindo as mensagens que chega nele. Assim é possível inclusive dobrar o alcance dos módulos comuns.

Como os três aparelhos de ar-condicionado que foram controlados se encontram perto um do outro, não foi necessária a utilização desse quarto arduino. Caso ainda outro ar-condicionado existisse no LARA para ser controlado poderia começar a ser necessário, pois apesar da distância ser pequena, a quantidade grande de metais no laboratório atrapalha muito a comunicação dos módulos. Em um cenário de testes isso ficou bem visível.

A planta do LARA é ilustrada na figura 6.3. Pode-se perceber a distância entre os aparelhos. De fato deveria ter-se cuidado para que um módulo de outro aparelho não interferisse nos comandos de um anterior a ele, pois a distância do IR alcançada foi razoavelmente grande.

A idéia inicial era prender os módulos com fita dupla-face de forte fixação em algum lugar próximo

Figura 6.3: Planta do LARA.

dos aparelhos de ar-condicionado para que eles pudessem ser controlados sem problemas. Porém, com a distância razoável em que o IR conseguia emitir e a mensagem chegar, notou-se que deixando os módulos em uma bancada próxima ao AC já era o suficiente.

De qualquer forma os módulos foram presos para que pudessem ficar mais estáveis e longe do alcance das pessoas, mantendo assim o sistema mais confiável.

Os módulos se comportaram como o esperado, se comunicando muito bem dentro do LARA. Como a distância era pequena, dificilmente ocorria a perda de pacotes, se mantendo assim desnecessária a adição de um quarto arduino para aumentar o alcance.

6.2 ANÁLISE E RESULTADOS GERAIS

6.2.1 Funcionamento Geral

De acordo com a seção anterior, pode-se perceber que as partes individuais do projeto funcionaram corretamente. Essa seção descreve o funcionamento do projeto inteiro em si, ou seja, com todas as etapas do projeto somadas.

De fato observou-se que o projeto funcionou como o esperado. O usuário conectava o seu iPhone no servidor residente no computador remoto, e então poderia controlar os ACs de acordo com sua necessidade e criar os gráficos de consumo para obter visualização de como os ACs estavam sendo utilizados.

Um problema potencial para o projeto é a política de distribuição de aplicativos da apple. Para um aplicativo como o LARA Control, ou seja, de uso interno, a licença de desenvolvedor (para testar aplicativos e distribui-los) é de 299 dólares por ano. Isso pode inviabilizar todo o projeto considerando sua implementação em um lugar público.

Além disso, para que o projeto seja por si só funcional, é necessário que todos os usuários do laboratório tenham o smartphone. Porém, como isso não é possível, o projeto na verdade é feito como um complemento. Vários projetos já foram feitos no LARA utilizando um supervisório no computador remoto. Este projeto apresenta um complemento onde facilita e flexibiliza a utilização dos aparelhos de ar-condicionado do laboratório pelos usuários que possuem o iPhone.

6.2.2 Relatórios

Os relatórios representam papel fundamental no projeto. Eles que são capazes de mostrar ao usuário, juntamente com os gráficos, como os ACs estão sendo utilizados.

Ou seja, a partir dos relatórios, que são de fácil visualização e entendimento, pode-se saber por exemplo se o aparelho ficou ligado uma noite inteira, se está sendo ligado em horários desnecessários ou até em épocas desnecessárias.

Uma análise rápida do relatório já pode fazer o usuário ter uma boa noção do que está acontecendo e assim definir políticas de racionalização de energia para o laboratório, assim como implementar outros sistemas que auxiliam nessa tarefa.

O servidor cria um relatório para cada ar-condicionado do LARA. A figura 6.4 ilustra um desses relatórios.

Como pode-se perceber, os arquivos de relatórios são simples arquivos de texto que mantém informação de qual aparelho foi ligado ou desligado, asssim como o horário e data em que isso aconteceu. Simples e objetivo.

6.2.3 Gráficos

Para completar um sistema de racionalização eficiente, o supervisório juntamente com o servidor e os relatórios são capazes de emitir gráficos de consumo mensal dos aparelhos. Os gráficos são baseados nos relatórios, e portanto os complementam. Adicionam visualização aos relatórios.

Como os gráficos são de consumo mensal, para coletar dados que sejam relevantes é preciso tempo, portanto os relatórios ainda apresentam relevância grande no projeto para o usuário.

Considerando isso, foi criado um relatório hipotético para criar um gráfico sem necessitar esperar meses para isso. O gráfico é ilustrado na figura 6.5. Nota-se que o gráfico apresenta comandos para salvá-lo, obter zoom, entre outros. Dados reais não são viáveis para o projeto devido a demora para consegui-los.

Pode-ser ver então quais meses estão gastando mais energia, relacionando esse gasto com a época do ano ou então relacionando a um mês anterior a uma política de racionalização.

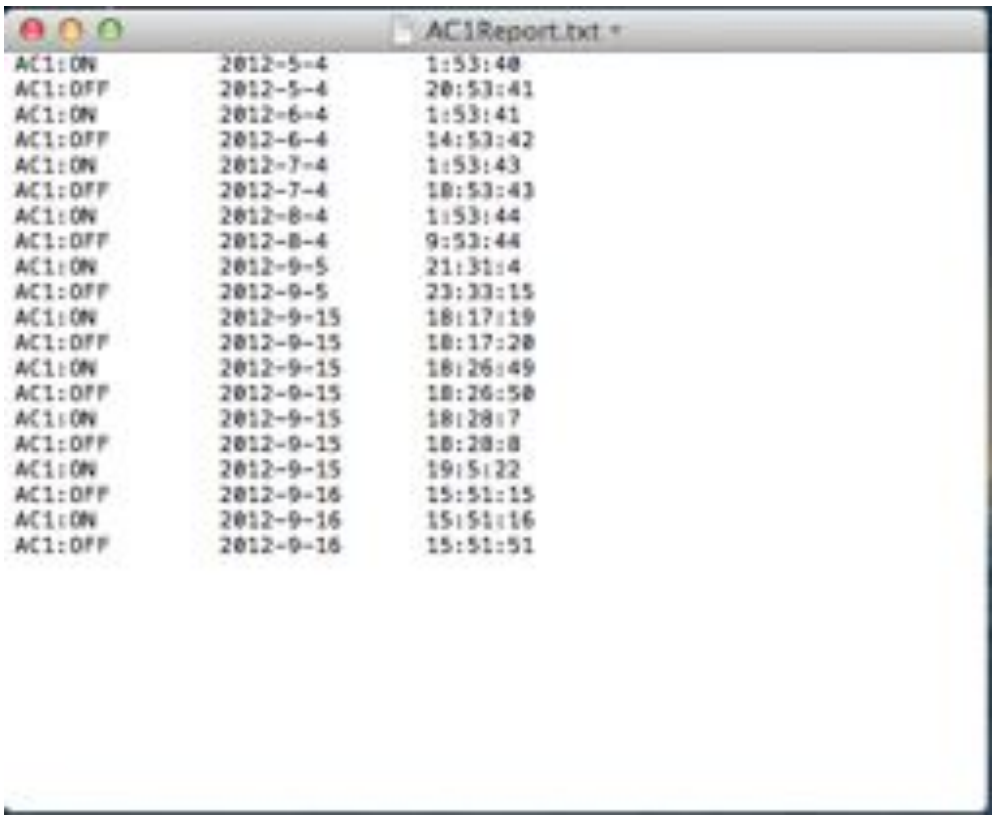


Figura 6.4: Exemplo de relatório de uso do AC.

Por exemplo, caso seja notado em um relatório que as pessoas estão deixando os aparelhos ligados várias noites desnecessariamente, é possível lembrar as pessoas de desligar os aparelhos, e então comparar o consumo depois dessa política com o mês anterior.

Os relatórios juntamente com os gráficos podem ser utilizados de diversas outras formas para racionalizar energia. A potência atual mostrada no supervisório também pode auxiliar nesse quesito.

Além disso, com futuros projetos (que já estão sendo feitos no LARA) de medidores de energia somados a este projeto pode-se conseguir uma racionalização bastante eficiente de sistemas de climatização prediais.

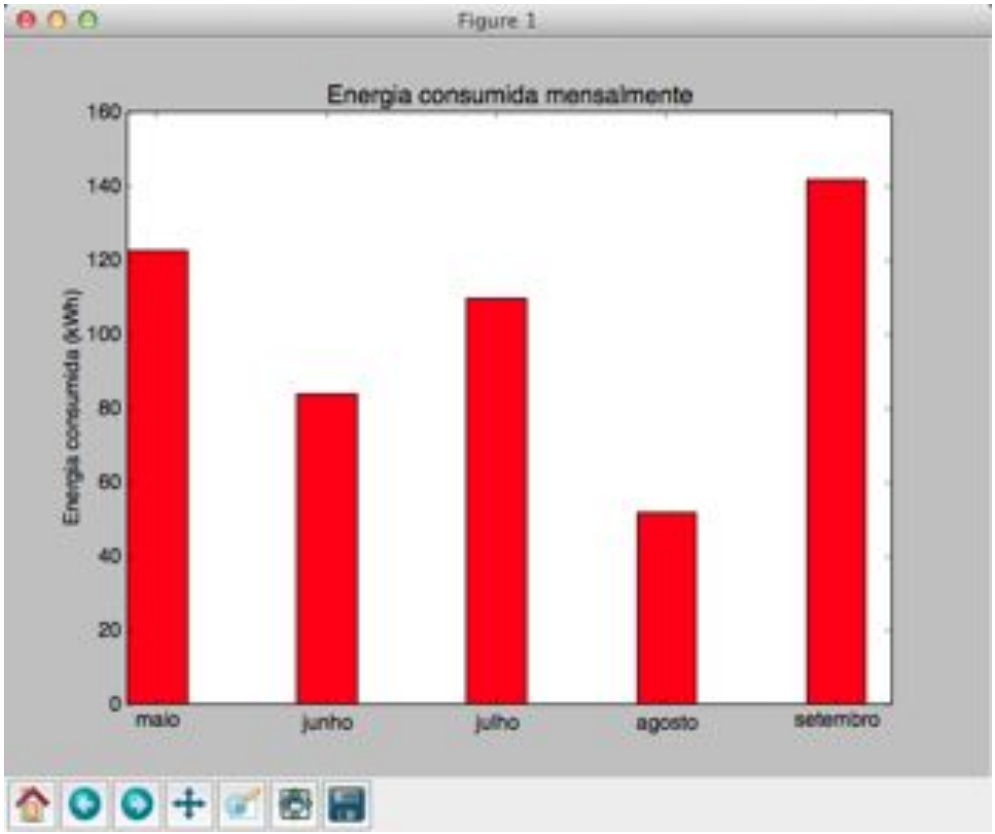


Figura 6.5: Exemplo de gráfico ilustrativo criado a partir de um relatório (dados fictícios).

7 CONCLUSÕES

7.1 CONCLUSÕES GERAIS

Este projeto apresentou uma nova forma de controlar o aparelho de ar-condicionado, diferentemente daquelas já feitas no LARA. Tanto da parte de acionamento do aparelho, a qual usou aqui controle via infra-vermelho, como da parte de interface com o usuário, sendo utilizado aqui um smartphone, no caso, o iPhone.

As implementações aqui utilizadas são bem interessantes, pois apresentam um meio flexível e não-invasivo de controle dos aparelhos de ar-condicionado. Isso tanto da parte do IR, como dos arduinos e do servidor.

Foram quatro etapas separadas. Uma referente ao aplicativo supervisório que representa a interface do usuário com o iPhone. A segunda com um servidor recebendo a mensagem Wi-Fi enviada pelo smartphone. A terceira com o servidor enviando a mensagem serialmente para o arduino e esse distribuindo entre os outros módulos via X-Bee, e por fim o arduino transmitindo a mensagem via IR para o ar-condicionado. Sem contar a criação de gráficos e relatórios.

Muitos conceitos novos tiveram que ser aprendidos para este trabalho, tendo assim grande aproveitamento. Além disso, abre uma ramificação para novos trabalhos realizados no LARA. Podendo incluir novas redes de sensores ao LARA Control, entre outros.

Concluiu-se que é possível fazer o controle do condicionador de ar via IR repetindo ou aprendendo os comandos do controle remoto. Também é possível utilizar um smartphone no processo. O projeto obteve êxito em todas as etapas.

O IR permite utilizar a malha de controle do aparelho de ar-condicionado, sendo assim mantendo sua vida útil e retirando uma parte desnecessária ao projeto, além de ser não invasivo e de fácil implementação. Outros métodos utilizados no LARA utilizava um tiristor para fazer esse controle. Deixando a temperatura fixa e cortando a energia do aparelho e religando-a fazendo um controle liga-desliga. Isso diminui a vida útil dos motores do direcionador de ar do aparelho, assim como de seu compressor. Inclusive se for mal implementado pode diminuir consideravelmente a vida útil do aparelho.

Além disso, os gráficos e relatórios criados ajudam na racionalização de energia de sistemas de climatização prediais devido a informação que é apresentada ao usuário de como o sistema está sendo utilizado. Assim, diversas políticas de racionalização podem ser implementadas no laboratório ou qualquer outro lugar que utilize o projeto, deixando-o altamente útil e interessante, tanto do ponto de vista de economia, como do ponto de vista verde, ou seja, preservar a natureza diminuindo o gasto elétrico.

É interessante notar também que por mais que comunicação via IR seja uma tecnologia já antiga, é utilizada até hoje, por ser barata, eficiente e flexível.

Arduino é vastamente utilizado no mundo inteiro e smartphones estão cada vez mais presentes na vida das pessoas, sendo boa escolha então para utilizá-lo como controle de aparelhos e dispositivos.

7.2 SUGESTÕES DE TRABALHOS FUTUROS

O projeto utiliza tecnologias atuais, que podem ser integradas em diversos outros projetos, tendo como sugestões de trabalhos futuros:

- Criação de uma interface de usuário com Android, ou seja, criação de um aplicativo LARA Control para esse sistema operacional para smartphones.

- Implementar um sistema de medição de energia com o LARA Control.

- Criação de um servidor com uma GUI, ou interface gráfica de usuário.

- Tentar resolver o protocolo do controle remoto do ar-condicionado para assim não ser preciso mandar os códigos *raw* mas sim por um número hexadecimal. Devem ser implementadas então funções capazes de criar um código IR a partir das configurações desejadas.

- Integrar o controle via IR do condicionador de ar com um software supervisório, além de poder manter o controle via iPhone. Assim, pode-se ter um controle automático do aparelho e ainda uma intervenção do usuário para casos que sejam exceções à regra.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Disponível em:. http://www.arcfn.com/2009/08/multi-protocol-infrared-remote-library.html.

[2] Disponível em:. https://developer.apple.com/.

[3] Disponível em:. http://www.python.org/.

[4] Disponível em:. http://twistedmatrix.com/trac/.

[5] Disponível em:. http://pyserial.sourceforge.net/.

[6] Disponível em:. http://matplotlib.org/.

[7] Disponível em:. http://numpy.scipy.org/.

[8] Disponível em:. http://docs.python.org/library/datetime.html.

[9] Disponível em:. http://www.digi.com/support/productdetail?pid=3352.

[10] Disponível em:. http://code.google.com/p/xbee-api/wiki/.

[11] Fonte:. http://en.wikipedia.org/wiki/Infra_red.

[12] Fonte:. http://en.wikipedia.org/wiki/Modulation.

[13] Fonte:. http://en.wikipedia.org/wiki/PWM.

[14] Fonte:. http://en.wikipedia.org/wiki/XBee.

[15] Oficialmente em:. http://arduino.cc/en/Guide/ArduinoXbeeShield.

[16] Site oficial:. http://arduino.cc/en/Main/ArduinoBoardUno.

[17] Site oficial:. http://arduino.cc/en/Main/ArduinoBoardMega.

[18] Site oficial:. http://www.rogercom.com/Produtos.htm.

[19] L. P. BORGES and R. C. DORES. Automação predial sem fio utilizando bacnet/zigbee com foco em economia de energia, 2010.

[20] M.V. M. Euzébio. Droidlar - automação residencial através de um celular android, 2011.

[21] C. L. MARTE. Automação predial: a inteligência distribuída nas edificações. *São Paulo: Carthago e Forte*, 1995.

[22] S. J. MONTEBELLER. Estudo sobre o emprego de dispositivos sem fio - wireless na automação do ar condicionado e de outros sistemas prediais. Master's thesis, Escola Politécnica da Universidade de São Paulo, 2006.

[23] J. D. J. URBANO and M. C. C. NOVAIS. Instrumentação e controle de um sistema de ar condicionado híbrido instrumentação e controle de um sistema de ar condicionado híbrido utilizando bacnet sobre zigbee, 2010.

ANEXOS

I. PROTOCOLO IR DE CONTROLE DOS ACS

Neste anexo se encontra o protocolo infra-vermelho utilizado para o controle dos aparelhos de ar-condicionado. Os valores se encontram em micro-segundos e cada valor seguido de um ”m”significa que representa o tempo de uma marca, e da mesma forma cada valor seguido de um ”s”significa que representa o tempo de um espaço.

O protocolo abaixo é para as configurações ”mode”e ”fan”fixas em ”auto”. O relógio está desativado (00:00). Para ambos comandos, a temperatura é fixa em 21 graus celsius, ou seja, para o comando de desligar não faz diferença, porém ao mandar o comando de ligar o aparelho se encontrará nesta temperatura.

Comando ″ON″:

m4550 s4300 m700 s1500 m650 s450 m700 s1450 m700 s1550 m650 s450 m700 s450 m700 s1450 m650 s450 m700 s450 m700 s1450 m650 s450 m700 s400 m700 s1500 m700 s1450 m700 s450 m700 s1450 m650 s450 m700 s400 m700 s450 m700 s1450 m700 s1500 m650 s1500 m650 s1550 m650 s1500 m650 s1500 m700 s1500 m650 s1500 m700 s400 m750 s350 m750 s400 m700 s450 m650 s450 m700 s400 m700 s1500 m700 s1450 m700 s450 m650 s1500 m700 s400 m750 s350 m750 s400 m650 s1500 m700 s400 m750 s400 m650 s1500 m700 s400 m750 s1450 m650 s1500 m750 s1450 m650 s4700 m4550 s4300 m700 s1500 m700 s400 m700 s1500 m700 s1450 m700 s500 m650 s450 m650 s1500 m650 s450 m700 s450 m650 s1500 m650 s450 m700 s450 m700 s1450 m650 s1500 m700 s400 m750 s1450 m650 s450 m700 s450 m650 s450 m700 s1450 m700 s1500 m650 s1500 m700 s1500 m650 s1500 m650 s1550 m700 s1450 m650 s1500 m700 s400 m750 s400 m700 s400 m750 s400 m650 s450 m700 s400 m700 s1500 m650 s1500 m700 s400 m750 s1450 m650 s450 m700 s400 m750 s400 m700 s1450 m700 s450 m650 s400 m750 s1450 m650 s450 m700 s1500 m650 s1500 m700 s1450 m750

Comando ″OFF″:

m4550 s4350 m750 s1400 m700 s450 m700 s1450 m750 s1450 m700 s400 m750 s350 m650 s1550 m650 s450 m700 s400 m750 s1450 m650 s450 m650 s450 m750 s1400 m700 s1500 m700 s400 m700 s1500 m700 s400 m650 s1500 m700 s1500 m700 s1450 m650 s1550 m700 s400 m650 s1500 m700 s1450 m700 s1500 m700 s400 m750 s350 m700 s450 m700 s400 m700 s1450 m750 s350 m750 s400 m700 s1450 m700 s1500 m650 s1500 m700 s400 m700 s450 m700 s400 m700 s400 m750 s350 m700 s400 m750 s400 m700 s400 m750 s1400 m700 s1500 m650 s1500 m750 s1450 m650 s1500 m750 s4500 m4600 s4300 m700 s1500 m650 s450 m650 s1500 m750 s1450 m700 s400 m700 s400 m750 s1450 m650 s450 m650 s450 m650 s1500 m700 s450 m650 s500 m750 s1400 m700 s1500 m700 s400 m700 s1500 m650 s450 m700 s1500 m650 s1500 m700 s1450 m700 s1500 m700 s400 m700 s1450 m750 s1450 m700 s1450 m700 s400 m750 s400 m700 s400 m750 s350 m750 s1450 m650 s450 m700 s400 m750 s1450 m700 s1450 m700 s1500 m700 s400 m700 s400 m700 s400 m750 s400 m700 s400 m700 s400 m700 s400 m750 s1450 m700 s1450 m650 s1500 m700 s1500 m650 s1550 m700

II. CÓDIGO-FONTE DO SUPERVISÓRIO PARTE: I

```
//
//  LARAControlViewController.h
//  LARAControl
//
//  Created by Daniel Vilela on 08/08/12.
//  Copyright (c) 2012 UnB. All rights reserved.
//

#import <UIKit/UIKit.h>

NSInputStream *inputStream;
NSOutputStream *outputStream;

NSMutableArray * messages;

typedef struct ToggleTime {
    NSInteger seconds;
    NSInteger minutes;
    NSInteger hours;
    NSInteger day;
    NSInteger month;
    NSInteger year;
} ToggleTime;

BOOL AC1State = NO;
BOOL AC2State = NO;
BOOL AC3State = NO;

NSString *ACGraphed = @"";

@interface LARAControlViewController : UIViewController <NSStreamDelegate,
    UITextFieldDelegate>

@property (weak, nonatomic) IBOutlet UIView *IPSetView;
@property (weak, nonatomic) IBOutlet UIView *SupervisoryView;
@property (weak, nonatomic) IBOutlet UIView *makeGraphView;
@property (weak, nonatomic) IBOutlet UIButton *infoButton;

@property (weak, nonatomic) IBOutlet UIButton *AC1Button;
@property (weak, nonatomic) IBOutlet UIButton *AC2Button;
@property (weak, nonatomic) IBOutlet UIButton *AC3Button;

@property (weak, nonatomic) IBOutlet UITextField *IPField;
@property (weak, nonatomic) IBOutlet UILabel *makeGraphField;
@property (weak, nonatomic) IBOutlet UILabel *powerConsumeField;
```

```
- (IBAction)infoButton:(id)sender;

- (IBAction)AC1Button:(id)sender;
- (IBAction)AC2Button:(id)sender;
- (IBAction)AC3Button:(id)sender;

- (IBAction)startClient:(id)sender;

- (IBAction)makeGraphCommand:(id)sender;
- (IBAction)graphNotNow:(id)sender;

- (ToggleTime)DiscoverDate;
- (NSString *)makePowerConsumeTextField;

@end
```

III. CÓDIGO-FONTE DO SUPERVISÓRIO PARTE: II

```
//
//  LARAControlViewController.m
//  LARAControl
//
//  Created by Daniel Vilela on 08/08/12.
//  Copyright (c) 2012 UnB. All rights reserved.
//

#import "LARAControlViewController.h"

@interface LARAControlViewController ()

@end

@implementation LARAControlViewController
@synthesize IPSetView;
@synthesize SupervisoryView;
@synthesize makeGraphView;
@synthesize infoButton;
@synthesize AC1Button;
@synthesize AC2Button;
@synthesize AC3Button;
@synthesize IPField;
@synthesize makeGraphField;
@synthesize powerConsumeField;

- (void)viewDidLoad
{
    [super viewDidLoad];
    messages = [[NSMutableArray alloc] init];
  // Do any additional setup after loading the view, typically from a nib.
}

- (void)viewDidUnload
{
    [self setIPField:nil];
    [self setIPSetView:nil];
    [self setSupervisoryView:nil];
    [self setInfoButton:nil];
    [self setAC1Button:nil];
    [self setAC2Button:nil];
    [self setAC3Button:nil];
    [self setMakeGraphView:nil];
    [self setMakeGraphField:nil];
    [self setPowerConsumeField:nil];
    [super viewDidUnload];
```

```
    // Release any retained subviews of the main view.
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
    interfaceOrientation
{
    return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
}

- (IBAction)infoButton:(id)sender
{
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"How to Use"
                                                    message:@"Click the AC Buttons to turn
                                                        on or off LARA's ACs. Hold one of
                                                        the buttons to create graphs of
                                                        power consumption."
                                                    delegate:self
                                                    cancelButtonTitle:@"Dismiss"
                                                    otherButtonTitles:nil];

    [alert show];
}

- (IBAction)AC1Button:(id)sender
{
    NSString *response;
    ToggleTime time =self.DiscoverDate;

    if( AC1State == NO) {
        [AC1Button setImage:[UIImage imageNamed:@"B_ON.bmp"] forState:
            UIControlStateNormal];
        response  = [NSString stringWithFormat:@"AC1:ON:%d:%d:%d:%d:%d:%d", time.
            seconds, time.minutes, time.hours, time.day, time.month, time.year];
        NSLog(@"Turning AC1 On");
        AC1State = YES;
    }
    else {
        [AC1Button setImage:[UIImage imageNamed:@"B_OFF.bmp"] forState:
            UIControlStateNormal];
        response  = [NSString stringWithFormat:@"AC1:OFF:%d:%d:%d:%d:%d:%d", time.
            seconds, time.minutes, time.hours, time.day, time.month, time.year];
        NSLog(@"Turning AC1 Off");
        AC1State = NO;
    }

    NSData *data = [[NSData alloc] initWithData:[response dataUsingEncoding:
        NSASCIIStringEncoding]];
  [outputStream write:[data bytes] maxLength:[data length]];
}

- (IBAction)AC2Button:(id)sender
{
```

```
    NSString *response;
    ToggleTime time =self.DiscoverDate;

    if( AC2State == NO) {
        [AC2Button setImage:[UIImage imageNamed:@"B_ON.bmp"] forState:
            UIControlStateNormal];
        response  = [NSString stringWithFormat:@"AC2:ON:%d:%d:%d:%d:%d:%d", time.
            seconds, time.minutes, time.hours, time.day, time.month, time.year];
        NSLog(@"Turning AC2 On");
        AC2State = YES;
    }
    else {
        [AC2Button setImage:[UIImage imageNamed:@"B_OFF.bmp"] forState:
            UIControlStateNormal];
        response  = [NSString stringWithFormat:@"AC2:OFF:%d:%d:%d:%d:%d:%d", time.
            seconds, time.minutes, time.hours, time.day, time.month, time.year];
        NSLog(@"Turning AC2 Off");
        AC2State = NO;
    }

    NSData *data = [[NSData alloc] initWithData:[response dataUsingEncoding:
        NSASCIIStringEncoding]];
  [outputStream write:[data bytes] maxLength:[data length]];
}

- (IBAction)AC3Button:(id)sender
{
    NSString *response;
    ToggleTime time = self.DiscoverDate;

    if( AC3State == NO) {
        [AC3Button setImage:[UIImage imageNamed:@"B_ON.bmp"] forState:
            UIControlStateNormal];
        response  = [NSString stringWithFormat:@"AC3:ON:%d:%d:%d:%d:%d:%d", time.
            seconds, time.minutes, time.hours, time.day, time.month, time.year];
        NSLog(@"Turning AC3 On");
        AC3State = YES;
    }
    else {
        [AC3Button setImage:[UIImage imageNamed:@"B_OFF.bmp"] forState:
            UIControlStateNormal];
        response  = [NSString stringWithFormat:@"AC3:OFF:%d:%d:%d:%d:%d:%d", time.
            seconds, time.minutes, time.hours, time.day, time.month, time.year];
        NSLog(@"Turning AC3 Off");
        AC3State = NO;
    }

    NSData *data = [[NSData alloc] initWithData:[response dataUsingEncoding:
        NSASCIIStringEncoding]];
  [outputStream write:[data bytes] maxLength:[data length]];
}
```

```
- (IBAction)startClient:(id)sender
{
    //[self.view bringSubviewToFront:SupervisoryView];
    [self initNetworkCommunication];
    IPSetView.hidden = YES;
    infoButton.hidden = NO;
}

- (IBAction)makeGraphCommand:(id)sender
{
    NSString *response;

    response  = [NSString stringWithFormat:@"GRAPH:%@:00:00:00:00:00:00", ACGraphed];
    NSLog(@"Sending Graph Command for %@", ACGraphed);

    NSData *data = [[NSData alloc] initWithData:[response dataUsingEncoding:
        NSASCIIStringEncoding]];
  [outputStream write:[data bytes] maxLength:[data length]];

    self.makeGraphView.hidden = YES;
}

- (IBAction)graphNotNow:(id)sender
{
    self.makeGraphView.hidden = YES;
}

- (void)initNetworkCommunication
{
    CFReadStreamRef readStream;
    CFWriteStreamRef writeStream;
    CFStreamCreatePairWithSocketToHost(NULL, (__bridge CFStringRef)(IPField.text),
        80, &readStream, &writeStream);
    inputStream = (__bridge NSInputStream *)readStream;
    outputStream = (__bridge NSOutputStream *)writeStream;

    [inputStream setDelegate:self];
    [outputStream setDelegate:self];

    [inputStream scheduleInRunLoop:[NSRunLoop currentRunLoop] forMode:
        NSDefaultRunLoopMode];
    [outputStream scheduleInRunLoop:[NSRunLoop currentRunLoop] forMode:
        NSDefaultRunLoopMode];

    [inputStream open];
    [outputStream open];
}

- (void)stream:(NSStream *)theStream handleEvent:(NSStreamEvent)streamEvent
{
  switch (streamEvent)
    {
```

```
case NSStreamEventOpenCompleted:
  NSLog(@"Stream opened");
  break;

    //Prepared to receive info from the server
case NSStreamEventHasBytesAvailable:
        if (theStream == inputStream)
        {
            uint8_t buffer[1024];
            int len;

            while ([inputStream hasBytesAvailable])
            {
                len = [inputStream read:buffer maxLength:sizeof(buffer)];
                if (len > 0)
                {
                    NSString *output = [[NSString alloc] initWithBytes:buffer
                        length:len encoding:NSASCIIStringEncoding];

                    if (nil != output) {
                        //NSLog(@"server said: %@", output);
                        NSArray *array = [output componentsSeparatedByString:@":"
                            ];
                        if ([[array objectAtIndex:0] isEqual:@"UPDATE"]) {
                            if( [[array objectAtIndex:1] isEqual:@"ON"] ) {
                                AC1State = YES;
                                [AC1Button setImage:[UIImage imageNamed:@"B_ON.
                                    bmp"] forState:UIControlStateNormal];
                            }
                            else {
                                AC1State = NO;
                                [AC1Button setImage:[UIImage imageNamed:@"B_OFF.
                                    bmp"] forState:UIControlStateNormal];
                            }

                            if( [[array objectAtIndex:2] isEqual:@"ON"] ) {
                                AC2State = YES;
                                [AC2Button setImage:[UIImage imageNamed:@"B_ON.
                                    bmp"] forState:UIControlStateNormal];
                            }
                            else {
                                AC2State = NO;
                                [AC2Button setImage:[UIImage imageNamed:@"B_OFF.
                                    bmp"] forState:UIControlStateNormal];
                            }

                            if( [[array objectAtIndex:3] isEqual:@"ON"] ) {
                                AC3State = YES;
                                [AC3Button setImage:[UIImage imageNamed:@"B_ON.
                                    bmp"] forState:UIControlStateNormal];
                            }
                            else {
```

```
                                    AC3State = NO;
                                    [AC3Button setImage:[UIImage imageNamed:@"B_OFF.
                                        bmp"] forState:UIControlStateNormal];
                                }
                            }
                        }
                    }
                }
            }
        break;

    case NSStreamEventErrorOccurred:
      NSLog(@"Cannot connect to the host!");
      break;

    case NSStreamEventEndEncountered:
            [theStream close];
            [theStream removeFromRunLoop:[NSRunLoop currentRunLoop] forMode:
                NSDefaultRunLoopMode];
      break;

    default:
      //NSLog(@"Unknown event");
            break;
  }
}

- (IBAction)AC1LongPress:(id)sender
{
    ACGraphed = @"AC1";
    self.powerConsumeField.text = self.makePowerConsumeTextField;
    self.makeGraphField.text = @"Make Graph for AC1?";
    self.makeGraphView.hidden = NO;
}

- (IBAction)AC2LongPress:(id)sender
{
    ACGraphed = @"AC2";
    self.powerConsumeField.text = self.makePowerConsumeTextField;
    self.makeGraphField.text = @"Make Graph for AC2?";
    self.makeGraphView.hidden = NO;
}

- (IBAction)AC3LongPress:(id)sender
{
    ACGraphed = @"AC3";
    self.powerConsumeField.text = self.makePowerConsumeTextField;
    self.makeGraphField.text = @"Make Graph for AC3?";
    self.makeGraphView.hidden = NO;
}

- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {
```

```
    if (theTextField == self.IPField) {
        [theTextField resignFirstResponder];
    }
    return YES;
}

- (ToggleTime)DiscoverDate
{
    ToggleTime ToggleTimeUsed;
    NSDate *today = [NSDate date];
    NSCalendar *gregorian = [[NSCalendar alloc]
                             initWithCalendarIdentifier:NSGregorianCalendar];
    NSDateComponents *Components =
    [gregorian components:(NSSecondCalendarUnit | NSMinuteCalendarUnit |
        NSHourCalendarUnit | NSDayCalendarUnit | NSMonthCalendarUnit |
        NSYearCalendarUnit) fromDate:today];

    ToggleTimeUsed.seconds = [Components second];
    ToggleTimeUsed.minutes = [Components minute];
    ToggleTimeUsed.hours = [Components hour];
    ToggleTimeUsed.day = [Components day];
    ToggleTimeUsed.month = [Components month];
    ToggleTimeUsed.year = [Components year];

    return ToggleTimeUsed;
}

-(NSString *)makePowerConsumeTextField
{
    NSInteger count = 0;
    NSString *str = nil;

    if(AC1State == YES)
        count++;
    if(AC2State == YES)
        count++;
    if(AC3State == YES)
        count++;

    str = [[NSString alloc ] initWithFormat:@"Current␣Power:␣%dW", count*1895];
    return str;
}

@end
```

IV. CÓDIGO-FONTE DO SERVIDOR EM PYTHON

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

#
#  server.py
#  LARAControl Server
#
#  Created by Daniel Vilela on 01/08/12.
#  Copyright (c) 2012 UnB. All rights reserved.
#

from twisted.internet.protocol import Factory, Protocol
from twisted.internet import reactor
from datetime import datetime
import matplotlib.pyplot as plt
import numpy.numarray as na
import serial

#---------------- Defining time struct ----------------#
class timeStruct():
    def __init__(self, sec, min, hour, day, month, year):
        self.seconds = sec
        self.minutes = min
        self.hours = hour
        self.day = day
        self.month = month
        self.year = year


#---------------- Defining report function ----------------#
def createReport(selectedAC, ACstate, time):
  dict = {}
  dict = (selectedAC, ACstate, time.year, time.month, time.day, time.hours, time.
      minutes, time.seconds)
  line = "{0}:{1}\t\t{2}-{3}-{4}\t{5}:{6}:{7}\t\n".format(*dict)

  if selectedAC == "AC1":
    ACReport = open("AC1Report.txt", "a")

  elif selectedAC == "AC2":
    ACReport = open("AC2Report.txt", "a")

  else:
    ACReport = open("AC3Report.txt", "a")

  ACReport.write(line)
```

```
  ACReport.close()


#---------------- Defining split function ----------------#
def splitFunction(selectedAC):
  try:
    if selectedAC == "AC1":
      ACReport = open("AC1Report.txt", "r")

    elif selectedAC == "AC2":
      ACReport = open("AC2Report.txt", "r")

    else:
      ACReport = open("AC3Report.txt", "r")
  except IOError:
    print "Nenhum arquivo de log encontrado.\n"
    raise NameError('Error: No file for graph')
    return

  #read and split
  flag = "first"
  on_time = []
  on_time_aux = 0
  first_month = 0
  for line in ACReport:
    report = line.split('\t')
    date = report[2].split('-')

    if(flag == "first"):
      flag = "ON"
      first_month = date[1]
      month_control = date[1]
    month = date[1]

    if flag == "ON":
      t1 = datetime.strptime(report[3], '%H:%M:%S')
      if(month_control != month):
        month_control = month
        on_time.append(((on_time_aux/3600)*6447.562)/1000) #kWh consumidos no ês
        on_time_aux = 0
      flag = "OFF"

    else:
      t2 = datetime.strptime(report[3], '%H:%M:%S')
      delta = t2 - t1
      on_time_aux += delta.seconds
      flag = "ON"

  on_time.append(((on_time_aux/3600)*6447.562)/1000)
  on_time.append(int(first_month))

  ACReport.close()
```

```
    return on_time


#---------- Defining graph function ----------#
def createBarGraph(selectedAC):
    try:
        consumed_power = splitFunction(selectedAC)
    except NameError:
        return

    first_month = consumed_power[len(consumed_power)-1]
    consumed_power.pop()

    values = tuple(consumed_power)
    ind = na.array(range(len(values))) + 0.5
    width = 0.35
    plt.bar(ind, values, width, color='r')
    plt.ylabel('Energia consumida (kWh)'.decode('utf8'))
    plt.title('Energia consumida mensalmente'.decode('utf8'))

    all_months = ['janeiro', 'fevereiro', 'çmaro', 'abril', 'maio', 'junho', 'julho', '
        agosto', 'setembro', 'outubro', 'novembro', 'dezembro']
    months_used = []
    for i in range(0, len(consumed_power)):
        months_used.append(all_months[first_month + i - 1])

    plt.xticks( ind+width/2, tuple(months_used) )
    plt.show()


#---------- Defining serial function ----------#
def sendSerial(selectedAC, ACState):

    try:
        ser = serial.Serial('/dev/tty.usbserial-RCQR4VNB', 9600)
    except serial.SerialException:
        print "Dispositivo serial ãno encontrado!\n"
        return

    byte = 0b00000000
    if(selectedAC == "AC1"):
        if(ACState == "ON"):
            byte = 0b00000011
        else:
            byte = 0b00000010
    elif(selectedAC == "AC2"):
        if(ACState == "ON"):
            byte = 0b00001100
        else:
            byte = 0b00001000
    else:
        if(ACState == "ON"):
```

```
        byte = 0b00110000
      else:
        byte = 0b00100000

    try:
      ser.write(chr(byte))
    except serial.SerialException:
      print "ãNo é ípossvel enviar comando serial."

    ser.close()


#---------- Defining update ACs function ----------#
def updateACs():
    AC1 = updateACsHelper("AC1")
    AC2 = updateACsHelper("AC2")
    AC3 = updateACsHelper("AC3")

    dict = {}
    dict = (AC1, AC2, AC3)
    msg = "UPDATE:{0}:{1}:{2}".format(*dict)

    return msg


#---------- Defining updateACHelper function ----------#
def updateACsHelper(selectedAC):
    try:
      if selectedAC == "AC1":
        ACReport = open("AC1Report.txt", "r")

      elif selectedAC == "AC2":
        ACReport = open("AC2Report.txt", "r")

      else:
        ACReport = open("AC3Report.txt", "r")
    except IOError:
      return "OFF"

    for line in ACReport:
      report = line.split('\t')
      if (report[0] != ''):
        ACState = report[0].split(':')

    return ACState[1]


#---------- Defining LARAControl class ----------#
class LARAControl(Protocol):
    def connectionMade(self):
      self.factory.clients.append(self)
      print "clients are ", self.factory.clients
```

```
    msg = updateACs()
    for c in self.factory.clients:
      c.message(msg)

  def connectionLost(self, reason):
    self.factory.clients.remove(self)

  def dataReceived(self, data):
    a = data.split(':')
    print a
    if len(a) > 1:
      try:
        ACselector = a[0]
        ACState = a[1]
        Time = timeStruct(a[2], a[3], a[4], a[5], a[6], a[7])
      except:
        return

      if(a[0] == "GRAPH"):
        self.AC = ACselector
        createBarGraph(a[1])
        msg = "Creating Graph for " + self.AC + "\n"
        print msg

      else:
        self.AC = ACselector
        self.name = ACState
        msg = "Turning " + self.AC + " " + self.name + '\n'
        print msg

        createReport(ACselector, ACState, Time)
        sendSerial(ACselector, ACState)

        msg = updateACs()
        for c in self.factory.clients:
          c.message(msg)

  def message(self, message):
    self.transport.write(message)


#---------------- Main ----------------#
print"\nMM.....DM..7MMMMMMMMMMN....MM
    ....."
print"MM.....MMM..........+MMM...MMM
    ~....."
print"MM.....MMMMM.....MM..MMNMM
    ....."
print"MM.....MMM..MM.....MM..MM. MMM
    ....."
print"MM.....MMD...MM.....MM=..MM...,MM
    ....."
```

```
print "MM.....MMM.....MM.....MMMMMMMMZ...MM....MMM.
    ..........."
print "MM.....DMM.....MM.....MMM..=.....MM.....MMM.
    ........."
print "MM.....MMI.....MM.....=MM.....MM.....~MM
    ~........"
print "MM.....MMM.....MM.....MM.....MM.....MM.....MMM
    ......."
print "MM.....DMM.....MM.....MMM.....MM.....MM.....MMM.
    ...."
print "MM.....MM7.....MM.....ZMM.....MM.....+MM=...
    ."
print "MMMMMMMMMMMM...MMM.....MM.....MM...MM.....MMM.
    ."
print "..........N.....,D.....N...8I.....,N.
    ."
print "......******....******...****.....**.********.******.....******...**.....
    ."
print "......**////**..**/////**./**/**...///**/////**/.**////**...**/////**./**.....
    ."
print "....**.....//..**.....//**/**//**..**/**.....**.....**...**..**.....//**/**.....
    ."
print "...../**........../**......./**/**.//**./**.....**.....**/******..**......./**/**.....
    ."
print "...../**........../**......./**/**..//**/**.....**.....**/**///**..**......./**/**.....
    ."
print "...../**.....**//**.....**./**/**...///**.....**.....**/**..//**.**.....**./**.....
    ."
print "....//******..//*******..../**/**....//***.....**.....**/**...//*******..../********
    ."
print "......//////....///////.....//..//.....///.....//.....//..//.....///////...////////.
    ."

print "\n\nLARA Control Server Started\n"

factory = Factory()
factory.protocol = LARAControl
factory.clients = []
reactor.listenTCP(80, factory)
reactor.run()
```

V. CÓDIGO-FONTE DOS MÓDULOS X-BEE

```
/*#
#  EndDevice.ino
#  X-Bee End Device
#
#  Created by Daniel Vilela on 15/08/12.
#  Copyright (c) 2012 UnB. All rights reserved.
#*/

#include <IRremote.h>

#define AC1
//#define AC2
//#define AC3

const int ledPin = 13; // the pin that the LED is attached to
int incomingByte;      // a variable to read incoming serial data into
int mask;

IRsend irsend;

// Defining ON IR code protocol
unsigned int ON_BUF[] = {4550 ,4300 ,700 ,1500 ,650 ,450 ,700 ,1450 ,700
,1550 ,650 ,450 ,700 ,450 ,700 ,1450 ,650 ,450 ,700 ,450 ,700 ,1450 ,650
,450 ,700 ,400 ,700 ,1500 ,700 ,1450 ,700 ,450 ,700 ,1450 ,650 ,450 ,700
,400 ,700 ,450 ,700 ,1450 ,700 ,1500 ,650 ,1500 ,650 ,1550 ,650 ,1500,650
,1500 ,700 ,1500 ,650 ,1500 ,700 ,400 ,750 ,350 ,750 ,400 ,700 ,450 ,650
,450 ,700 ,400 ,700 ,1500 ,700 ,1450 ,700 ,450 ,650 ,1500 ,700 ,400 ,750
,350 ,750 ,400 ,650 ,1500 ,700 ,400 ,750 ,400 ,650 ,1500 ,700 ,400 ,750
,1450 ,650 ,1500 ,750 ,1450 ,650 ,4700 ,4550 ,4300 ,700 ,1500 ,700 ,400
,700 ,1500 ,700 ,1450 ,700 ,500 ,650 ,450 ,650 ,1500 ,650 ,450 ,700 ,450
,650 ,1500 ,650 ,450 ,700 ,450 ,700 ,1450 ,650 ,1500 ,700 ,400 ,750 ,1450
,650 ,450 ,700 ,450 ,650 ,450 ,700 ,1450 ,700 ,1500 ,650 ,1500 ,700 ,1500
,650 ,1500 ,650 ,1550 ,700 ,1450 ,650 ,1500 ,700 ,400 ,750 ,400 ,700 ,400
,750 ,400 ,650 ,450 ,700 ,400 ,700 ,1500 ,650 ,1500 ,700 ,400 ,750 ,1450
,650 ,450 ,700 ,400 ,750 ,400 ,700 ,1450 ,700 ,450 ,650 ,400 ,750 ,1450
,650 ,450 ,700 ,1500 ,650 ,1500 ,700 ,1450 ,750 };


// Defining OFF IR code protocol
unsigned int OFF_BUF[] = {4550 ,4350 ,750 ,1400 ,700 ,450 ,700 ,1450 ,750
,1450 ,700 ,400 ,750 ,350 ,650 ,1550 ,650 ,450 ,700 ,400 ,750 ,1450 ,650
,450 ,650 ,450 ,750 ,1400 ,700 ,1500 ,700 ,400 ,700 ,1500 ,700 ,400 ,650
,1500 ,700 ,1500 ,700 ,1450 ,650 ,1550 ,700 ,400 ,650 ,1500 ,700 ,1450
,700 ,1500 ,700 ,400 ,750 ,350 ,700 ,450 ,700 ,400 ,700 ,1450 ,750 ,350
,750 ,400 ,700 ,1450 ,700 ,1500 ,650 ,1500 ,700 ,400 ,700 ,450 ,700 ,400
,700 ,400 ,750 ,350 ,700 ,400 ,750 ,400 ,700 ,400 ,750 ,1400 ,700 ,1500
,650 ,1500 ,750 ,1450 ,650 ,1500 ,750 ,4500 ,4600 ,4300 ,700 ,1500 ,650
```

```
,450 ,650 ,1500 ,750 ,1450 ,700 ,400 ,700 ,400 ,750 ,1450 ,650 ,450 ,650
,450 ,650 ,1500 ,700 ,450 ,650 ,500 ,750 ,1400 ,700 ,1500 ,700 ,400 ,700
,1500 ,650 ,450 ,700 ,1500 ,650 ,1500 ,700 ,1450 ,700 ,1500 ,700 ,400 ,700
,1450 ,750 ,1450 ,700 ,1450 ,700 ,400 ,750 ,400 ,700 ,400 ,750 ,350 ,750
,1450 ,650 ,450 ,700 ,400 ,750 ,1450 ,700 ,1450 ,700 ,1500 ,700 ,400 ,700
,400 ,700 ,400 ,750 ,400 ,700 ,400 ,700 ,400 ,700 ,400 ,700 ,400 ,750
,1450 ,700 ,1450 ,650 ,1500 ,700 ,1500 ,650 ,1550 ,700};


void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
}

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

    //Applying masks
    #ifdef AC2
      incomingByte = incomingByte >> 2;
    #else
      #ifdef AC3
        incomingByte = incomingByte >> 4;
      #endif
    #endif

    mask = incomingByte & 0b00000011;
    if((mask >> 1) == 1) {
       if((mask % 2) != 0) {
         digitalWrite(ledPin, HIGH);
         irsend.sendRaw(ON_BUF, 199, 38);
         delay(40);
       }
       else {
         digitalWrite(ledPin, LOW);
         irsend.sendRaw(OFF_BUF, 199, 38);
         delay(40);
       }
    }
  }
}
```

VI. DESCRIÇÃO DO CONTEÚDO DO CD

O CD contém todos os arquivos relacionados com o projeto, tais como códigos-fonte, logos, figuras, monografia, entre outros.

Deve-se observar que como o projeto foi desenvolvido em um computador mac, algumas incompatibilidades com windows podem ser encontradas (como impossibilidade de rodar o código do aplicativo iPhone em um computador Windows PC ou a apresentação do trabalho de graduação que foi feita no Keynote e não no Powerpoint).

Os arquivos são divididos em pastas de acordo com cada etapa do projeto ou arquivos auxiliares, sejam elas:

- Comunicação IR;
- iPhone;
- Servidor Python;
- Módulos X-Bee;
- Logos e figuras;
- Monografia.

Todos os programas utilizados no projeto são gratuitos.