

Universidade Paulista – UNIP

Campus Pinheiros

Ciências da Computação

AUTOMAÇÃO RESIDENCIAL:

Utilizando hardware e software livres

AGAIL CIMADON SANCHEZ

ALEXANDRE BIONDI ERMETTI

ANDERSON QUIEROZ

ERICK MAIA FURLAN

São Paulo - SP

2013

|

AGAIL CIMADON SANCHEZ

ALEXANDRE BIONDI ERMETTI

ANDERSON QUIEROZ

ERICK MAIA FURLAN

AUTOMAÇÃO RESIDENCIAL

Utilizando hardware e software livres

Trabalho de Conclusão de Curso apresentado
como exigência para obtenção do grau de
Bacharelado em Ciências da Computação da
Universidade Paulista – UNIP.

Orientador: Fernando Luz

São Paulo – SP

2013

FICHA CATALOGRÁFICA

A939 Sanchez, Agail Cimadon. et al.
Automação residencial: utilizando hardware e software livres /
Agail Cimadon Sanchez. et al. – São Paulo, SP: UNIP, 2013.
115 f.; il.; 28 cm

Orientador: Fernando Luz
Co-orientador: Roberto Leminski

Trabalho de Conclusão de Curso de Graduação - Apresentado
ao Instituto de Ciências Exatas e Tecnologia – Curso Ciência da
Computação da Universidade Paulista, Campus Pinheiros - São
Paulo/SP, 2013.

1. Domótica 2. Automação 3. Raspberry PI 4. Arduino 5.
Inteligente 6. Autonomia I. Ermetti, Alexandre Biondi II. Oliveira,
Anderson Queiroz de III. Furlan, Erick Maia IV. Título

CDU: 681.3

AGAIL CIMADON SANCHEZ
ALEXANDRE BIONDI ERMETTI
ANDERSON QUIEROZ
ERICK MAIA FURLAN

AUTOMAÇÃO RESIDENCIAL

Utilizando hardware e software livres

Trabalho de Conclusão de Curso apresentado
como exigência para obtenção do grau de
Bacharelado em Ciências da Computação da
Universidade Paulista – UNIP.

Aprovado em:

BANCA EXAMINADORA

_____/____/____

Professor Uanderson Celestino

Universidade Paulista – UNIP

_____/____/____

Professor Fernando Luz

Universidade Paulista – UNIP

_____/____/____

Roberto Leminski

Universidade Paulista UNIP

DEDICATÓRIA

Dedicamos esse trabalho a nossas famílias que sempre nos apoiaram e incentivaram a nunca desistir e a sempre acreditar que esse dia chegaria.

AGRADECIMENTOS

Agradecemos ao nosso orientador professor Fernando Luz, pela dedicação nas orientações prestadas durante a elaboração deste trabalho, sempre nos incentivando e colaborando no desenvolvimento, propondo melhorias e apontando erros.

"Nunca pense que um pequeno grupo de indivíduos altamente dedicados não pode mudar o mundo. Na verdade, foram eles os únicos que já o fizeram."

(Margaret Mead – Antropóloga)

RESUMO

Automação residencial (Domótica) é tudo o que sua casa pode fazer para você automaticamente para tornar a sua vida mais agradável ou mais produtiva. Para a maioria das pessoas a automação residencial começa e termina quando algum dispositivo familiar, como um Ar condicionado ou uma TV controlada por algo diferente de um botão no painel frontal ou do controle remoto original, mas esse conceito vai muito, além disso. Nos dias de hoje nossa sociedade tem uma grande preocupação com a economia energética aliada a questões ambientais e a acessibilidade, que são pensadas para tornarem as edificações acessíveis a deficientes físicos e idosos, proporcionando uma maior interação dos usuários e moradores. Tendo esse pano de fundo e imbuídos pelos conhecimentos adquiridos no curso de ciências da computação fomos à procura dessas “tecnologias” para que pudéssemos aprender e construir um sistema de automação. Durante essa procura nos deparamos com os projetos Raspberry PI e Arduino, dois grandes projetos que tem boa aceitação e muita documentação disponível. Foi nesse momento que surgiu a dúvida qual é o melhor? Qual utilizar? É possível utilizar os dois ao mesmo tempo? Essas questões nos motivaram a escrever este artigo, onde o intuito é avaliar a viabilidade da combinação do poder do Raspberry PI com a versatilidade do Arduino, ou ainda identificar dentro de nossa ideia inicial qual dos dois é mais indicado para a criação de um projeto de automação residencial.

Palavras-chave: Domótica, Automação, Raspberry PI, Arduino, Inteligente, Autonomia.

ABSTRACT

Home automation (domotics) is all that your home does for you automatically to make life more enjoyable and more productive, a smart home is one that seems to apply the intelligence to make it all happen. For most people, home automation begins and ends when a familiar device like a TV or air conditioning are controlled by something other than a button on the front panel or remote original. However, this concept goes far beyond that, today we have a major concern with the convenience and economy coupled with environmental and accessibility that make building accessible to the disabled and elderly people. Automation is very allied to robotics and informatics, in recent years a large movement of new technologies both commercial and free (open source) came on the market. This ease of access is considered a gateway for enthusiasts and students who want to learn more about the topic and build their own projects. How curious and imbued by the knowledge acquired in the course of computer science went looking these "technologies" so that we could learn and build an automation system. While searching we came across the projects Raspberry PI and Arduino, two great projects that have good acceptance and a lot of documentation available that was when the question arose what is the best? What should we use? You can use both at the same time. These issues motivated us to write article, intuited assess the feasibility of combining the power of the Raspberry PI with the versatility of the Arduino to create a project of home automation.

Keywords: Domotic, Automation, Raspberry PI, Arduino, Intelligent, Autonomy

LISTA DE ILUSTRAÇÕES

Figura 1 - Dispositivos X10	21
Figura 2 - Datasheet Arduino Uno	26
Figura 3 - Shields Arduino	29
Figura 4 - Componentes do Raspberry PI	32
Figura 5 - Esquema de montagem sensor de chuva	40
Figura 6 - Esquema de montagem servo motor.....	41
Figura 7 - Esquema de montagem dos sensores	44
Figura 8 - Esquema de montagem Buzzer	45
Figura 9 - Esquema de montagem sensor.....	49
Figura 10 - Esquema de montagem coolers	50
Figura 11 - Esquema de montagem do LDR	53
Figura 12 - Esquema de montagem dos LED RGB	54
Figura 13 - Carregar imagem Raspbian OS	58
Figura 14 - Montagem Raspberry PI.....	59
Figura 15 - Rasp. - Config.....	59
Figura 16 - Desktop do Raspbian	60
Figura 17 - Camada de Tarefas.....	70
Figura 19 - Interface de usuário.....	72

LISTA DE TABELAS

Tabela 1 - Lista de componentes do módulo de chuva.	39
Tabela 2 - Lista de componentes do módulo de alarme.	43
Tabela 3 - Lista de componentes do módulo Termostato.	47
Tabela 4 - Lista de componentes do módulo de Luminosidade.	52
Tabela 5 - Lista de componentes do módulo central.	56
Tabela 6 - Tabela de Usuários.	66
Tabela 7 - Tabela Módulos	67
Tabela 8 - Tabela de Tarefas.	67
Tabela 9 - Tabela condições.	67
Tabela 10 - Tabela horários.	68
Tabela 11 - Tabela de filas.	68
Tabela 12 - Tabela de logs.	68

LISTA DE ABREVIATURAS E SIGLAS

PC	Personal Computer (Computador Pessoal)
GNU	GNU is Not Unix (GNU Não é Unix)
FSF	Free Software Foundation (Fundação para o Software Livre)
CPU	Central Processing Unit (Unidade de Processamento Central)
ARM	Acorn RISC Machine
IDE	Integrated Development Environment
PIR	Passive Infrared
LED	Light Emitting Diode
LDR	Light Dependent Resistor
RAM	Random Access Memory
mAh	Miliampère-hora

SUMÁRIO

1. INTRODUÇÃO	15
1.1. Justificativa	17
1.2. Objetivos	17
1.2.1. Objetivo Geral.....	17
1.2.2. Objetivos Específicos.....	17
1.3. Estrutura Do Texto	18
2. AUTOMAÇÃO RESIDENCIAL.....	19
2.1. Primórdios	19
2.2. Novas Tecnologias.....	21
3. A REVOLUÇÃO DO OPEN HARDWARE	24
3.1. Arduino	26
3.1.1. Shields Arduino.....	28
3.2. Raspberry PI	31
3.2.1. Ponte para ligação do Arduino no Raspberry PI.....	33
4. PROJETO CASA INTELIGENTE “Smart Home”	36
4.1. Módulo Anti Chuva.....	38
4.1.1. Partes necessárias	39
4.1.2. Processo de Montagem.....	39
4.1.3. Funcionamento do Módulo	41

4.2. Módulo de Alarme	42
4.2.1. Partes Necessárias.....	42
4.2.2. Processo de Montagem	43
4.2.3. Funcionamento do Módulo	46
4.3. Módulo de Termostato.....	47
4.3.1. Partes Necessárias.....	47
4.3.2. Processo de Montagem	48
4.3.3. Funcionamento do Módulo	50
4.4. Módulo de Iluminação.....	51
4.4.1. Partes necessárias	52
4.4.2. Processo de Montagem	52
4.4.3. Funcionamento do Módulo	54
4.5. Central Raspberry PI.....	55
4.5.1. Partes Necessárias.....	56
4.5.2. Processo de Montagem (Configurando o Raspberry PI)	56
4.5.3. Programação do Módulo.....	60
4.5.4. Camada de Banco de Dados.....	66
4.5.5. Camada de Tarefas (Aplicação)	69
4.5.6. Camada de Filas.....	71
4.5.7. Camada de Interface	72
4.5.8. Funcionamento	73
5. CONCLUSÕES	77
REFERÊNCIAS	80
APÊNDICE A – CÓDIGO ARDUINO SENSORES.....	<u>83</u>82
APÊNDICE B – CÓDIGO ARDUINO ATUADORES	<u>85</u>84
APÊNDICE C – CÓDIGO CENTRAL RASPBERRY PI	<u>96</u>95

1. INTRODUÇÃO

Desde o surgimento dos primeiros computadores, a computação de ponta era acessível somente a órgãos governamentais, grandes corporações e renomados campi universitários (Filho, 2007). No entanto, com o lançamento do Computador Pessoal (do inglês Personal Computer - PC), aliado ao surgimento das redes de computadores, esse cenário começou a mudar.

Nos anos 80, surgiu no mercado computadores domésticos de baixo custo, como o ZX Spectrum¹ e o Commodore 64 (Angerhausen, 1984), permitindo que a computação pessoal e a tecnologia fossem inseridas pouco a pouco no cotidiano das pessoas.

Concorrentemente a este processo, em Setembro de 1983, Richard Matthew Stallman escreveu um comunicado na Arpanet (a rede predecessora da internet) dizendo que iria iniciar o projeto de desenvolvimento de um Sistema Operacional compatível com Unix denominado de GNU (sigla para GNU Não é Unix) (Stallman, 1983). Surge então o conceito de Software Livre, que consiste em uma maneira de os usuários serem livres para executar, copiar, distribuir, estudar, modificar e melhorar o software (Stallman, 2010). O Projeto GNU em 04 de Outubro de 1985 deu lugar a Free Software Foundation (FSF, Fundação para o Software Livre), que é uma organização sem fins lucrativos, que se dedica a eliminação de restrições sobre a cópia, redistribuição, estudo e modificação de programas de computadores (Free Software Foundation, 2013). O GNU influenciou de forma significativa varias comunidades de software a criarem seus próprios sistemas e programas livres.

Uma importante característica do software livre é o compartilhamento de código-fonte. Esse compartilhamento pode simplificar o desenvolvimento de novas aplicações, que não precisam ser programadas a partir do zero. Essa vantagem tem impacto significativo na redução de custos e na diminuição da duplicação de esforços. Além disso, o compartilhamento se refere à possível melhoria na qualidade devido ao maior número de desenvolvedores e usuários envolvidos com o software. Um maior número de desenvolvedores é capaz de identificar e corrigir mais bugs (falhas) em menos tempo e um número maior de usuários gera situações de uso e

necessidades variadas. Do ponto de vista econômico, o software livre promove o estabelecimento de vários fornecedores com base no mesmo software. A competição entre fornecedores traz vantagens aos usuários, como melhorias nos serviços de suporte e redução nos preços de pacotes (manuais, CDs, etc.). Cerca de 80% do dinheiro gasto com software pelas empresas são voltados para aplicações personalizadas e treinamento. Esse modelo de negócio (suporte e venda de pacotes) incentiva o surgimento de pequenas empresas que podem atender os mercados locais e conseqüentemente redução da dependência de empresas estrangeiras.

Com a força que o software livre deu para a informática como um todo, não demorou muito a surgir pessoas interessadas em desenvolver softwares para serem utilizados em construções, máquinas e até carros ditos “inteligentes”.

Atualmente, muitas construções são ditas inteligentes por utilizarem tecnologia para obter eficiência energética, menor emissão de poluentes ou ainda para proporcionar um maior conforto e segurança aos seus utilizadores. (Deboni, Engel, Alvarez, & Bissoli, 2011). Mas este tipo de tecnologia possui um custo elevado, tornando-se pouco acessível aos cidadãos comuns, pois geralmente o hardware empregado nesse tipo de automação são proprietários e de arquitetura fechada.

Uma mudança deste cenário foi iniciada em 2005, com o surgimento da placa de prototipagem rápida chamada Arduino (Schmidt, 2011), que possui a filosofia de livre acesso ao código-fonte do software e ao projeto de arquitetura do hardware.

Em 2009, David Mellis, um dos criadores do Arduino, deu uma entrevista para a Revista Info sobre Arduino, e ao ser questionado sobre as vantagens do hardware livre, ele deu a seguinte declaração:

Uma vantagem é a possibilidade de adaptar o modelo de negócios e o fornecimento a diferentes situações. Outras empresas podem vender kits para a montagem de dispositivos compatíveis com Arduino, por exemplo. Também podem redesenhar os produtos para trabalhar com componentes que são mais baratos e fáceis de conseguir em seus países. Um exemplo de produto derivado das placas Arduino que atende a um uso específico é a ArduPilot, placa de código aberto para navegação autônoma em aeronaves. (Grego, 2009)

Desta forma, o uso do Software Livre em conjunto com a facilidade de manuseio de hardware provido pelo Arduino, permite a realização de projetos de

automação com um custo mais acessível. Para comprovar esta hipótese, será confeccionado um projeto de automação domiciliar com o uso de hardware e software livres em uma maquete, provando uma opção de baixo custo em relação aos sistemas de automação disponíveis no mercado.

1.1. Justificativa

Justifica-se o estudo do tema proposto devido à vontade mutua dos integrantes em utilizar hardware e software livres no desenvolvimento do projeto e a experiência acadêmica do grupo. Tendo em mente que, uma das questões mais atuais é a forma que a informática e a computação pessoal estão presentes no dia a dia das pessoas, e como as mesmas estão ajudando a aumentar a comodidade nas tarefas diárias. Seguindo essa tendência, nossa proposta é utilizar hardware e software livres no desenvolvimento de um projeto de automação residencial voltado ao conforto e ao bem estar dos moradores. Visando automatizar rotinas, minimizar o desconforto de algumas atividades diárias e gerar maior qualidade de vida ao cotidiano dos moradores, além é claro de prover economia de recursos. Além disso, o uso desta automação será elaborado para poder ser utilizada por usuários que não tenham um conhecimento aprofundado de informática.

1.2. Objetivos

1.2.1. Objetivo Geral

O principal objetivo é desenvolvimento de um projeto de automação residencial utilizando equipamentos e programas de código aberto e livres de qualquer patente, com o uso de conhecimentos adquiridos durante a graduação e por meio de pesquisa científica.

1.2.2. Objetivos Específicos

- Desenvolver um sistema inteligente criado para gerir uma residência, automatizando algumas tarefas simples do dia a dia como, por exemplo, o

fechamento automático de janelas durante o início de uma tempestade.

- Exemplificar o funcionamento desse sistema, desenvolvendo uma maquete em escala reduzida, com o objetivo final de demonstrar a viabilidade da utilização de dispositivos simples e de baixo valor de aquisição.

1.3. Estrutura Do Texto

Antes de começarmos a descrever o hardware utilizado, os passos da montagem e a explicação do código, vamos estabelecer as fundações de nosso trabalho. No capítulo 2 vamos começar com o que exatamente entendemos por uma automação residencial, situar o leitor no contexto histórico e explicar por que a internet e o movimento open source são a base para nossa solução de automação feita em casa e por que isso faz sentido, hoje ou no futuro.

Em seguida no capítulo 3 vamos descrever o open hardware e apresentar os hardware escolhidos Arduino e Raspberry PI. No capítulo 4 vamos explicar como dividimos o sistema e relacionar os materiais e técnicas utilizados no decorrer do projeto, dando uma abordagem explicativa e didática sobre todas as etapas e conhecimentos necessários ao seu desenvolvimento. Finalmente no capítulo 5 vamos descrever o funcionamento de todo o sistema e situar nosso trabalho aonde chegamos, o que temos para melhorar e quais seriam os próximos passos.

2. AUTOMAÇÃO RESIDENCIAL

O que exatamente o termo automação residencial significa? No seu nível mais básico é um produto ou serviço que traz algum nível de ação ou interação do ambiente doméstico com o proprietário, sem haja uma intervenção direta do mesmo. O termo mais utilizado quando falamos de automação residencial é Domótica.

A palavra domótica deriva das palavras Domus (casa) e Robótica (controle automatizado de algo), ou seja, a domótica define-se como a possibilidade de controle de forma automática das nossas casas tornando-as que vulgarmente costumamos designar por “casas inteligentes”. (Alves & Mota, 2003)

A domótica é muito mais do que apenas um controle remoto para a TV, pode-se dar vários exemplos, tais como a programação de um PC para gravar um programa de televisão, o acionamento do Ar-Condicionado ou o acionamento de um sistema de alarme através da utilização de sensores. A automação residencial pode ser resumida como um mecanismo que remove ao máximo a interação humana em vários processos, que tecnicamente são possíveis e convenientes de serem substituídos por algum sistema eletromecânico.

Esses conceitos não são novos e a automação predial já era citada décadas antes de se tornar realidade. Alguns exemplos são os escritos de Ficção Científica do escritor HG Wells, história em quadrinhos e alguns desenhos animados, como os Jetsons. A seguir, será apresentado um pouco da história do desenvolvimento da automação residencial.

2.1. Primórdios

Em abril de 1968, a revista americana Popular Mechanics, deu destaque para uma matéria que apresentava um computador doméstico (Electronic Computing Home Operator (ECHO)) feito por Jim Sutherland um engenheiro da Westinghouse Electric. Ele foi feito a partir de um conjunto de peças eletrônica, e era utilizado para o cálculo das finanças domésticas, armazenamento das listas de compras, entre uma série de outras tarefas. Ainda podemos ler o artigo original da Popular

Mechanics online no Google books².

O ECHO nunca foi comercializado e ainda nos anos 60, amadores e uma série de grandes empresas brincaram com a ideia, porém foi nos anos 70, assim como com a computação pessoal, que a automação residencial viu o nascimento de uma nova era dessa tecnologia. Em 1975 a Pico Electronics, em parceria com a Birmingham Sound Reproducers introduziu o padrão X10 no mercado. O padrão X10 foi projetado para permitir que os transmissores e receptores utilizados em sistemas de fiação elétrica fossem controlados a distância por meio de mensagens de radiodifusão, possibilitando assim que comandos como "ligar" e "desligar" pudessem ser feitos via frequência de rádio. Três anos mais tarde, em 1978, os produtos X10 começaram a fazer sucesso em lojas voltadas para os entusiastas da eletrônica e nos anos 80, foi à vez do CP-290 interface para o computador Mattel Aquarius fazer o seu caminho no mercado. A unidade CP-290 permitia que computadores se comunicassem com equipamentos compatíveis com o X10. Ao longo dos anos, foi incluído suporte para Windows e Mac, o que deu aos interessados em automação residencial a capacidade de programar seus sistemas de iluminação, termostatos, e portas de garagem a partir de seus computadores domésticos. Estava introduzida a febre da automação residencial.

O X10 foi realmente revolucionário, mas infelizmente tinha algumas falhas:

- Problemas de isolamento e interferência;
- Alguns comandos eram perdidos na transmissão;
- O escopo de suporte aos produtos X10 era limitado;
- O escopo de comandos disponíveis era limitado;
- Baixa velocidade de transmissão de sinal;
- Falta de criptografia;
- Falta de mensagem de confirmação.

Figura 1 - Dispositivos X10



Fonte: [http://en.wikipedia.org/wiki/X10_\(industry_standard\)](http://en.wikipedia.org/wiki/X10_(industry_standard)).

Com essas limitações e como toda a inteligência era proprietária, a domótica caseira ainda não tinha entrado no mercado doméstico em grande escala. No entanto os anos 90 trouxeram vários avanços tecnológicos advindos do boom das empresas Dot.com e de fundações de software livre. As mesmas estavam fornecendo um novo conjunto de ferramentas, protocolos e padrões que resolviam muitas das falhas que o padrão X10 tinha como limitação.

2.2. Novas Tecnologias

Com a explosão da web nossas casas foram invadidas com vários equipamentos de rede e computação antes restritos aos ambientes acadêmicos e profissionais. Com o barateamento desses componentes hoje a maioria de nós tem acesso a equipamentos que interligado podem fornecer aos entusiastas da domótica o que eles estavam esperando.

As Redes residenciais em especial o protocolo Wi-Fi permite que computadores e aparelhos eletrônicos se comunicassem uns com os outros em uma casa sem a necessidade de quebrar paredes e passar cabos. Enquanto protocolos como FTP e HTTP se tornavam o padrão de acesso a informações através da Internet, alguns desenvolvedores de hardware viram a oportunidade de aproveitar essas tecnologias de comunicação em dispositivos de código aberto. Diferente dos aparelhos X10, as tecnologias da Web fornecem um framework inteiro para lidar com códigos de erro e mensagens.

Aproximadamente ao mesmo tempo em que a plataforma Arduino estava sendo desenvolvida, os primeiros tablets estavam começando a ser liberados no mercado. De 2005 até agora, houve uma explosão de dispositivos portáteis, celulares, tablets e smartphones. Este crescimento tem sido comumente referenciado como a era "pós-PC". Estes dispositivos têm proporcionado plataformas de computação móvel que pode executar um software complexo e pequeno o suficiente para caber no bolso do usuário. Em resultado disso, as aplicações têm sido desenvolvidos para os sistema IOS (iPhone), Windows Phone e Android que permitem ao utilizador controlar vários aparelhos eletrônicos de consumo, tais como a TV da sala ou algumas máquinas de lavar.

Devido ao seu tamanho, portabilidade e em alguns casos baixo custo eles têm se tornado a plataforma ideal para fazer interface com aparelhos domésticos e equipamentos e fornecem uma mídia que o usuário está familiarizado.

Ao lado da explosão em hardware, também houve uma explosão equivalente em software. Em especial o sistema operacional de código aberto Android.

O sistema operacional Android é um sistema operacional baseado em Linux voltado para dispositivos móveis. Como parte do Open Handset Alliance - um consórcio de 84 empresas que operam na esfera móvel, seu desenvolvimento foi apoiado pelo Google que adquiriu o sistema operacional móvel.

O objetivo foi criar um sistema operacional de código aberto que possa competir com empresas como a Apple, oferecendo um sistema robusto e que possa funcionar em vários dispositivos independente do fabricante. Como resultado disso, os fabricantes de eletrodomésticos comerciais começaram a incorporar a tecnologia e software nos seus produtos, e uma novíssima geração de "smart-devices" começou a aparecer nas lojas de todo o mundo.

Produtos como o refrigerador RF4289HARS da Samsung rodando Android e a máquina de lavar WT6001HV da LG estão rapidamente pavimentando o caminho para o futuro das casas inteligentes, abraçando tecnologias open-source, e se conectando cada vez mais a rede, graças a processadores como o ARM (destinado a pequenos aparelhos).

E também não são apenas os eletrodomésticos que estão recebendo esta

"reforma". Sistemas de termostato como o "Nest"³ de uma empresa fundada por ex-funcionários da Apple que estão repensando como termostatos inteligentes devem trabalhar.

Os códigos de barras e códigos QR disponível em vários produtos permiti que o consumidor possa digitalizá-los em seu smartphone e baixar informações diretamente da web fornecendo detalhes sobre o item, sua utilização, aplicações, etc. Isso pode ser estendido para permitir a gestão dos estoques de produtos em casa, informações diversas sobre os produtos, tais: como consumir, datas de vencimento dos produtos na geladeira, etc. É possível ainda gerar dinamicamente lista de compras para fazer um pedido em lojas online que entregam na porta de casa.

Esta combinação de hardware, software e informação oferece o potencial para a casa de hoje se tornar parte da "Internet das Coisas"⁴.

Graças ao código-fonte aberto e as tecnologias de padrão aberto utilizados nestes dispositivos, é fácil combinar projetos de Homebrew construídos com o Open Hardware e produtos comerciais para construir uma casa inteligente, criando uma rede de dispositivos que podem se comunicar uns com os outros para combinar a execução de tarefas.

3. A REVOLUÇÃO DO OPEN HARDWARE

Com a chegada da web, um conjunto de tecnologias de código aberto foi compartilhada na rede, fomentando a adoção do software livre e aumentando a demanda de hardwares compatíveis. Aliados a isso muitos projetos que visam forçar as grandes empresas de equipamentos a documentar e expor as informações de seus hardwares proprietários para que os mesmos pudessem portados ou o sistema fosse adaptado para suporta-los foram surgindo.

Ainda em 1997, Bruce Perens o criador do Open Source Definition, co-fundador do Open Source Initiative, e radialista⁵ amador lançou o Programa de Certificação de Hardware Livre⁶. O objetivo do programa é permitir que os fabricantes de hardware certificassem seus produtos como livre e acessíveis, isto implica em garantir a disponibilidade de documentação para a programação da interface do driver de um dispositivo de hardware específico.

O programa é grátis e sem taxas para as empresas o objetivo é proporcionar aos equipamentos certificados o direito de aplicar um logotipo do Open Hardware em sua embalagem e em seus postos de vendas. A ideia é fornecer ao consumidor a certeza de que uma mudança no sistema operacional ou até mesmo o fim da fabricação do dispositivo não o tornaria inutilizável, pois um novo software poderia ser escrito para seu dispositivo.

Essa iniciativa permitiu que muitos programadores amadores e grupos reunidos pelo mundo todos pudessem combinar o poder do PC, a comunicação e as tecnologias de multimídia da web, com capacidade de interagir com o ambiente de um micro controlador, e a portabilidade de um dispositivo móvel. Isso oferece o conjunto perfeito de fatores que nos permitem construir dispositivos baratos para nossas casas que podem interagir com os dispositivos comerciais, mas que podem ser adaptado para as nossas próprias necessidades, proporcionando também uma ótima ferramenta para aprender sobre tecnologia.

As origens do Raspberry PI e do Arduino estão em muitos aspectos ligados a essa evolução. Em 2006 um grupo de cientistas da computação liderados por Eben

Upton do Laboratório de Informática da Universidade de Cambridge, apostaram na ideia de produzir um microcomputador com objetivo educacional, que fosse barato e voltado para os entusiastas amadores de computação, alunos de graduação, e curiosos de plantão⁷.

O objetivo era fornecer suporte para o aprendizado de futuros alunos de graduação em Ciência da Computação. Esse objetivo foi alcançado graças a projetos como do Open Hardware e principalmente os avanços na área de computação móvel - que em grande parte foi impulsionada pela indústria da telefonia móvel. Com a criação da CPU (Central Process Unit) ARM (Acorn RISC Machine), Eben Upton projetou vários conceitos de microcomputador até que em 2008, graças às características dos chips ARM (que ocupam pouco espaço e tem baixo consumo de energia), o chip ARM1176JZF-S foi o escolhido. Assim, a Raspberry PI Foundation começou a fabricar o microcomputador Raspberry PI⁸. Até 2011, os primeiros modelos Alpha ainda estavam sendo desenvolvido, mas finalmente em 2012 o público conseguiu ver do que o Raspberry PI é capaz.

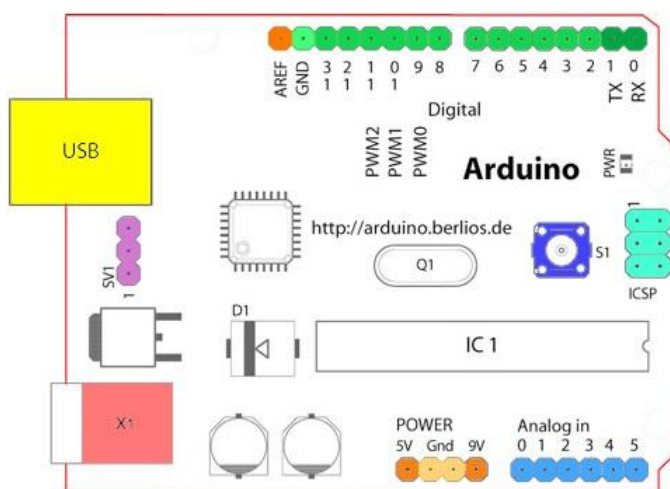
Paralelamente a todos esses acontecimentos, um professor chamado Massimo Banzi, que ensinava eletrônica e programação de computadores no Interaction Design Institute Ivrea, decidiu criar sua própria placa de circuito impresso micro controlada para aulas de Design, nascia assim o Arduino. O intuito de Massimo era criar uma opção barata em relação a outros sistemas de prototipagem disponíveis na época. Para a conclusão do Projeto Arduino Massimo contou com a ajuda de seu aluno David Cuartielles e outros especialistas em programação, Daves Mellis, Tom Igoe e Gianluco Martino. Foi criado um ambiente de desenvolvimento integrado, que pode pegar uma linguagem de programação de alto nível como o C, e passar para a linguagem de máquina que o Arduino entende. Assim qualquer pessoa pode utilizar um Arduino para criar um programa e ver o resultado de imediato, podendo ainda interagir com o ambiente agregando atuadores como leds, sensores, motores e etc.. A única exigência é a de ser fácil, para introduzir a programação a artistas e pessoas não familiarizadas com o desenvolvimento de software. A partir daí várias pessoas ao redor do mundo começaram a utilizar o Arduino e a fazer coisas incríveis com ele, surgindo assim

uma febre mundial da eletrônica caseira. O ápice do seu sucesso foi o recebimento de uma menção honrosa na categoria Comunidades Digitais em 2006, pela Prix Ars Electronic⁹, além de atingir a marca de mais de 50.000 placas vendidas até outubro de 2008. Considerando estas características, muitos projetos de automação começaram a surgir com a utilização de hardwares livres, primeiramente para o mercado residencial e hoje vemos muitos projetos que visam fins lucrativos.

3.1. Arduino

A plataforma Arduino permite ao usuário criar hardware e software personalizados para controlá-lo através de sua linguagem de programação própria. Atualmente existem vários modelos de placas Arduino no mercado que variam no tamanho e em seus componentes. O benefício para entusiastas e amadores é que pouco ou nenhum conhecimento sobre como componentes eletrônicos soldados é necessário. Vamos conhecer a anatomia do Arduino:

Figura 2 - Datasheet Arduino Uno



Fonte: (<http://playground.arduino.cc/Code/2B>).

Iniciando no sentido horário da parte superior:

- Pino de Referência analógico (laranja)
- Terra Digital (verde claro)
- Pinos digitais 2-13 (verde)
- Pinos digitais 0-1/Serial In/Out - TX/RX (verde escuro)
- Botão Reset - S1 (azul escuro)

- Programador interno de circuito Serial (azul piscina)
- Pinos Analógicos 0-5 (azul claro)
- Pinos de Alimentação e Terra (energia: laranja, terras: laranja claro).
- Fonte de alimentação externa (9-12VDC) - X1 (rosa)
- Alterna entre alimentação externa e Alimentação USB - SV1 (roxo)
- USB (usados para comunicação serial) (amarelo).

A base de funcionamento do Arduino é composta por duas partes principais, um Hardware, composto por um conjunto de componentes eletrônicos, montados em cima de uma placa de circuito impresso, que é a base para os protótipos. E o Software, o bootloader, que fica na memória de programas, embarcado no Arduino. É em tal memória que ficam os programas feitos pelo usuário, para execução. No desenvolvimento dos programas para serem carregados no Arduino, o usuário necessita de um IDE (Integrated Development Environment ou Ambiente Gráfico de Desenvolvimento Integrado). Após o desenvolvimento o usuário carrega o programa no Arduino, utilizando uma transferência feita via conexão USB, dependendo do modelo do Arduino, esta mesma conexão pode ser feita por comunicação serial ou ethernet. Os programas feitos na linguagem de programação Processing baseada na linguagem C própria para Arduino, quando chegam ao hardware se transformam em linguagem de máquina, que diz ao microcontrolador que ele tem que fazer.

Como já citado antes o Hardware do Arduino é baseado em microcontroladores, sendo estes microcontroladores AVR da empresa Atmel, dependendo do microcontrolador utilizado o Arduino recebe um codinome em italiano, em particular os Arduino usam os modelos, ATmega8, ATmega168, ATmega328 e o ATmega1280. Para se interligar com outros dispositivos, em sua base existem dois conjuntos de conectores, um com 14 pinos para entradas ou saídas de sinais digitais e um de 6 pinos para entrada de sinais analógicos. Internamente Ports são as conexões físicas entre CPU e conexões externas, Ports são grupos de 8 pinos do microcontrolador que são fisicamente conectados os registradores de 8 bits no circuito digital de E/S o por onde podemos enviar ou receber qualquer combinação de níveis lógicos na forma de 1 byte. Cada Port tem dois registradores, o registrador de direção de dados DDR e o registrador de dados propriamente dito ou simplesmente PORT. Quem comanda todas as linhas de controle tanto do DDR quanto no PORT é o Decodificador de Instruções que fica na

CPU do microcontrolador.

O ATMEGA328 possui um conversor analógico digital (A/D) de 10 bits de resolução no Port C com seis entradas que pode receber qualquer nível de tensão entre 0 e 5 volts. Já no Arduino essas entradas analógicas são os pinos A0 a A5. Na geração de voltagens analógicas de 0 a 5 volts em alguns de seus pinos de saídas digitais o Atmega328 emprega uma técnica conhecida por PWN ou Modulação por Largura de Pulso, no Arduino esses pinos PWN são os pinos digitais 3, 5, 6, 9, 10 e 11. Hoje o Arduino é muito conhecido e já existem vários projetos similares modificados, que são os chamados Forks, como por exemplo, o LilyPad Arduino, que tem como foco o ambiente têxtil. Hoje se pode encontrar Arduino dependendo da sua necessidade, modelos como mini, Uno e Mega. Tendo como diferença quantidade de portas digitais e analógicas, e o poder de processamento. Nesse mesmo enfoque temos os "Shields Arduino". Os Shields (escudo em inglês) são placas que podem ser conectadas em cima do PCB do Arduino estendendo as suas capacidades. Os Shields seguem a mesma filosofia, como o kit de ferramentas original: eles são fáceis de montar e baratos para se produzir.

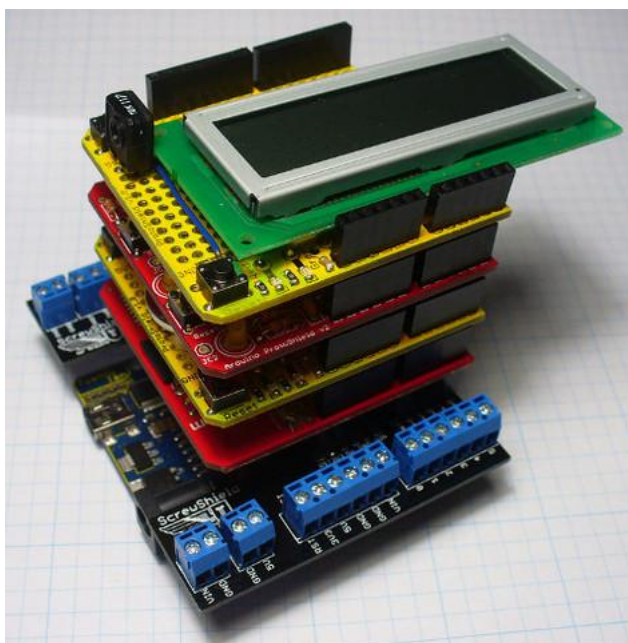
3.1.1. Shields Arduino

Assim como computadores de mesa têm "slots de expansão" em que você pode adicionar placas de expansão para vídeo, som, rede e muitas outras coisas, placas Arduino tem "Pinos" onde você pode instalar "Shields" para adicionar outras funcionalidades. A grande diferença, porém, é que os slots de expansão do computador são finitos e independentes: se o seu computador tem 6 slots, você pode colocar até 6 placas e cada placa é totalmente independente, não importa o que você usa. Com os Shields isso não é tão simples, não existe um número específico de "slots", e as placas podem ser literalmente empilhadas umas sobre as outras para combinar suas características. Na maioria das vezes só utilizamos um escudo de cada vez, mas alguns "Heavy User's" realmente querem testar os limites!

Às vezes Shields podem ser empilhados juntos e às vezes não, e nem sempre é possível que eles trabalhem juntos. Então, como sabemos se são compatíveis uns com os outros? Todos os Shields, exceto os que foram projetados para ficar no topo admitem o empilhamento. Muitos Shields estão equipados com

Pinos machos na parte inferior das placas, e oferecem um conector fêmea no topo, onde outro Shield pode ser plugado. O empilhamento de Shields torna nossa vida muito mais fácil. Podemos observar o empilhamento de vários Shields na foto abaixo com destaque para o Shield com um LCD que é um dos poucos tipos que não admite outro empilhamento sobre ele. Podemos observar também que as dimensões dos Shields podem variar dependendo de seus componentes, um bom exemplo é o Shield Ethernet isso porque o conector RJ45 é fisicamente elevado sendo necessário utilizar um extensor de pinos para que seja possível o empilhamento.

Figura 3 - Shields Arduino



Fonte: (<http://tronixstuff.com/>).

Um pequeno problema é com alguns Shields como os baseados no chipset W5100 Wiznet, que inclui o Shield oficial de Ethernet e muitos Shields de terceiros. O W5100 tem uma falha de projeto e faz com que ele mantenha o controle do barramento SPI, mesmo quando outro dispositivo foi selecionado, o que torna muito difícil de combinar o Shield oficial Ethernet com outros Shields que usam SPI. O Shield Ethernet da Freetronics usa o chipset W5100, mas inclui uma correção para que ele não tenha esse problema, mas é preciso cuidado ao escolher os Shields para que os mesmos funcionem juntos.

Outro ponto a ser observado são os softwares contidos nos Shields (como bibliotecas, drivers, etc.) às vezes pode ser uma dor de cabeça combina-los, principalmente se eles usam muita memória Flash ou SRAM, se tem interrupções conflitantes ou requisitos de tempo apertados. Isso é algo que pode ser difícil para

determinarmos só pelas informações fornecidas, a menos que saibamos os detalhes internos de todas as bibliotecas envolvidas. Na maioria das vezes a tentativa e erro é utilizada, por isso o apoio da comunidade é fundamental e evita muitos problemas.

Os cuidados ainda devem ser estendidos à alimentação elétrica, o Arduino só tem acesso a cerca de 500 mA, seja através da conexão USB ou de um fonte de alimentação externa. Isso deixa uma quantidade relativamente baixa de energia disponível para compartilhar com uma quantidade grande de Shields, por isso, se temos Shields que necessitem de uma grande quantidade de energia, podemos ter problemas ao combiná-los. Muitas vezes o Shield que consome muita energia não é evidente, mas os culpados mais comuns são Shields LCD (alguns com backlights pode consumir até 300mA!) e Shields sem fio (Wireless, Bluetooth).

Devemos ainda ter especial cuidado com os escudos que dependem do fornecimento de 3.3V, porque tem um limite muito baixo de corrente de apenas 50 mA. Shields que requerem uma grande quantidade de corrente em 3.3V (como o Shield ZigBee) devem implementar seu próprio regulador on-board de tensão em vez de depender do fornecimento 3.3V do Arduino.

Se seus Shields usam qualquer forma de comunicação via RF (Rádio Frequência) é possível que eles possam interferir uns com os outros. Por exemplo, Shields Wi-Fi contém um receptor de comunicação de cerca de 2,4 GHz, por isso, se você tem outro dispositivo usando 2.4GHz por perto é provável que venhamos a ter problemas.

O ruído elétrico pode também ser um problema. A situação piora caso seja necessário combinar um Shields que precisa de um ambiente elétrico extremamente limpo (como um conversor analógico-digital) com um Shield que faz muito ruído elétrico, como um Shields que utiliza algum tipo de motor. Cuidados na montagem da fonte de alimentação, associação de capacitores extras para cada placa, muitas vezes pode ajudar se o ruído elétrico é um problema.

Os Shields estão longe de ser um problema, mas conforme o explanado devem ser utilizados com moderação e inteligência, pois podem inviabilizar o projeto ou ainda oferecer muitos efeitos colaterais. É muito indicado pesquisar e ver se alguém já testou e se teve algum problema antes de incluir um novo em seu projeto.

3.2. Raspberry PI

Raspberry PI é um computador do tamanho de um cartão de crédito desenvolvido no Reino Unido pela Fundação Raspberry PI. Todo o hardware é integrado em uma única placa. O objetivo principal é de estimular o ensino de ciência da computação básica em escolas.

Mas porque Raspberry PI? A explicação é que nos primeiros anos dos computadores pessoais muitos desses aparelhos foram nomeados com nomes de frutas em inglês (Ex.: Apple, Apricot, Tangerine...) por isso o nome Raspberry (Framboesa em português) foi sugerido e o PI fica por conta da linguagem de programação Python, que era utilizada nas versões precoce em sugestões para uma linguagem "oficial" adequada para a Raspberry PI.

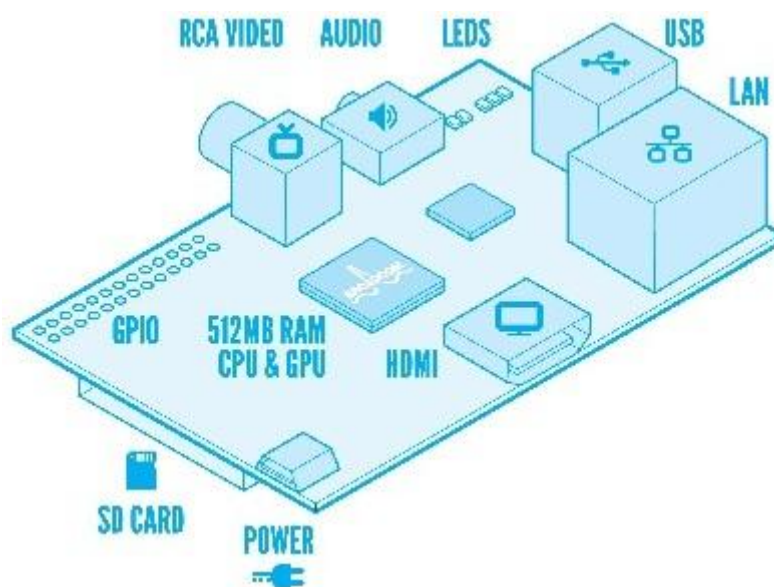
O Raspberry PI é um pequeno dispositivo que mede 85,60 milímetros x 53,98 milímetros x 17 milímetros e pesa apenas 45g. O seu tamanho reduzido o torna perfeito para automação residencial, pois esse pequeno dispositivo pode ser colocado em uma pequena caixa elétrica, ou substituindo um dispositivo de termostato já existente na parede por exemplo. O Raspberry PI foi construído em uma placa de circuito impresso que contém em sua parte traseira o chip Broadcom BCM2835. O BCM2835 é um processador de aplicações multimídia, que foi desenvolvido pela Broadcom Corp para dispositivos móveis e sistemas embarcados que inclui um processador ARM1176JZF-S de 700 MHz, uma GPU VideoCore IV, e 512 MB de memória RAM na versão B do Raspberry PI em versões mais recentes. Esta não é uma quantidade enorme de memória RAM, e muito menos do que você esperaria em um PC comum, onde a memória RAM está disponível na casa dos gigabytes. No entanto, para o tipo de aplicações que serão de construídas, 256 MB ou 512 MB de memória RAM será mais do que suficiente.

A CPU ARM1176JZF-S de 32 bits multi-processado rodando a 700 MHz é o componente principal do Raspberry PI, e responsável pela execução das instruções de um programa de computador através de operações matemáticas e lógicas. O Raspberry PI está bem equipado com o ARM série 11 igualmente a grandes aparelhos tais como iPhone, Kindle (da Amazon), e o Samsung Galaxy.

A GPU é um chip especialmente projetado para acelerar a geração e

manipulação de cálculos de imagens. No caso do Raspberry PI, ele vem equipado com um chip Broadcom VideoCore IV capaz de reproduzir Vídeos em qualidade HD e com capacidade de aceleração de hardware com suporte a tecnologia OpenGL. É especialmente útil se você deseja rodar jogos ou assistir vídeos. A porta Ethernet é a principal porta de entrada do Raspberry PI para se comunicar com outros dispositivos e com a Internet. Você será capaz de usar a porta Ethernet para conectar o Raspberry PI em um roteador, como o que você usa para acessar a Internet ou um switch de rede, se houver uma rede local. O projeto não inclui uma memória não volátil ou como a maioria conhece um disco rígido, mas possui uma entrada de cartão SD para o armazenamento de dados.

Figura 4 - Componentes do Raspberry PI



Fonte: Raspberrypi.org

Conforme visto na imagem acima temos um conector de áudio analógico de 3,5 milímetros que permite conectar fones de ouvido ou alto-falantes ao Raspberry PI, sendo útil para projetos baseados em áudio ou para Medias Center. Outro conector comum em nosso dia a dia é o RCA ele é designado para conectar cabos compostos usados geralmente para ligar o leitor de DVD na TV. Eles geralmente vêm com três plugues: vermelho, branco e amarelo. O Raspberry PI tem apenas uma porta RCA a qual serve para conectar o cabo de vídeo (amarelo) em uma TV, permitindo assim usar a TV como um monitor. Além disso, temos o conector High Definition Multi-media Interface (HDMI) ele permite que o Raspberry PI possa ser ligado aos televisores de alta definição e monitores que suportam essa tecnologia. Isto fornece uma opção adicional para a porta RCA para vídeo composto e que

também suporta áudio. Se o objetivo principal for transmitir áudio e vídeo a partir de do Raspberry PI para a TV, está com toda a certeza é a melhor escolha. Na parte de expansões o Raspberry PI possui duas portas USB 2.0, além de uma porta micro USB. O USB é um dos conectores mais comuns nos dias de hoje, geralmente é utilizado para a conexão de periféricos e de dispositivos de armazenamento. O Raspberry PI vem equipado com dois deles, o que permite ligar teclado, mouse, dispositivos de armazenamento externo e uma gama de acessórios compatíveis com essa tecnologia, à porta micro USB é utilizada para ligar o dispositivo a uma fonte de energia.

Os pinos de entrada/saída (GPIO) do Raspberry PI são a principal maneira de se conectar com outras placas eletrônicas, inclusive com o Arduino. General Purpose Input/Output (GPIO) são basicamente portas programáveis de entrada e saída de dados. São utilizadas para prover uma interface entre os periféricos e os microcontroladores/microprocessadores. Muitas vezes desenvolvemos sistemas baseados em um Circuito integrado, e precisamos de portas de sinalização extra, que não estão disponíveis por padrão no Circuito integrado. Tal como o nome sugere, os pinos GPIO podem aceitar comandos de entrada e saída e, assim, podem ser programados no Raspberry PI. Isto é útil em projetos de automação residencial, onde pode ser utilizado para armazenar dados de sensores ou manipular motores com base em um programa em execução no sistema operacional do Raspberry PI. Depois de termos abordado as capacidades técnicas do Raspberry PI, vamos agora olhar para o Arduino e na sequência vamos conhecer os Shields Arduino, uma maneira de conectar as duas tecnologias através dos pinos GPIO.

3.2.1. Ponte para ligação do Arduino no Raspberry PI

Conforme vimos os Shields Arduino são uma forma de expansão para contornar as limitações do Arduino padrão, mas para termos ainda mais possibilidades é possível aliar a versatilidade do Arduino com o poder do Raspberry PI. A ideia por trás do Shield do Arduino para Raspberry PI é criar uma ponte de ligação que permite usar qualquer um dos Shields, placas e módulos projetados para Arduino no Raspberry PI. Ela inclui também a possibilidade de conectar sensores analógicos e digitais, usando a mesma pinagem do Arduino, mas com o

poder e as capacidades do Raspberry PI. A fim de criar a maior compatibilidade possível entre eles foi criado a biblioteca arduPi que permite a utilização do Raspberry PI com o mesmo código utilizado no Arduino. Para isso, foram implementadas funções de conversão, de modo que podemos controlar (da mesma forma) como no Arduino todas as interfaces I / O: I2C, SPI, UART, analógicas e digitais no Raspberry PI.

Para resumir os Shields Arduino que podem ser utilizados no Raspberry PI juntamente com a biblioteca arduPi são:

- Módulos sem fio Arduino (XBee, RFID, NFC, Bluetooth, Bluetooth Pro, Wi-Fi, GPRS, 3G);
- Sensores (0-5V analógico, digital);
- Shields Arduino (CanBus, Relés,...);
- Módulos eletrônicos/atuadores através das portas I2C, SPI, UART.

Como falado antes para poder trabalhar com o Shield a instalação da biblioteca arduPi é necessária, ela é feita em C e aceita comandos da linguagem de programação Processing, utilizada no Arduino para poder as funções de um atuador ou sensor. Para a sua utilização como a mesma é feita em C precisamos escrever o código e compila-lo. Após a compilação é gerado um executável que quando ativo executara os comandos da linguagem Arduino e se comunicara com o Shield Arduino, como se fosse um programa nativo Arduino.

Outra forma de integrar o Arduino ao Raspberry PI é através da conexão USB, ligando diretamente o Arduino no Raspberry PI ao invés da utilização de Shields.

Por exemplo, em nosso projeto não utilizaremos nenhum Shield, os Arduino serão interligados ao Raspberry PI através das portas USB via conexão serial a qual será provida pelo protocolo RFC232 presente no pacote ArduPi o qual para ser utilizado necessita da linguagem de programação Python. Python é uma linguagem de programação de alto nível desenvolvida no final dos anos 80 por Guido van Rossum em homenagem ao popular programa de comédia Monty Python Flying Circus¹⁰. Ela foi escolhida por ser utilizada nativamente no Raspberry PI e altamente recomendada para desenvolver projetos para o mesmo, pois além de ter uma vasta

documentação, ser de fácil entendimento, ter um bom desempenho de execução permite construir aplicações web, conexão com banco de dados e o uso da conexão serial. Com a conexão serial vamos criar um canal de comunicação interligando os dois Arduino ao Raspberry PI. Um Arduino informara os valores capturados pelos sensores ao Raspberry PI que segundo a programação tomará uma ação, e essa ação pode levar a ativação de um atuador no outro Arduino (um LED, por exemplo). A seguir vamos explicar melhor como o sistema será concebido.

4. PROJETO CASA INTELIGENTE “ARDUPI SMART HOME”

O site <http://www.significados.com.br> define escopo como: "escopo é o objetivo que se pretende atingir, é o fim o propósito, o desígnio. Em projetos pode-se entender também como o limite de, ou o que vai atingir uma determinada operação dentro da qual os recursos de sistema podem ser utilizados.". Visando facilitar a sua criação o projeto será dividido em módulos (partes de um mesmo sistema), podendo ser utilizados em separado ou interligados para juntos fornecerem um completo sistema de automação doméstica. Segundo a definição o escopo deste trabalho é a criação de um sistema, dividido em módulos, onde cada módulo corresponde a uma área de automação específica. Cada módulo deverá ser executado através de uma transação única sem interferir nos demais, por essa diretriz decidimos separar os sensores e os atuadores em dois Arduino diferentes e utilizar o Raspberry PI como uma ponte de ligação entre eles. Embora o Arduino e Raspberry PI sejam semelhantes fisicamente existe uma grande diferença entre eles, o Arduino é uma placa de prototipagem equipada com um microcontrolador. O Microcontrolador é um computador compactado com características e funcionalidades específicas no caso do Arduino ele controla portas analógicas e digitais programáveis, a programação dessas portas são armazenadas em uma memória somente de leitura, porém a mesma é limitada e isso é um problema para desenvolver um sistema complexo. Já o Raspberry PI pode ser descrito como um microcomputador convencional, com todas as características de um: multitarefa, sistema operacional, dispositivos de entrada e saída (como monitor, teclado e mouse), etc. Foi com base nessas características que escolhemos o Raspberry PI, para ser o nosso “cérebro”, pois com a facilidade de utilização que um computador pode oferecer aliada a versatilidade do Arduino podíamos facilmente trazer o sistema a vida. Com base nesses fatores procuramos estabelecer um padrão de comandos a serem transmitidos através de uma interface serial, optando assim pelo modelo cliente servidor. O Arduino como cliente fica encarregado de capturar os dados dos sensores convertendo estes dados em string, repassar os mesmos ao servidor Raspberry PI que após processá-los envia comandos para acionar os atuadores ligados ao outro cliente Arduino. Basicamente o Raspberry PI processa os dados coletados nos sensores e de acordo com a sua programação ativa ou não os

atuadores.

Nesse primeiro momento vamos nos ater a apenas quatro módulos, que foram pensados para demonstrar o poder dos componentes escolhidos e que os mesmos pudessem conviver sem interferência mutua entre eles. Sendo assim os módulos escolhidos foram:

Módulo Anti Chuva: Esse módulo foi pensado para evitar a correria dos moradores de uma casa com muitas janelas na ocasião de uma chuva de verão (que por vezes nos pega de surpresa) ou ainda ao deixar o apartamento, por exemplo, esqueceu-se de fechar a janela e foi surpreendido com uma chuva repentina. Nossa ideia nesse caso é utilizar um sensor de umidade que ao detectar o início da chuva envia o sinal para que um servo motor devidamente instalado nas janelas da casa faça o seu fechamento de maneira automática evitando o transtorno de termos nossa casa molhada e nossas cortinas ensopadas.

Módulo Termostato: Nas cidades cada vez mais sem arvores e com grande poluição o aumento da temperatura não é nenhuma novidade, tanto que a utilização do ar-condicionado já virou quase que uma unanimidade nos lares urbanos. O que leva a um pequeno problema que é o motivo da eterna briga de homens e mulheres qual é a temperatura ideal do ar-condicionado? Os ambientes de trabalho devem atender ao disposto no subitem 17.5.2 da NR-17, obedecendo-se, no mínimo, o índice de temperatura efetiva entre 20° e 23°C. Para facilitar decidimos usar um Termostato para avaliar a temperatura do ambiente e automaticamente e ajustar a temperatura do ar-condicionado.

Módulo de Iluminação: Quem nunca deu uma topada com o dedão na quina do sofá tentando encontrar o interruptor de luz? O conceito de sensores de presença ligados às luzes da casa não é novo e nem estranho para a maioria de nós como visto em "Sensores de Movimento e Presença" (MAZZAROPPI, 2007, p.5-24), mas devido a algumas restrições e limitações dos sistemas existentes a criação de um módulo para esse fim parece razoável. O foco é usar além dos sensores de presença programação horária e regras lógicas para proporcionar não só comodidade como também economia de energia para a casa.

Módulo de Alarme: Desde os tempos mais remotos os seres humanos revezavam-se nos cuidados contra agentes que causavam danos à comunidade,

estabelecendo-se estrategicamente de forma a proteger-se de ataques de animais e de situações de risco. Até os chimpanzés possuem mecanismos de defesa e vigilância para alarmar suas comunidades em caso do perigo iminente. Portanto, os sinais de alarme no sentido de proteção ou defesa de grupos estão intimamente ligados à proteção das espécies¹¹.

Com a vida moderna cada vez passamos menos tempo em nossas casas o que pode representar um risco tanto de assaltos como também de acidentes como um incêndio por exemplo. Pensando nisso vamos utilizar vários sensores para monitorar portas e janelas contra invasão e a temperatura para prevenir incêndios e outros acidentes.

Após definidos cabe dizer que todos os Módulos serão controlados com um sistema central que será instalado no Raspberry PI que também irá registrar as informações em um banco de dados para assim fornecer dados a serem analisados. Os dados armazenados serão utilizados em caso de possíveis melhorias no sistema ou para formar prova da eficiência do mesmo.

Agora que conhecemos o escopo do sistema vamos apresentar de forma detalhada cada um dos módulos individuais com seus componentes e com sua descrição.

4.1. Módulo Anti Chuva





O módulo Anti Chuva foi pensado como uma prevenção para a casa, muitas vezes saímos de casa e esquecemos a janela aberta, e se chove pode-se molhar tudo dentro. Vimos esta situação como uma necessidade que poderia ser tratada em nosso trabalho.

O esquema de funcionamento do módulo é dividido em três partes: sensor de umidade que vai estar ligado em um Arduino dedicado somente para sensores, a central de processamento (Raspberry PI) e por último outro Arduino que será responsável pelos atuadores, onde estará conectado o servo motor que será responsável por fechar a janela. Abordaremos o desenvolvimento do módulo na parte relacionada ao Arduino, deixando claro seu funcionamento em questões físicas

e lógicas.

4.1.1. Partes necessárias

Tabela 1 - Lista de componentes do módulo de chuva.

Qtd.	Componente	Descrição	Foto
1	Sensor de Chuva	Trila sensível a umidade de alta sensibilidade	
1	Controlador de Função	PCB LM393 controlador de função, 3,2 x 1,4 cm.	
8	Cabos Jumpers	Cabos de cobre (jumpers) para ligação dos componentes	
1	Servo Motor	Servo Motor Hextronik modelo HXT900	

Fonte: Autoria própria.

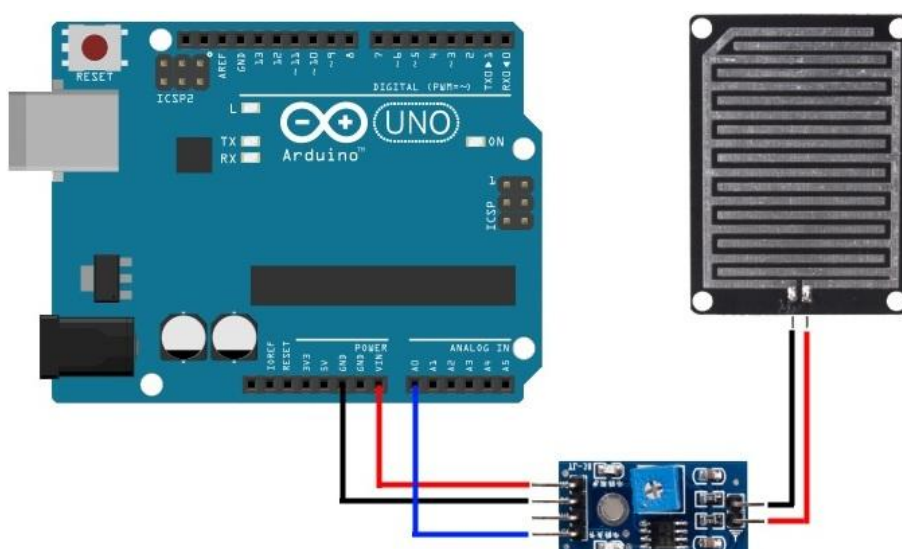
Nota: Todos componentes usados neste módulo são fáceis de achar em sites e lojas especializadas em eletrônica.

4.1.2. Processo de Montagem

Com todos componentes a mão vamos montar os dois circuitos que compõem o módulo, começando com a montagem do sensor de chuva. Para montar o esquema elétrico, é indicado que o Arduino esteja desligado tanto da porta USB como da energia elétrica. A seguir temos de separar dois cabos jumpers sendo um preto e um vermelho, os quais serão utilizados para conectar o sensor de chuva ao seu controlador LM393. Temos de conectar o cabo preto no conector esquerdo do sensor (fase neutro do circuito), a outra ponta do mesmo cabo encaixamos no conector esquerdo do controlador LM393, encaixamos uma ponta do cabo vermelho no encaixe direto do sensor de chuva, e a outra ponta do lado direito do controlador LM393. Agora ligaremos o Arduino a controladora, utilizando 3 cabos, um preto, um

vermelho e um azul. Começaremos ligando a parte positiva do circuito, encaixando uma das pontas do cabo vermelho na porta VIN do Arduino, e a outra ponta no encaixe VCC do controlador. Depois disso pegamos uma das pontas do cabo preto e encaixamos na porta GND (abaixo da porta de 5v), na sequencia encaixamos a outra ponta do cabo no encaixe GND do controlador. Até agora temos a fase positiva e negativa do circuito pronta, faltando somente ligar a parte lógica, para isso usamos o cabo azul, encaixando uma das pontas na porta analógica A0 do Arduino, e a outra ponta no encaixe A0 do controlador.

Figura 5 - Esquema de montagem sensor de chuva



Na segunda parte do circuito, vamos montar o esquema elétrico do Servo Motor em outro Arduino, e para isso o mesmo tem que estar desligado da porta USB ou da energia elétrica. Os primeiros passos para ligar o Servo Motor no Arduino são: separar três cabos jumpers, um preto, um vermelho e um amarelo, em seguida encaixar uma das pontas do cabo jumper vermelho na porta de 5 volts no Arduino, pegando a outra ponta e encaixando no conector referente ao fio vermelho do Servo Motor, assim ligando a fase positiva do circuito. Na sequência encaixar uma das pontas do jumper preto na porta GND abaixo da porta da de 5 volts do Arduino, que é a fase neutra do nosso circuito, e a outra ponta encaixar no conector referente à cor marrom do Servo Motor. Posteriormente encaixar uma das pontas do cabo jumper amarelo na porta analógica A1 do Arduino, e a outra ponta encaixar no conector referente à cor amarela do Servo Motor, que liga parte lógica do circuito. Deixando o Arduino pronto para receber o código.

Figura 6 - Esquema de montagem servo motor



Fonte: Autoria própria.

4.1.3. Funcionamento do Módulo

A seguir é mostrado o funcionamento do módulo, desde a lógica empregada no código (o qual está disponível no Apêndice 1) passando pela detecção da chuva, o recolhimento desta informação pelo Raspberry PI e finalmente a ativação do Servo Motor pelo programa, esse é o conjunto que forma o módulo Anti Chuva.

O sensor de chuva tem uma única função, a de detectar a presença de chuva e enviar um sinal elétrico informando a presença da mesma. Quando o programa do sensor de chuva é iniciado no Arduino, o método **chuva = analogRead (CHUVA);** é chamado iniciando o sensor de chuva em modo passivo, se for detectada a presença de chuva, é impresso em tela **CHV0:1000** (nome do sensor SCH e a sensibilidade da chuva). Quando esta informação é impressa, o Raspberry PI, que através de uma rotina lê a informação e ativa o Servo Motor no segundo Arduino indicando qual será o grau de rotação do braço do Servo Motor (para fechar realizar o fechamento da janela) esta indicação é possível usando a função **Serial.beign(9600);**, permite criar uma conexão onde se passa parâmetros para o funcionamento do Servo Motor. A variável responsável por receber os parâmetros

é a variável `c` do tipo `char` que armazena temporariamente o que é passado, se o que for passado corresponder aos parâmetros da função **OnOffSRM0** (função responsável pelo funcionamento do Servo Motor), o Servo Motor é ativado. Para ativarmos o Servo Motor usamos a combinação de comando e parâmetro, **SRM0** é o nome do servo e o parâmetro que indica o grau de rotação do braço é um número de 0 a 180 (graus), a junção do comando com o parâmetro fica como, por exemplo, **SRM0180**;

4.2. Módulo de Alarme

Este módulo foi pensado para ser uma preventiva a invasão residencial, onde sua função é simples, quando o alarme for ativado e for detectada presença, o mesmo emitirá um som repetidamente para alertar da invasão e será desativado somente por intervenção do usuário. Temos também um alarme em caso de incêndio ele é acionado por um sensor de temperatura que ao detectar uma certa temperatura, aciona o mecanismo emitindo um som repetidamente para alertar do possível incêndio, em cada alarme é criado um intervalo de repetição diferente para que o usuário saiba diferenciar.

O esquema de funcionamento do módulo é dividido em três partes:




1. Sensores: PIR (Passive Infra Red ou sensor de presença), sensor de temperatura;
2. Uma central de processamento (Raspberry PI), onde será feita toda a parte de programação (decisão dos módulos),
3. Atuador: Buzzer (Emissor de Som)¹² que irá simular uma sirene.

Abordaremos mais sobre o desenvolvimento do módulo na parte relacionada ao Arduino, deixando claro seu funcionamento em questões físicas e lógicas.

4.2.1. Partes Necessárias

¹² O Buzzer é um dispositivo eletrônico pouco maior que um comprimido composto de 2 camadas de metal e uma camada interna de cristal piezoelétrico (como um sanduíche), que ao ser alimentado com uma fonte de sinal, vibra na mesma frequência que a recebida, funcionando como uma espécie de sirene ou alto-falante.

Tabela 2 - Lista de componentes do módulo de alarme.

Qtd.	Componente	Descrição	Foto
1	Sensor "PIR"	PIR modelo Dyp-me003	
1	Buzzer	12mm Com Oscilador Interno 12V	
1	Sensor Termostato	Sensor modelo LM35de -55° até +150°C	

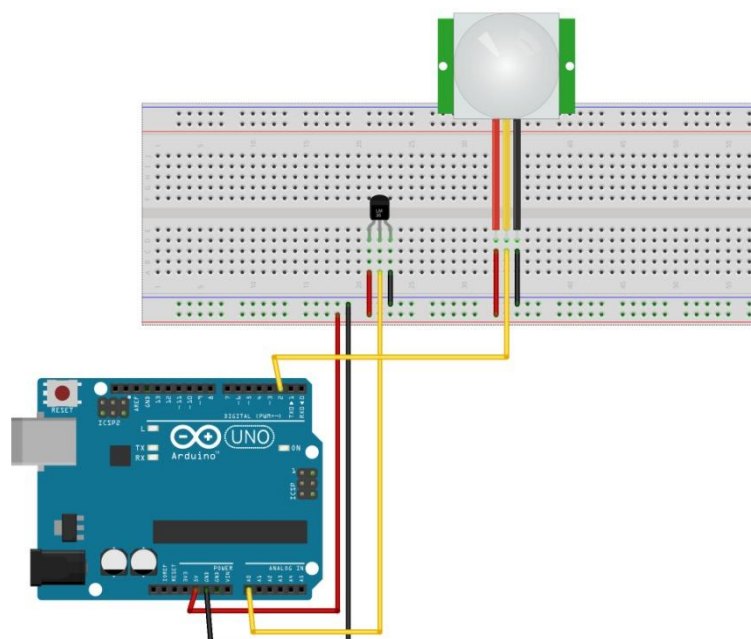
Fonte: Autoria própria.

Nota: Todos componentes usados neste módulo são fáceis de achar em sites e lojas especializadas em eletrônica.

4.2.2. Processo de Montagem

O primeiro esquema elétrico a ser montado, será o dos sensores, começando pelo sensor de movimento. Primeiramente o Arduino tem que estar desligado da energia elétrica ou desconectado da porta USB. Feito isso encaixamos a ponta do fio vermelho que sai do Sensor de Movimento na Protoboard, fazendo o mesmo processo com os fios amarelo e preto deixando um do lado do outro. Após encaixar todos os fios do sensor de movimento na Protoboard, vamos ligar o Arduino ao Sensor de Movimento. Separando cinco cabos Jumpers para fazer a ligação, dois vermelhos, um amarelo e dois pretos.

Figura 7 - Esquema de montagem dos sensores



Fonte: Autoria própria.

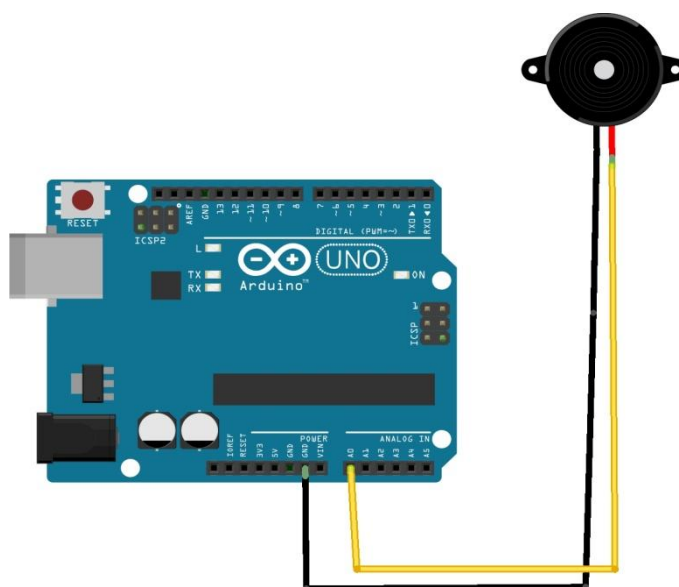
Pegando uma das pontas de um dos cabos Jumpers vermelhos encaixando na porta de 5 volts do Arduino, a outra ponta deve ser encaixada na parte lateral da Protoboard com indicação vermelha, sinalizando a fase positiva do circuito. Pegando o outro Jumper vermelho, encaixamos uma das pontas do mesmo na Protoboard, na mesma linha da fase positiva do circuito, e a outra ponta encaixamos na Protoboard abaixo do fio vermelho do Sensor de Movimento. Para ligar a fase negativa do circuito, separamos dois Jumpers pretos, encaixando uma das pontas de um dos cabos Jumpers na porta GND do Arduino, abaixo da porta de 5v, e a outra ponta encaixando na lateral da Protoboard com indicação azul. Sequentemente encaixamos uma das pontas do outro Jumper preto na mesma linha da fase negativa, ligada anteriormente, e a outra ponta do cabo jumper deve ser encaixada na Protoboard abaixo do fio preto do Sensor de Movimento. Na sequência vamos encaixar uma das pontas do Jumper amarelo na porta digital 2 do Arduino, e a outra ponta encaixando na Protoboard abaixo do fio amarelo do Sensor de Movimento, ligando a parte lógica do circuito para o Sensor de movimento e finalizando o mesmo.

Para ligar o Sensor de Temperatura, encaixamos as três pernas do mesmo na Protoboard, sendo a primeira a fase positiva, a segunda a fase lógica e a terceira a fase negativa do componente. Após encaixar o Sensor de Temperatura na

Protoboard, separamos 3 Jumpers, sendo um vermelho, um preto e um amarelo. Encaixamos uma das pontas do Jumper vermelho na Protoboard na mesma linha da fase positiva do circuito, e a outra ponta encaixamos na Protoboard abaixo da primeira perna do Sensor de Temperatura. Na sequência pegamos uma das pontas do Jumper amarelo e colocamos na porta analógica A0 do Arduino, e a outra ponta encaixamos abaixo da segunda perna do Sensor de Temperatura. Continuando pegamos uma das pontas do Jumper preto e encaixamos na linha da fase negativa, e a outra ponta encaixando na Protoboard abaixo da terceira perna do Sensor de temperatura. Tendo ligado o Sensor de Temperatura e o Sensor de Movimento, deixando os prontos para receber o programa.

Montando agora o segundo esquema elétrico no outro Arduino, para o Buzzer, as condições para manusear o Arduino são as mesmas do primeiro. O primeiro passo para conectar o Buzzer no Arduino, encaixar a ponta do fio preto do Buzzer na segunda porta GND abaixo da porta de 5 volts do Arduino, fechando a fase neutra do circuito. Em seguida encaixamos a ponta do fio vermelho na porta analógica A0 do Arduino, fechando o circuito com a fase positiva. E deixando o Arduino pronto para receber o código.

Figura 8 - Esquema de montagem Buzzer



Fonte: Autoria própria.

4.2.3. Funcionamento do Módulo

Quando o programa dos sensores é iniciado no Arduino, a função **Serial.begin(9600);** é carregado criando uma conexão serial, que permite ao Raspberry PI fazer leitura das saídas dos métodos dos sensores, que são carregados na sequência, os métodos **val = digitalRead(inputPin);** e **sensorValue = analogRead(sensorPin);** são chamados, deixando os sensores em modo passivo, o Sensor de Movimento é capaz de detectar movimentos e presença corporal com até 34°C, em um raio de 5 metros, se há movimentação ou presença, o sensor detecta e imprime em tela as informações necessárias para o Raspberry PI ler, que no caso **Serial.println("PIR0:MOTION");** o Raspberry PI conectado ao Arduino por uma porta USB, lê a informação via conexão serial para ver se houve alteração da informação, se houver, e o alarme estiver ativo o Raspberry PI ativa o Buzzer via conexão serial pelo método **Serial.begin(9600);** carregado no programa do Buzzer. Com este comando é a chamada a função **OnOffABU0** e o Buzzer é ativado. Para ativar o Buzzer é usado uma combinação de comando e parâmetros, o comando **ABU0** chama a função **OnOffABU0**, e logo em seguida passamos o comando **ABU0150002000**, onde 15000 indica os hertz do som, e o 2000 indicando o tempo em milissegundos para a pausa do som. A indicação de hertz para o Buzzer pode ir de 0 a 15000 e a indicação de milissegundos pode ir de 0 a 10000. Se o sensor de movimento informar **NOMOTION**, significa que não há mais presença, porém o alarme só pode ser desativado pelo usuário através da interface de controle provida pelo Raspberry PI, a qual envia o comando **ABU00000** desligando o Buzzer. O Sensor de Temperatura funciona em paralelo com o Sensor de Movimento, como citado antes quando o programa do Arduino é iniciado é carregado à função **Serial.begin(9600);** que permite o Raspberry PI fazer a leitura da saída do método **sensorValue = analogRead(sensorPin);** através de uma conexão serial criada pela mesma. O Sensor de Temperatura LM35 tem a capacidade de leitura de temperaturas de -55 °C a +150 °C, tendo uma precisão de 0,5 °C. Quando o sensor faz a leitura de 75 °C imprimir em tela **Serial.print("STP0:75");** o Raspberry PI ao fazer esta leitura irá processar a informação e ativar o Buzzer, passando um comando e um parâmetro diferente do usando para o alarme de invasão. Como falado antes está ativação é possível,

porque o Raspberry PI está ligado aos Arduino via portas USB, e que se comunicam através de conexão serial, o comando passado pelo Raspberry PI para ativar o Buzzer seria **ABU09001000**. Sendo 900 hertz para o som e 1000 indicando o tempo em milissegundos para a pausa do som, para assim ser facilmente distinguido do alarme de invasão.



4.3. Módulo de Termostato



O Módulo Termostato foi pensado para manter o ambiente agradável, pois, utiliza um sensor para captar a temperatura, e a partir disso tomar uma ação. Em nossa maquete iremos utilizar dois Coolers (Pequenos ventiladores) para manter o ambiente ventilado, simulando o acionamento de, por exemplo, um ar-condicionado central ou ainda um climatizador. Será localizado um na cozinha e outro no quarto. Previamente a temperatura tem que ser informada pelo usuário, e assim quando tal temperatura for detectada pelo módulo, os dois Coolers serão ligados. O módulo é controlável através de uma interface, aonde o usuário pode ligar quaisquer dos dois ventiladores de onde estiver da casa, através de um smartphone, Tablet ou PC.

O esquema de funcionamento do módulo é dividido em três partes, sensor de temperatura, Arduino1 (dedicado para sensores), central de processamento (Raspberry PI), que recolhe as informações dos sensores e tomam uma ação, e por último os dois Coolers, que ficam conectados no Arduino2 dedicado a atuadores, sendo ativados pela central.

4.3.1. Partes Necessárias

Tabela 3 - Lista de componentes do módulo Termostato.

Qtd.	Componente	Descrição	Foto
7	Cabos Jumpers	Cabos de cobre (jumpers) para ligação dos componentes	
2	Cooler	Cooler DC12V/0.1A	

1	Circuito integrado	ULN2003AN	
1	Sensor Termostato	Sensor modelo LM35C de -55° até +150°C	

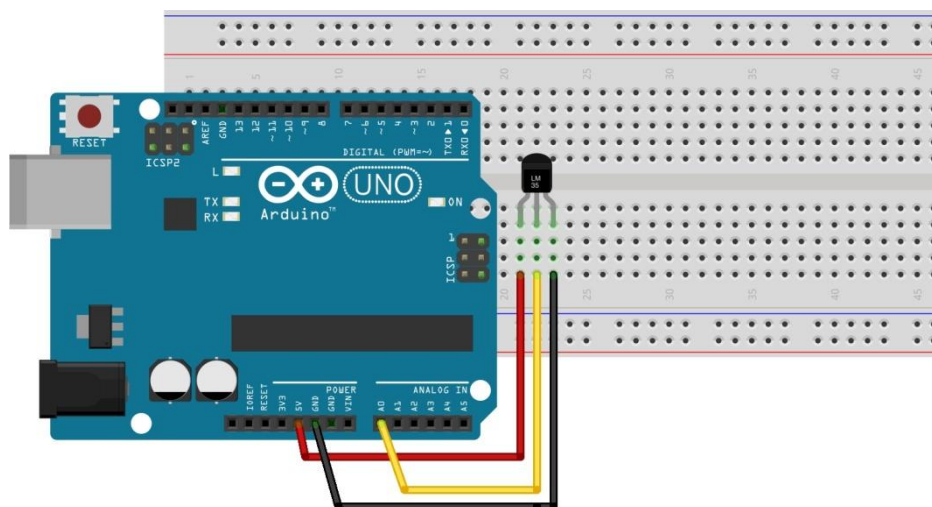
Fonte: Autoria própria.

Nota: Todos componentes usados neste módulo são fáceis de achar em sites e lojas especializadas em eletrônica.

4.3.2. Processo de Montagem

O primeiro esquema elétrico a ser montado, será o do Sensor de Temperatura. Para ligar o Sensor de Temperatura, encaixamos as três pernas do mesmo na Protoboard, sendo a primeira a fase positiva, a segunda a fase lógica e a terceira a fase negativa do componente. Após encaixar o Sensor de Temperatura na Protoboard, separamos 3 cabos, sendo um vermelho, um preto e um amarelo. Encaixamos uma das pontas do cabo vermelho na porta de 5 volts do Arduino, e a outra ponta encaixamos na Protoboard abaixo da primeira perna do Sensor de Temperatura, fechando a parte positiva do circuito. Na sequência pegamos uma das pontas do cabo amarelo e encaixamos na porta analógica A3 do Arduino, e a outra ponta na Protoboard abaixo da segunda perna do Sensor de Temperatura, fechando a parte lógica do circuito. Na sequência pegamos uma das pontas do cabo preto e encaixamos na porta GND do Arduino abaixo da porta de 5 volts, e a outra ponta encaixamos na Protoboard abaixo da terceira perna do Sensor de Temperatura, fechando a fase neutra do circuito, e finalizando o circuito do sensor por inteiro.

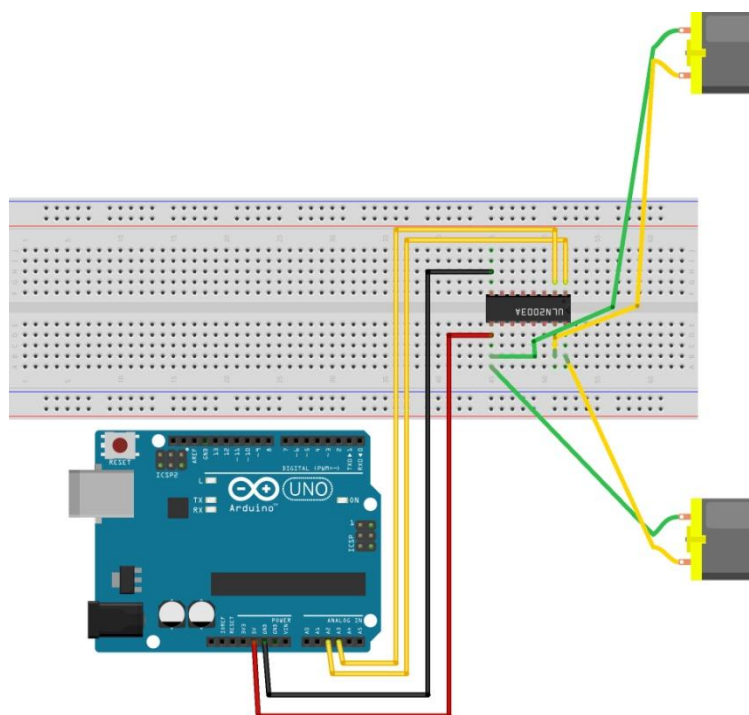
Figura 9 - Esquema de montagem sensor



Fonte: Autoria própria.

O segundo esquema elétrico a ser montado, será para os Coolers, seguindo os mesmos requisitos do anterior. Primeiramente encaixaremos o CI ULN2003AN na Protoboard, ele será responsável por passar a voltagem necessária para os Coolers, como o Arduino fornece apenas 5V para a alimentação dos atuadores, o CI pega estes 5V e transforma em 12V alimentando dois Coolers que trabalham em 12V, assim fornecendo a alimentação necessária para o funcionamento dos mesmos. Para identificarmos aonde se localiza a entrada positiva e negativa do CI, identificamos na superfície do mesmo um meio círculo cavado, que indica sua frente, então sua fase negativa fica na parte de traz, sendo a última perna esquerda. Do outro lado é a fase positiva do circuito. As demais pernas do lado esquerdo significam fase lógica, e finalmente as do lado direito são as correspondentes da mesma. Agora que já identificamos o CI, vamos ligá-lo ao Arduino. Separamos 4 cabos, sendo um preto, um vermelho e dois amarelos. Pegando uma das pontas do cabo vermelho, encaixamos a mesma na porta de 5 volts no Arduino, e a outra ponta encaixamos na Protoboard do lado da perna positiva do CI, fechando a fase positiva para o CI. Na sequência pegamos uma das pontas do cabo preto e encaixamos na porta GND do Arduino, e a outra ponta ao lado da perna negativa do CI, fechando a fase negativa. Separamos uma das pontas de um dos cabos amarelos e encaixamos na porta analógica A2 do Arduino, e a outra ponta na Protoboard ao lado CI, do lado da primeira perna do lado esquerdo, abaixo do meio círculo cavado, fechando a primeira fase lógica para o CI.

Figura 10 - Esquema de montagem coolers



Fonte: Autoria própria.

Finalmente pegamos uma das pontas do outro cabo amarelo e encaixamos na porta analógica A3 do Arduino, e a outra ponta encaixamos na Protoboard ao lado do CI, do lado da segunda perna do lado esquerdo, finalizando a fase lógica para o CI. Ligamos agora os Coolers no CI, encaixando a ponta do fio vermelho do primeiro Cooler na Protoboard, do lado da fase positiva do CI. Ligando agora a fase negativa para o Cooler, encaixamos a ponta do fio amarelo na Protoboard à frente da primeira fase lógica do CI, ficando do lado direito, finalizando o circuito para o primeiro Cooler. Na ligação do segundo Cooler o processo é o mesmo, a fase positiva fica do lado da fase positiva do primeiro Cooler. A fase negativa fica à frente da segunda fase lógica do CI, ficando do lado direito. Assim o CI faz o controle de voltagem para os Coolers, sendo alimentando pelo Arduino e passando a corrente aumentada para os Coolers.

4.3.3. Funcionamento do Módulo

Quando o primeiro Arduino é iniciado o programa do Sensor de Temperatura carrega duas funções, a primeira é a **Serial.begin(9600);** que permite uma conexão serial entre Arduino e Raspberry PI, e a segunda função é: **termostato2 = analogRead (TERMOSTAT02);** que deixa o sensor de temperatura em modo de


detecção, fazendo leitura da temperatura, e imprimindo em tela, para que o Raspberry PI possa lê-las. Como o Arduino e o Raspberry PI estão ligados por uma porta USB, quando a rotina de leitura do Raspberry PI é ativada, é passado um comando para o Arduino solicitando status do sensor através da conexão serial. Se for o mesmo **STATUS**, este comando invoca a função Status que retorna o valor da temperatura, se a temperatura for igual a que está memorizada para acionar os Coolers, o mesmo ativa a conexão serial com o segundo Arduino, através da função **Serial.begin(9600);**. Através desta conexão o Raspberry PI envia os seguintes comandos, **FAN0ON** e **FAN1ON**, que aciona a função **OnOffFAN0** e **OnOffFAN1**, ativando assim os Coolers, os comandos para todos os Coolers são dividido em duas partes, sendo **FAN0** e **FAN1** o comando, e **FAN0ON** e **FAN1ON** para ligar os Coolers, e se caso for necessário desliga-los é passado **FAN0OFF** e **FAN1OFF**. Sendo este todo o processo de funcionamento do Código.

4.4. Módulo de Iluminação

O Módulo de Iluminação foi pensado, visando tornar o ambiente agradável e econômico para quem reside na casa, utilizando a iluminação via LED para isso. Serão utilizados LEDs RGB aonde é possível escolher cores e intensidade da luz em qualquer cômodo da casa, podendo ser controlado por um smartphone, tablet ou computador conectado à rede. O módulo disponibiliza uma atuação inteligente no ambiente, com o uso de sensor, assim que o mesmo percebe a falta de luminosidade as luzes da casa começam a acender. Deixando um ambiente cômodo sem necessidades de intervenções. O esquema de funcionamento do módulo é dividido em três partes: sensor de luminosidade (Sensor LDR), que está em um Arduino dedicado somente para sensores, central de processamento que é um Raspberry PI, que faz toda a parte de decisão dos módulos, e por último o Arduino responsável pelos atuadores, onde estão os LEDs. Abordaremos o desenvolvimento do módulo na parte relacionada ao Arduino, deixando claro seu funcionamento em questão física e lógica.

4.4.1. Partes necessárias

Tabela 4 - Lista de componentes do módulo de Luminosidade.

Qtd.	Componente	Descrição	Foto
1	Sensor LDR	LDR 5mm, Sensor De Luz Para Arduino	
19	Cabos Jumpers	Cabos de cobre (jumpers) para ligação dos componentes	
1	Resistor	10kohm	
4	LEDs RGB	3 mm ou 5 mm Difusos	

Fonte: Autoria própria.

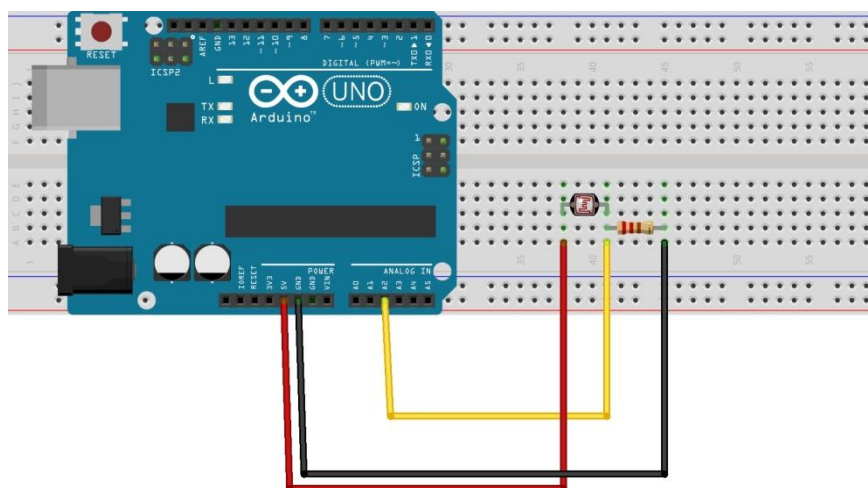
Nota: Todos componentes usados neste módulo são fáceis de achar em sites e lojas especializadas em eletrônica.

4.4.2. Processo de Montagem

O primeiro esquema elétrico a ser montado, será o do Sensor de Luminosidade LDR. O primeiro passo é encaixar o Sensor LDR na Protoboard, lembrando que a perna maior é a fase positiva do mesmo, após encaixar na Protoboard, encaixamos uma das pernas do resistor abaixo da fase negativa do Sensor LDR, e a outra perna encaixamos na horizontal do Sensor LDR. O uso do resistor é necessário para que não corra o risco de ser passada uma tensão maior do que a necessária, assim queimando o Sensor LDR. Agora que temos o sensor e o resistor encaixados na Protoboard, separamos três cabos para interligar o Sensor LDR ao Arduino, sendo um Jumper preto, um amarelo e um vermelho. Pegando uma das pontas do Jumper vermelho, encaixamos a mesma na porta 5V do Arduino, e a outra ponta na Protoboard abaixo da fase positiva do Sensor LDR. Na sequência pegamos uma das pontas do cabo preto e encaixamos na porta GND do Arduino,

abaixo da porta de 5 volts, e a outra ponta na Protoboard abaixo da perna do resistor, que não está ligado ao Sensor LDR, fechando assim a fase negativa do circuito. Por último pegamos uma das pontas do Jumper amarelo e encaixamos na porta analógica A2, e a outra ponta encaixamos na Protoboard abaixo da perna do resistor que está ligado ao Sensor LDR, finalizando assim a fase lógica do circuito, e o deixando pronto para receber o programa.

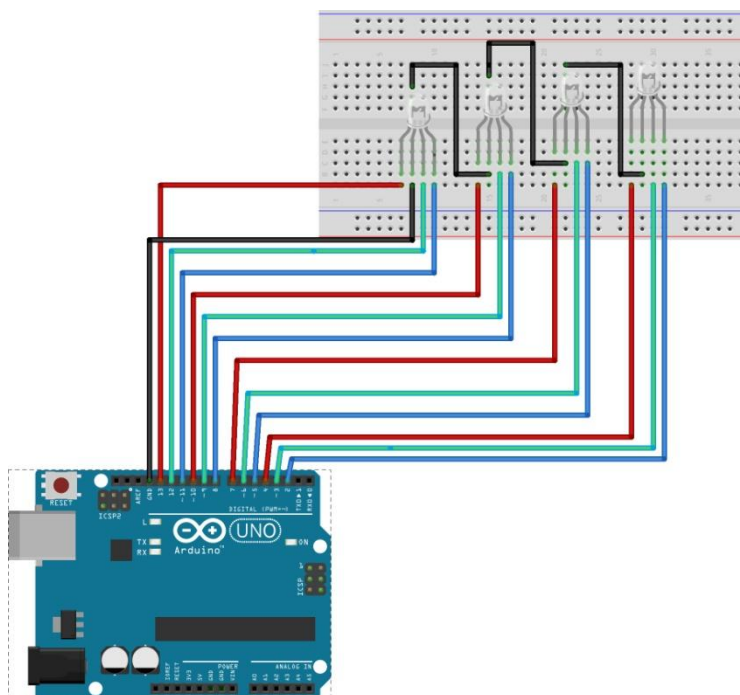
Figura 11 - Esquema de montagem do LDR



Fonte: Autoria própria.

O segundo esquema elétrico a ser montado é o dos LEDs RGB, que seguem a mesma preventiva do primeiro Arduino, o mesmo tem que estar desligado da energia elétrica e desconectado do USB. Primeiramente encaixaremos os LEDs RGB na Protoboard, separados com a diferença de um buraco da superfície da mesma, após isso vamos fazer a primeira ligação do circuito. Separamos 4 cabos pretos, pegando uma das pontas de um dos cabos e encaixando na porta GND do Arduino, que fica acima na porta digital 13, a outra ponta encaixamos na Protoboard abaixo da maior perna do primeiro LED RGB, sendo a fase negativa do LED RGB. Para ligar a fase negativa dos outros LEDs não será necessário mais portas GND, repassaremos o sinal do 1º para os demais. Encaixando uma das pontas de um dos jumpers preto, na Protoboard acima da maior perna do primeiro LED, e a outra ponta na Protoboard abaixo da maior perna do segundo LED RGB, repetindo até o ultimo LED. Como o LED RGB usa 4 fases, e a fase negativa de cada LED já foi ligada, precisamos ligar às fases positivas de cada LED, sendo a primeira responsável por ativar a cor vermelha, a segunda ativa a cor verde e a última ativa a cor azul.

Figura 12 - Esquema de montagem dos LED RGB



Fonte: Autoria própria.

Separamos doze Jumpers, sendo quatro vermelhos, quatro verdes e quatro azuis, informando quais cores de cada LED estaremos interligando ao Arduino. Pegamos uma das pontas de um dos Jumpers vermelho e encaixamos na porta digital 13 do Arduino, e a outra ponta encaixamos na Protoboard abaixo da primeira perna do primeiro LED RGB. Na sequência pegamos uma das pontas de um dos Jumpers verde e encaixamos na porta digital 12 do Arduino, e a outra ponta encaixamos na Protoboard abaixo da terceira perna do primeiro LED RGB. Seguindo o mesmo processo separamos uma das pontas de um dos Jumpers azul, e encaixamos na porta digital 11 do Arduino, e a outra ponta encaixamos na Protoboard abaixo da quarta perna do primeiro LED RGB que ativa a cor azul, finalizando assim a ligação das fases positivas do primeiro LED RGB. Para os demais LEDs RGB o processo é o mesmo, sendo que no final as portas digitais de 13 a 2 estarão preenchidas, ligando os quatro LEDs RGB ao Arduino. Deixando o circuito pronto para receber o programa

4.4.3. Funcionamento do Módulo

O Módulo de Luminosidade como falado antes em é dividido em três partes, a primeira parte é feita pelo Sensor LDR, responsável por detectar o grau de luminosidade do ambiente que fica em um Arduino dedicado para sensores. A segunda parte sendo a mesma a central de processamento é feita pelo Raspberry PI, que faz a leitura dos números indicados pelo sensor, que determinam o grau de luminosidade do ambiente, e a partir disso ativando a terceira parte. Sendo está representada pelos LEDs RGB que ficam em um Arduino dedicado a atuadores.

A comunicação entre o Raspberry PI e os Arduino, é possível fisicamente, pois os Arduino estão ligas e alimentados por portas USB no Raspberry PI, a comunicação lógica é feita através de conexão serial, carregadas nos dois Arduino. Ao iniciar o sistema, quando o Raspberry PI quer ativar um atuador, ele estabelece uma conexão e a partir dela é passa um comando para ligar um LED por exemplo. Quando uma leitura no Arduino dedicado a sensores o processo é o mesmo, mas só que ao estabelecer a conexão, é enviado um comando que retorna o resultado de detecção de cada sensor.

Quando o Arduino aonde se localiza do Sensor de Temperatura é iniciado é carregado à função **Serial.begin(9600);** que cria uma conexão serial, permitindo que o Raspberry PI faça a leitura do resultado da função **void Status()** ,enviando o comando **STATUS**. Após isso o Raspberry PI faz a leitura do resultado de detecção do Sensor LDR, pegando a informação processando e ativando um LED ou mais do segundo Arduino, que permite conexão serial através da função **Serial.begin(9600);** Para que o Raspberry PI possa ligar um LED, através da conexão serial passamos um comando é um parâmetro como por exemplo **LED0255000255**, este comando informa ao Raspberry PI para ativar o primeiro LED, identificado como LED0 que chama a função **OnOffLEDRGB0**. O Raspberry PI pode ativar mais de um LED se quiserem, os mesmos são identificados como LED0, LED1, LED2 e LED3 que representam a iluminação de cada cômodo da casa.

4.5. Central Raspberry PI

Para finalizar o nosso projeto vamos utilizar o Raspberry PI para a criação de um módulo de controle central. Este módulo será responsável pelo processamento

dos sinais digitais capturados pelos sensores conectados ao Arduino, e com base em uma análise preliminar (estabelecida pelo usuário), enviar um sinal para um determinado atuador conectado no outro Arduino. Para que isso seja possível utilizamos o Raspberry PI e scripts em Python responsáveis pela inteligência em capturar, processar e acionar todo o sistema.

4.5.1. Partes Necessárias

Tabela 5 - Lista de componentes do módulo central.

Qtd.	Componente	Descrição	Foto
1	Raspberry PI	Model B RAM, 512 MB, ARM 700 MHz	
1	Cabo USB	Cabo de Dados Micro USB to USB Marca Sony Ericsson - EC450	
1	Fonte de alimentação	Genérica com Porta USB - 5 Volts, 2000 mA	
1	Cartão SD	Cartão Micro SD com Adaptador SD – Marca Sony, class 10, 4gb	
1	Cabo HDMI	HDMI versão 3.0, iec60754-2	
1	Cabo de rede	Cabo RJ45 CAT 6	
1	Teclado e Mouse USB	Genéricos dentro dos padrões ABNT	

Fonte: Autoria própria.

No geral o todos os componentes podem ser encontrados em lojas do ramo e possuem baixo custo, com exceção ao Raspberry PI que não é tão fácil de ser encontrado no Brasil, mas pode ser importado via web.

4.5.2. Processo de Montagem (Configurando o Raspberry PI)

Conforme o descrito no Raspberry PI Wiki Para utilizar o dispositivo, é necessário instalar um sistema operacional em um cartão SD (Secure Digital Card ou SD Card). Somente após termos um sistema operacional em funcionamento é que podemos instalar algum software extra para começar a desenvolver códigos e conectar dispositivos ao Raspberry PI.

Na verdade é possível usar outros mecanismos de armazenamento, como uma unidade USB ou disco rígido externo também ligado a USB, mas o cartão SD é pequeno e deixa as portas USB liberadas, portanto, sendo a melhor escolha para dispositivos embarcados, como os encontrados em projetos de automação residencial.

Há uma grande variedade de marcas de cartões SD no mercado, e podem ser encontrados em diversos tamanhos. O Raspberry PI suporta praticamente todos os cartões SD, mas para a instalação de um sistema operacional completo é necessário um com no mínimo 2 GB de armazenamento.

Antes que possamos instalar o nosso sistema operacional, é preciso configurar o cartão SD. Isso envolve primeiro formatá-lo para o formato de sistema de arquivos FAT. FAT (File Allocation Table ou Tabela de Alocação de Arquivos) é um sistema onde uma Tabela (que na verdade é um mapa), mapeia a utilização do espaço do disco, graças a ela o sistema operacional é capaz de saber onde exatamente no disco um determinado arquivo está armazenado. segundo Gabriel Torres em: "Sistemas de Arquivo"¹³. Devido à sua robustez e simplicidade, ainda é encontrado em cartões SD e é o formato que será necessário para executar nosso sistema operacional. Podemos escolher entre uma grande variedade de sistemas operacionais que estão disponíveis para instalar no Raspberry PI. Nossa escolha foi o Raspbian. Raspbian é baseado no sistema operacional Debian Wheezy e foi otimizado para uso com o Raspberry PI.

Como descrito no site do projeto: "Raspbian é mais que um simples SO: ele vem com mais de 35 mil pacotes pré-compilados e empacotados para fácil instalação em seu Raspberry PI"¹⁴, ele foi escolhido por ser baseado no Debian pois herda as facilidades de instalação de pacotes através do apt-get, além é claro da interface amigável.

Começamos baixando os softwares necessários para fazer a instalação básica do Raspberry PI:

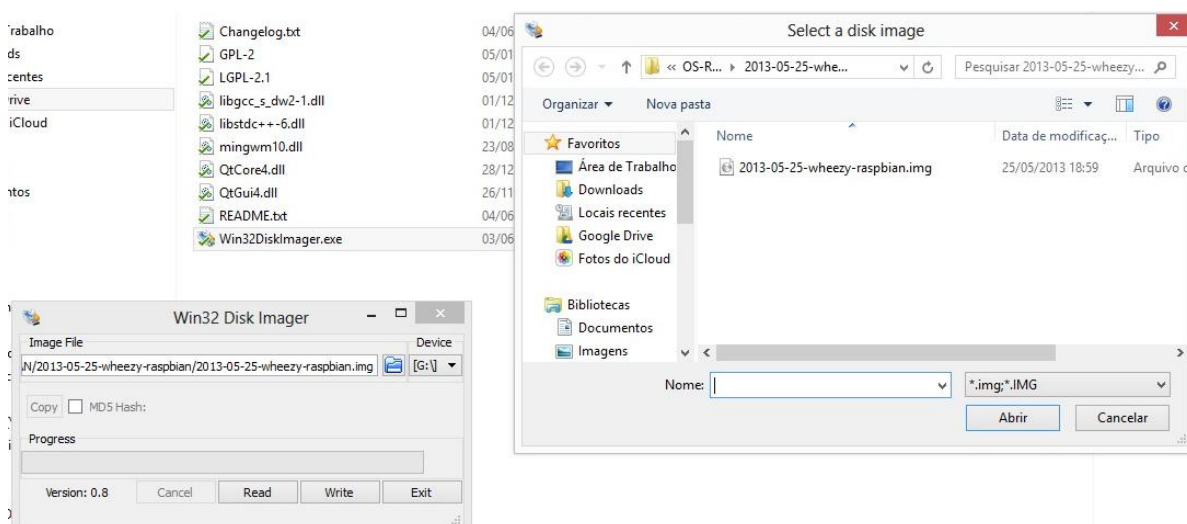
- OS Raspbian: <http://www.raspberrypi.org/downloads>
- Win32 Disk Imager: <http://sourceforge.net/projects/win32diskimager/>
- Python 2.7: <http://www.python.org/download/releases/2.7.6/>

Todos os softwares descritos são softwares livres, fáceis de usar e de serem instalados. A comunidade do Raspberry PI é bem ativa e tudo está bem documentado.

Após os downloads dos itens acima, vamos começar preparando o cartão SD para receber o Sistema Operacional, realizando os passos a seguir:

1. Descompacte o “Win32DiskImager”
2. Execute o programa com um duplo click em Win32DiskImager.exe.
3. Em “Image File” localize a imagem do Raspbian como na imagem

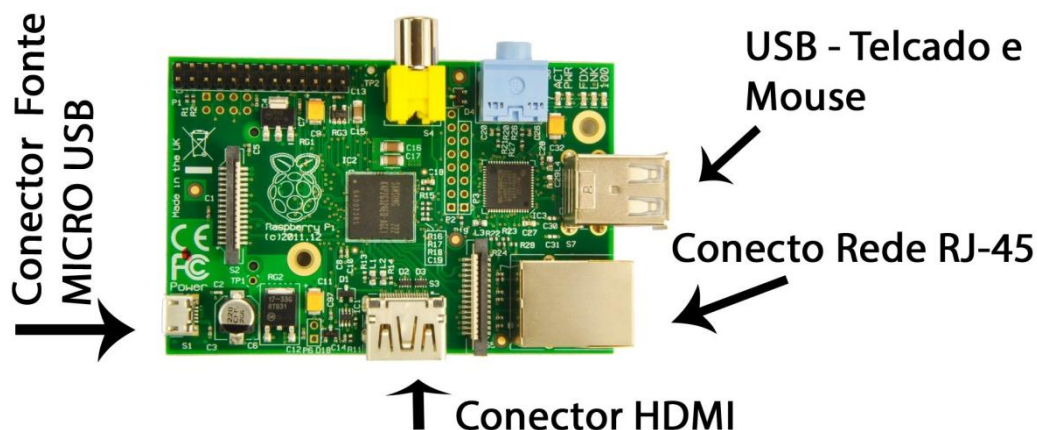
Figura 13 - Carregar imagem Raspbian OS



4. Em “Device” selecione a unidade onde está o montado Cartão SD
5. Verifique se os dados estão corretos e click em Write.

Com o cartão pronto vamos ao processo de montagem, que basicamente consiste em conectar os componentes conforme indicado na figura.

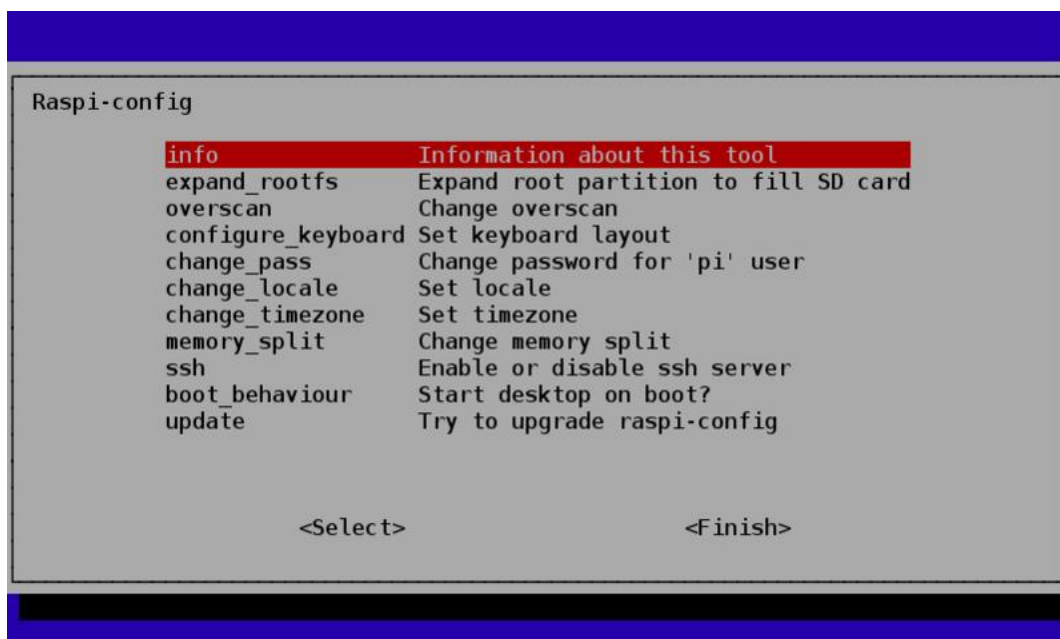
Figura 14 - Montagem Raspberry PI



Com tudo montado vamos ao primeiro boot, logo na primeira tela precisamos configurar o tamanho do Cartão SD e Iniciar o modo gráfico automaticamente.

1. Selecione a opção “expand_rootfs” para expandir a partição do cartão do cartão de memória, e pronto sua partição terá o tamanho total do cartão.
2. Para Iniciar no modo gráfico após boot, Selecione “Start desktop on boot”

Figura 15 - Rasp. - Config.



Após completarmos a instalação seremos levados ao desktop do Raspbian onde podemos ver alguns ícones entre eles o navegador web Midori, e o IDE Python¹⁵

¹⁵ IDE (integrated development environment em português ambiente de desenvolvimento integrado).

Figura 16 - Desktop do Raspbian



Pronto assim já temos o sistema instalado e quase pronto para receber o código do sistema de automação.

4.5.3. Programação do Módulo

As comunicações entre o Raspberry PI e Arduino podem ser feitas de várias formas como, por exemplo, GPIO, Wifi, USB ou via Ponte. Vamos conhecer um pouco sobre elas:

- **GPIO:** Este tipo de comunicação é nativo no Raspberry PI e funciona como uma porta serial.
- **WIFI:** Wifi no formato de uma conexão TCP/IP também pode ser usado, no entanto necessita de shield específico para funcionar.
- **USB:** Conexão nativa em ambas as plataformas e que transmite dados e energia.
- **Ponte de Ligação:** Nesse formato é necessário um Shield específico que além de conectar disponibiliza todo o ambiente do Arduino no Raspberry PI.

Inicialmente pensamos em ir pelo caminho mais fácil que seria utilizar o shield “connection bridge” o qual permite que o Raspberry PI se comunique com o Arduino de forma quase nativa. Porém como é necessário instalar a biblioteca arduPi que é

um conjunto personalizado de arquivos escritos em C pela equipe do Hacks Cooking¹⁶ iria fugir conceitualmente de nossa ideia inicial de só utilizar programas e peças livres, além de alguns pontos mais técnicos os quais iremos esclarecer na conclusão. No sistema apresentado utilizaremos a interface USB, pela vantagem da mesma transportar dados e energia suficientes para ativar as placas Arduino, e como essa conexão é nativa e mais prática que a via GPIO ela foi à escolhida.

Para iniciar a utilização da conexão serial temos que utilizar o pacote pySerial, que pode ser obtido na página oficial do projeto <http://pyserial.sourceforge.net/>. Após o download, basta acessar a pasta (via terminal) onde foi feito download e descompactar o pacote usando o comando TAR:

```
tar -xzf pyserial-2.7.tar.gz
```

E instalar o pacote pySerial:

```
cd pyserial-2.7
```

```
sudo python setup.py install
```

Após definido o modelo de comunicação física estruturamos a comunicação lógica, de maneira a possibilitar o acionamento de quaisquer módulos com as mais diversas características de maneira simples e eficiente. Para isto criamos comandos que foram constituídos com base na análise de Strings e na conversão delas em valores validos para cada modulo no Arduino.

As Strings de comandos devem ser criadas de acordo com as seguintes regras: Os quatro primeiros caracteres da string representam o modulo a ser acionado, a partir do quinto caracter será considerado como parâmetro, como mostra o exemplo:

```
[CMD][PAR]  
|  |-----> Parâmetros a serem enviados para o modulo.  
|      |---> 4 dígitos, para identificar o modulo a ser acionado.
```

Para acionar um LED devemos utilizar um comando maior conforme visto abaixo. Esse tipo de comando deve enviar um sinal digital para cada pino referente

ao valor RGB inteiro podendo estar entre 0 e 255. Nesse projeto utilizaremos o comando conforme segue para acionar somente a cor vermelha do LED:

```
[RGB0][255][000][000]
|      |-----> Identificador do LED a ser ativo.
|      |      |-----> Nº inteiro de 0 a 255 ref. a cor Vermelha.
|      |      |-----> Nº inteiro de 0 a 255 ref. a cor Verde.
|      |      |-----> Nº inteiro de 0 a 255 ref. a cor Azul.
```

A resposta a esse comando deverá ser a confirmação da cor acessa como mostra exemplo:

Resposta:

```
[CMD][:][STATUS]
|      |-----> Comando enviado
|      |      |-----> Separador
|      |      |-----> Status do modulo ou comando
```

No caso como acendemos a cor vermelha a resposta será “RGB0:255000000” sendo:

```
[RGB0][:][255000000]
|      |-----> Modulo Acionado.
|      |      |-----> Separador
|      |      |-----> Status das cores do LED
```

A partir deste modelo de comunicação o Arduino recebe estes valores como e um vetor de caracteres, que pode ser tratado facilmente com auxílios de recursos da própria linguagem como mostra o código de exemplo abaixo, ele descreve como aplicar o modelo de comando a um LED RGB.

```
//Biblioteca necessária dos C
#include <stdio.h>
#include <string.h>
#include <libio.h>
//Define todos os pinos a serem usados
#define LEDR 1 //Define a porta digital 1 para o Pino referente à cor VERMELHA do LED
#define LEDG 2 //Define a porta digital 2 para o Pino referente à cor VERDE do LED
#define LEDB 3 //Define a porta digital 3 para o Pino referente à cor AZUL do LED
//Declara todas as variáveis globais
//readString recebe um valor temporário para o input de dados.
String readString;
//Declara Previamente as funções.
```

```

// O método Execulte, processa o Vetor de Caracteres contendo os comandos e
parâmetros
// enviados pelo Raspberry PI.

void Execulte(String str);
// O método OnOffLED recebe os parâmetros para ativar ou desligar o LED RGB
void OnOffRGB(String str);
//Setup Inicia todos os pinos necessários no bootloader
void setup(){
    //Iniciar Serial com velocidade 9600 bps
    Serial.begin(9600);
    // Declara que os pinos definidos nas variáveis LEDR, LEDG, LEDB será iniciado no
modo
    // OUTPUT
    pinMode(LEDR, OUTPUT);
    pinMode(LEDG, OUTPUT);
    pinMode(LEDB, OUTPUT);
}
//Execulta em loop todas as funções.
void loop(){
    //
    // Enquanto a comunicação serial está disponível captura os dados imputados.
    while(Serial.available()){
        //Espera 3 milissegundos
        delay(3);
        //Captura char da entrada
        char c = Serial.read();
        //Adiciona o char lido no final da String readString.
        readString +=c;
    }

    //Se a String readString for conter mais de 0 caracteres
    if(readString.length()>0){
        //Executa o método "Execulte"
        Execulte(readString);
        //Limpa o conteúdo do readString
        readString="";
    }
}
//Método que trata a entrada de dados Capturados no método LOOP
void Execulte(String str){
    //Define que a string value deve possuir apenas 5 caracteres
    String value = str.substring(0,5);
    //Cria um vetor de char para armazenar os caracteres lido value
    char charBuf[str.length()];
    //Adiciona a String value dentro o vetor charBuff
    value.toCharArray(charBuf, 5);
    // Compara se a entrada de value é igual a RGB0 se for igual executa o método
OnOffRGB
    // passando como parâmetro toda a string recebida no inicio do método Execulte.
    if(strcmp(charBuf,"RGB0")==0) OnOffRGB0 (str);
}
// Liga o Desliga o LED RGB0
void OnOffRGB0 (String str){
    //Verifica o tamanho da variável str.
    if(str.length()){
        char charBufR[4]; //Cria a variável charBufR com tamanho para quatro
caracteres, para armazenar o cor da cor Vermelha.
        char charBufG[4]; //Cria a variável charBufG com tamanho para quatro

```

```

caracteres , para armazenar o cor da cor Verde.
    char charBufB[4]; //Cria a variável charBufB com tamanho para quatro
caracteres , para armazenar o cor da cor Azul.
    // Recebe o conteúdo dentro da variável str como por exemplo RGB0255000000,
    // Pega valores do quarto caractere até o sétimo
    str.substring(4,7).toCharArray(charBufR, 4);
    // charBufR = 255
    // Pega valores do sétimo ao décimo caractere.
    str.substring(7,10).toCharArray(charBufG, 4);
    // charBufG = 000
    // Pega valores do décimo até o décimo terceiro.
    str.substring(10,13).toCharArray(charBufB, 4);
    // Após as separação dos valores, vamos convertê-los em valores numéricos
    inteiros para isto usa-se a função "atoi"
    int iR = atoi(charBufR); //atoi converte os caracteres 255 em inteiro da
    variável charBufR, passando para a variável iR de tipo inteiro.
    int iG = atoi(charBufG); //atoi converte os caracteres 000 em inteiro da
    variável charBufG, passando para a variável iG de tipo inteiro.
    int iB = atoi(charBufB); //atoi converte os caracteres 000 em inteiro da
    variável charBufB, passando para a variável iB de tipo inteiro.
    Serial.print("RGB0:"); //Printa em tela RGB0:.
    analogWrite(LED0,iR); //Liga a Cor vermelha " LED0" do LED RGB0 com valor de
    0 a 255 que esta dentro da variável iR.
    Serial.print(iR); //Printa em tela o valor dentro da Variável iR, podendo ser
    de 0 a 255.
    Serial.print(" "); //Printa em tela um espaço em branco.
    analogWrite(LED0,iG); //Liga a Cor verde "LED0" do LED RGB0 com valor de 0 a
    255 que esta dentro da variável iG.
    Serial.print(iG); //Printa em tela o valor dentro da Variável iG, podendo ser
    de 0 a 255.
    Serial.print(" "); //Printa em tela um espaço em branco.
    analogWrite(LED0, iB); //Liga a Cor Azul "LED0" do LED RGB0 com valor de 0 a
    255 que esta dentro da variável iB.
    Serial.println(iB); //Printa em tela o valor dentro da Variável iB, podendo
    ser de 0 a 255.
}

```

Após a compilação e upload no Arduino testes podem ser feitos utilizando o "Serial monitor" do Arduino IDE. Após a validação dos comandos vamos constituir um script em Python a ser executado diretamente no Raspberry PI. Este código através da biblioteca pySerial permite a comunicação entre os Hardwares de maneira simples e eficiente.

Agora que o código do Arduino está pronto vamos ao Python, neste exemplo criaremos um pequeno programa que se conecta através da porta USB, e envia um comando previamente determinado.

```

# Arquivo CommandSerial.py
#
# Projeto Led RGB & Serial - RASPBERRY PI/ARDUINO/PYTHON

# importa a classe serial instalada anteriormente, e a classe time nativa do
python
import serial, time

```



```

# Cria a class CommandSerial
class CommandSerial:
    #Cria o método Send que envia o comando para o arduino
    def Send(self,cmd):
        #Inicial a comunicação Serial na porta /dev/ttyACM0
        ser = serial.Serial('/dev/ttyACM0', 9600)
        # Envia o comando para o arduino
        ser.write(cmd)
        resp = ser.readline()
        #fecha comunicação serial
        ser.close()
        return resp

#
# Arquivo console.py
#
# importa as biblioteca serial e sys
import serial,sys
# Importa class CommandSerial.py
from CommandSerial import *
# Inicia a variavel conn com classe CommandSerial()
conn = CommandSerial()
# Envia argumentos
resp = conn.Send(sys.argv[1])
# Imprime resposta.
print resp

```

A partir do modelo proposto é possível reproduzir um ambiente homogêneo que consegue interagir com diversos sensores e atuadores distintos em diferentes placas de Arduino, utilizando apenas o Raspberry PI e os recursos de comunicação serial do Python.

Com os dados apresentado até agora é possível criar um padrão para qualquer tipo de implementação no Arduino. Com tudo agora será apresentado um modelo para o funcionamento conjunto de todos os módulos dividido em quatro camadas sendo elas:

- **Banco de dados:** Nesta camada estão armazenados os dados de cara sensor e atuador, bem como informações de usuários e logs e tarefas a serem executadas.
- **Tarefas:** É a camada responsável por processar os sinais recebidos do Arduino bem como se necessários enviar um comando.
- **Filas:** Responsável por gerenciar os comandos a serem enviados para atuadores e sensores a fim de evitar problemas de comunicação simultânea em uma das placas do Arduino.
- **Interface:** Uma interface com usuário responsável por interseções manuais e alimentação de dados no banco de dados.

A seguir vamos descrever cada uma das camadas, para na sequência explicar o funcionamento do módulo.

4.5.4. Camada de Banco de Dados

Esta camada foi elaborada para gerenciar os dados necessários para o funcionamento da aplicação sendo eles:

- **Autenticação:** Responsável por guardar usuário e senha e nome de cada usuário.
- **Dados dos módulos:** Responsável por armazenar os dados dos módulos sendo eles: endereço da porta serial no Raspberry PI, comando, e estado do módulo.
- **Tarefas:** Armazena dados de quando os comandos do Arduino devem ser executados ou comando a serem enviados quando uma condição previamente estabelecida e satisfeita.
- **Fila de comandos:** Grava temporariamente os comandos enviados para cada Arduino e isso porque na comunicação serial não aceita comunicações simultâneas na mesma porta.
- **Logs de operação:** Armazena todas as operações feitas na interface e todos os comandos enviados para os Arduino.

Para um melhor entendimento vamos conhecer as estruturas das tabelas:

Tabela 6 - Tabela de Usuários

Nome Coluna	Tipo	Tamanho	Características	Descrição
Usuário	Numérico	6	Chave primaria, auto incremental.	Nome de usuário para logar no sistema
Senha	Alfanumérico	100	Não nulo	Senha do usuário para logar no sistema.
Nome	Alfanumérico	50	Não nulo	Armazena o nome usuário

Esta tabela grava os dados dos usuários, e é responsável pela autenticação do sistema. A aplicação é multiusuário e pode ser modificada por qualquer um dos usuários.

Tabela 7 - Tabela Módulos

Nome Coluna	Tipo	Tamanho	Características	Descrição
Id	Numérico	6	Chave primaria, auto incremental.	Identificação do modulo no sistema.
Nome	Alfanumérico	50	Não nulo	Nome do modulo
Comando	Alfanumérico	4	Não nulo	Identificador do comando a ser enviado para Arduino
Porta	Alfanumérico	255	Não nulo	Porta Serial apontada no Raspberry PI
Estado	Alfanumérico	50	Não nulo	Armazena o último comando enviado com sucesso

Está tabela grava dados dos módulos, necessária para o agendamento e processamento dos sinais recebidos pelos módulos.

Tabela 8 - Tabela de Tarefas.

Nome Coluna	Tipo	Tamanho	Características	Descrição
Id	Numérico	6	Chave primaria, auto incremental.	Identificação do modulo no sistema.
Idmodulo	Numérico	6	Não nulo, Relacionamento com id da tabela módulos	Identificação do modulo na tabela de módulos
Nome	Alfanumérico	50	Não nulo	Nome da Tarefa a ser enviada.

Responsável pelo armazenados dos dados de tarefas e quando devem ser executadas. Sendo possível criar um Schedule (Agendamento) para as tarefas do sistema.

Tabela 9 - Tabela condições.

Nome Coluna	Tipo	Tamanho	Características	Descrição
Idtarefa	Numérico	6	Não nulo, Relacionamento com id da tabela tarefas	Identificação da Condição
Valor	Alfanumérico	20	Não nulo	Compara com o valor capturado só sensor
Operador	Alfanumérico	1	Não nulo	Operador lógico para comparação de valores
Comando	Alfanumérico	100	Não nulo	Comando a ser enviado se a condição for satisfatória

Grava as condições necessárias para validar se uma tarefa é válida e se deve ser executada, basicamente é onde colocamos a inteligência para a realização de uma ação.

Tabela 10 - Tabela horários.

Nome Coluna	Tipo	Tamanho	Características	Descrição
Id	Numérico	6	Chave primaria, auto incremental.	Identificação do modulo no sistema.
Data inicial	Data hora	20	Nulo	Data inicial para processo, se nulos são ignorados.
Data final	Data hora	20	Nulo	Data final para processo, se nulos são ignorados.
Repetir	Numérico	6	Nulo	Campo inteiro que recebe valor em segundos para repetir a tarefa.
Espera	Data hora	20	Nulo	Controla o tempo de espera caso existam

Armazena dados de quando a tarefa for executada sendo possível ser recursiva a cada x segundos, ou em períodos específico.

Tabela 11 - Tabela de filas.

Nome Coluna	Tipo	Tamanho	Características	Descrição
Id	Numérico	6	Chave primaria, auto incremental.	Identificação do modulo no sistema.
Usuário	Alfanumérico	20	Não nulo	Usuário que gerou comando
Idtarefa	Numérico	6	Nulo	Id da tarefa exultada
Data	Data Hora	20	Não nulo	Data hora do envio de comando
Tipo	Alfanumérico	20	Não nulo	Definira se o comando é de entrada ou saída
Porta	Alfanumérico	255	Não nulo	Porta onde deve ser executado

Nesta tabela são armazenados os comandos a serem enviados para o Arduino, como temos que criar uma fila os comandos são executados na ordem, o primeiro que chega é o primeiro a ser executado.

Tabela 12 - Tabela de logs.

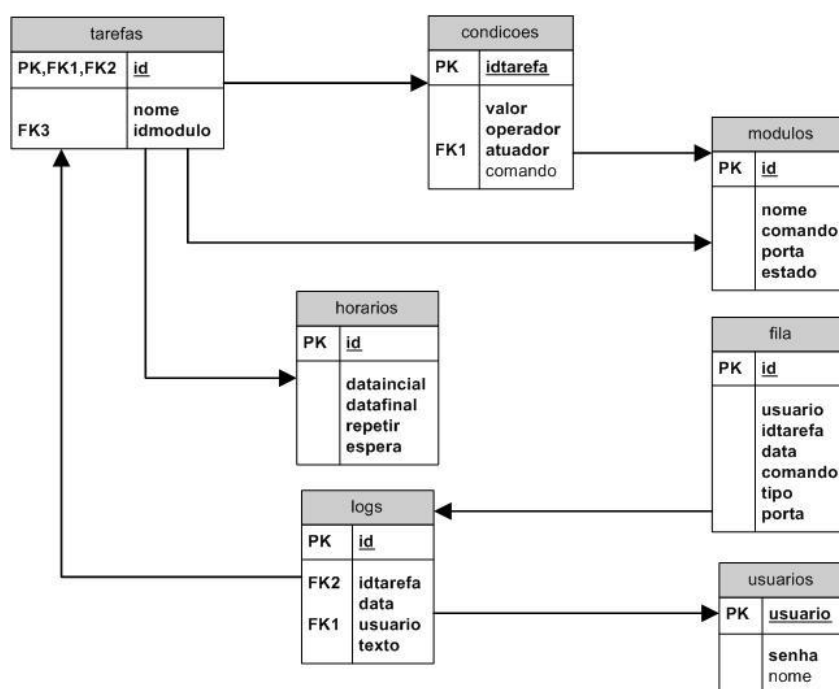
Nome Coluna	Tipo	Tamanho	Características	Descrição
Id	Numérico	6	Chave primaria, auto incremental.	Identificação do modulo no sistema.
Data	Data Hora	20	Não nulo	Data hora da tarefa executada
Idtarefa	Numérico	6	Não nulo, Relacionamento com id da tabela tarefas	
Usuário	Alfanumérico	20	Não nulo, Relacionamento com usuário da tabela usuários	

Texto	Alfanumérico	255	Não nulo	Descrição completa do que foi executado.
-------	--------------	-----	----------	--

Armazenando todas as informações geradas no sistema, e permitindo verificar se uma determinada tarefa pré-programada foi executada e colhendo informações para a geração de relatórios e/ou gráficos.

A seguir vamos conhecer o diagrama do banco:

Figura 17 - Diagrama Entidade/Relacionamento



O banco vai ser implementado utilizando o SGBD MySQL em sua versão (5.5.31), nesse momento iremos utilizar a criação dos usuários e demais dados do sistema diretamente no banco de dados, sendo que a criação de uma interface para esse fim será encarada como uma melhoria para uma futura versão do sistema.

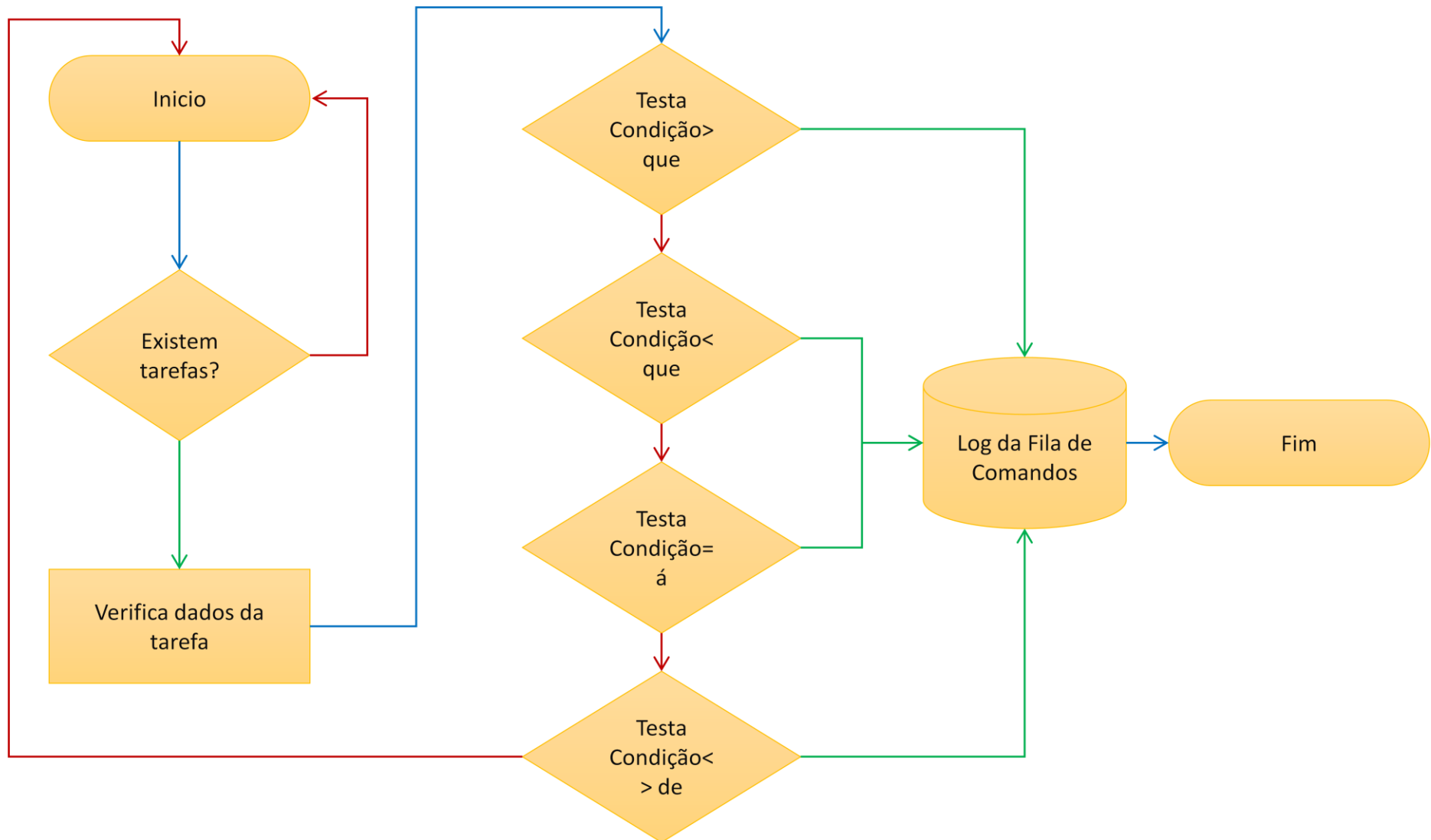
4.5.5. Camada de Tarefas (Aplicação)

Esta camada consiste de um serviço que processa os dados armazenados na tabela Modulo, Tarefas, Condições e horários. Isto ocorre da seguinte maneira:

1. É verificado se existem tarefas a serem executadas no momento atual.
2. Se houver tarefas captura o valor do modulo indicado
3. Testa as condições necessárias
4. Se forem validas grava o comando na fila, se não retorna ao ponto 1.

Conforme demonstrado na imagem abaixo:

Figura 18 - Camada de Tarefas.



4.5.6. Camada de Filas

O Gerenciamento de filas é extremamente importante para que duas comunicações simultâneas não sejam executadas na mesma placa Arduino. Esse evento ocasionaria um erro de comunicação, isso por que o uso de portas seriais não diferentes em comunicações TCP/IP não permitem que mais de uma conexão seja estabelecida na mesma porta com a mesma velocidade conforme relatado em nossa conclusão.

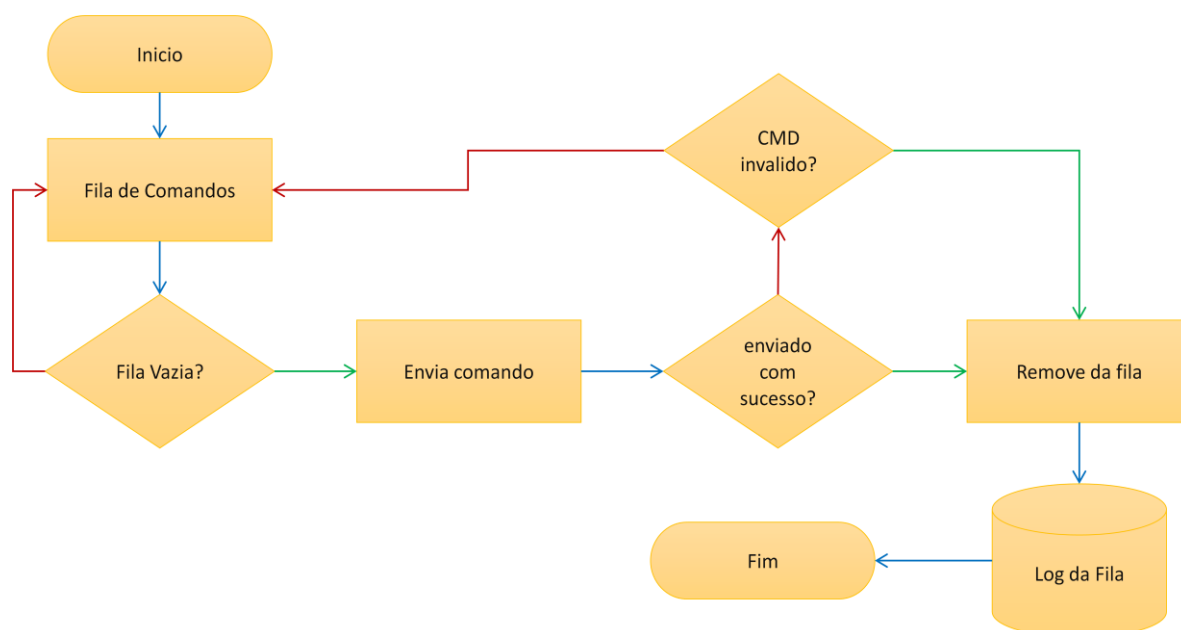
Para evitar esse tipo de evento criamos uma fila de comandos onde, todos os comandos da fila são enviados por um único serviço que tem apenas a função de transmitir informações para o Arduino.

Os passos necessários para isto são:

1. Verificar se há comandos na fila a serem enviados.
2. Enviar comando para o Arduino de destino.
3. Verificar se o comando foi enviado com sucesso.
4. Se sim remover da fila e gravar no log, se não reiniciar o processo.

Conforme demonstrado na imagem abaixo:

Figura 19 - Camada de Filas.



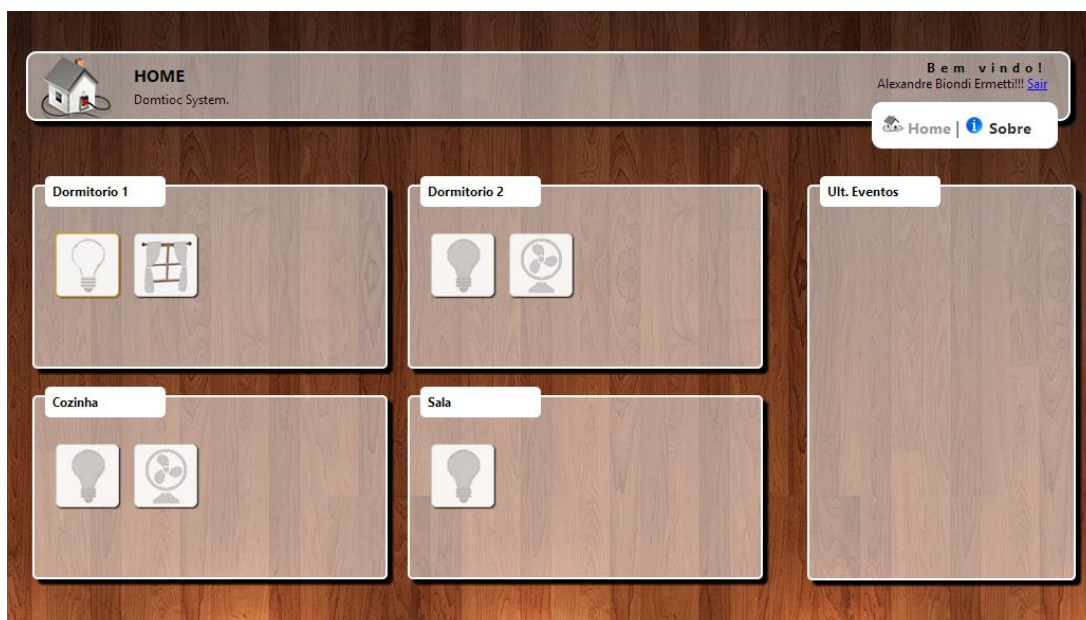
Com essa inteligência evitamos erros e controlamos a execução de tarefas de forma ordenada onde o primeiro comando a entrar é o primeiro a ser executado e assim por diante.

4.5.7. Camada de Interface

A interface ainda é considerada um ponto importante, pois não importa quantos módulos ou processos estejam funcionando simultaneamente e o quanto estes sejam otimizados, em algum momento eles podem precisar de uma intervenção humana.

Ela foi elaborada para que o usuário possa interagir com o sistema através de um web browser seja através de um PC, smartphone ou tablet. Elaborado utilizando HTML, CSS e Python 2 .6.7, é a responsável por capturar as interações com o usuário.

Figura 20 - Interface de usuário



Fonte:Produção própria

Utilizando recursos das bibliotecas CGI do Python simplificamos sua estrutura para que facilmente possa ser executada usando o apache como servidor web. Isto permite que o usuário possa acionar os módulos que necessitem de intervenção.

Nesses casos quando o usuário clica em um dos objetos da pagina ele envia um comando para fila que logo será processado pelo serviço de filas descrito anteriormente.

4.5.8. Funcionamento

Primeiramente é necessária a criação das regras no sistema, criando a estrutura básica para seu funcionamento no caso devemos iniciar criando um usuário e senha na tabela usuário e após as regras para que tarefa seja executada como nos exemplos.

Módulo Anti Chuva: Criar as seguintes entradas na tabela de módulos, sendo elas, por exemplo:

Id	Nome	Comando	Porta	Estado
1	Anti-chuva	STATUS	/usr/ttyUSB0	-
2	Servo Motor	SRM	/usr/ttyUSB1	0

A primeira entrada representa o sensor de chuva conectado no Arduino na primeira porta USB do Raspberry PI e não tem estado inicial. A Segunda entrada representa o servo motor a ser acionado do Arduino conectado na segunda porta do Raspberry PI com estado inicial do servo em zero grau.

Após é necessário criar uma entrada na tabela tarefas, condições e horários para este modulo como por exemplo.

Id	Idmodulo	Nome
100	1	STATUS CHUVA

Esta tabela serve apenas para relacionar as tarefas e horários caso haja mais de um.

Idtarefa	Valor	Operador	Atuador	Comando
100	90	>	2	SRM90

Perceba que aqui são atrelados os id dos dois módulos, por exemplo, sempre que o tarefa 100 (STATUS CHUVA) que está relacionado ao modulo 1 (Anti-chuva) retornar o valor ">" (maior) "90", será enviado o comando **"SRM90"** para o modulo 2 (servo motor), que girará os seu eixo em 90 graus e assim abra a janela.

Id	Hora inicial	Hora final	Repetir
100	Null	Null	30

A tabela de horário serve para indicar quando a tarefa deve ser executada no caso da tabela acima diz que a tarefa de checagem deve ocorrer de 30 em 30 segundos.

Módulo de Iluminação: Criar as seguintes entradas na tabela dos módulos, por exemplo:

Id	Nome	Comando	Porta	Estado
3	Sensor de presença	STATUS	/usr/ttyUSB0	-
4	Sensor luminosidade	STATUS	/usr/ttyUSB0	-
5	Luzes	RG0	/usr/ttyUSB1	000000000

Onde o primeiro representa o sensor de presença e sensor luminosidade conectado no Arduino na primeira porta USB do Raspberry PI e não tem estado inicial. A Segunda entrada representa o os leds a serem acionados no Arduino conectado na segunda porta do Raspberry PI com estado inicial em **000000000** (Apagado). Após é necessário criar uma entrada na tabela tarefas, condições e horários para este modulo como por exemplo.

Id	Idmodulo	Nome
101	3	Iluminação

Esta tabela serve apenas para relacionar as tarefas e horários caso haja mais de um.

Idtarefa	Valor	Operador	Atuador	Comando
101	2000	<	4	RGB255255255
101	True	=	4	RGB255255255

Sempre que o tarefa 101 (STATUS PRESENÇA) que está relacionado ao modulo 3 (Sensor de presença) retornar o valor "<" (menor) "2000" e 4 (Sensor de luminosidade) retornar "true", será enviado o comando **"RGB255255255"** para o modulo 5 (Luzes), que acendera na cor branca. Note que neste caso ambas as condições devem ser verdadeiras.

Id	Hora inicial	Hora final	Repetir
101	0000-00-00 17:00:00	0000-00-00 06:00:00	15

A tabela horário, serve para indicar quando a tarefa deve ser executada no caso da tabela acima diz que a tarefa de checagem deve ocorrer todos os dias entre

17 horas e 6 horas e deve ser repetida a cada 15 segundos.

Módulo termostato: Criar as seguintes entradas na tabela dos módulos, sendo elas, por exemplo:

Id	Nome	Comando	Porta	Estado
6	Sensor de Calor	STATUS	/usr/ttyUSB0	-
7	FAN1	SEM	/usr/ttyUSB1	0

Onde o primeiro representa o sensor de calor conectado no Arduino na primeira porta USB do Raspberry PI e não tem estado inicial. A Segunda entrada representa o FAN1 que representa o ar condicionado a ser aciona do Arduino conectado na segunda porta do Raspberry PI.

Após é necessário criar uma entrada na tabela tarefas, condições e horários para este modulo como por exemplo.

Id	Idmodulo	Nome
102	1	Temperatura

Está tabela serve apenas para relacionar as tarefas e horários caso haja mais de um.

Idtarefa	Valor	Operador	Atuador	Comando
102	21	>	7	FAN1ON

Sempre que o tarefa 102 (Temperatura) que está relacionado ao modulo 6 (Sensor de temperatura) retornar o valor ">" (maior) "21", será enviado o comando "FAN1ON" para o modulo 7 (FAN).

Id	Hora inicial	Hora final	Repetir
102	<i>Null</i>	<i>Null</i>	30

A tabela de horário serve para indicar quando a tarefa deve ser executada no caso da tabela acima diz que a tarefa de checagem deve ocorre de 30 em 30 segundos.

Módulo de Alarme: Criar as seguintes entradas na tabela dos módulos, sendo elas, por exemplo:

Id	Nome	Comando	Porta	Estado
8	Alarme	ALRM	/usr/ttyUSB1	0
3	Sensor de presença	STATUS	/usr/ttyUSB0	-

Onde o primeiro representa o sensor de presença conectado no Arduino na primeira porta USB do Raspberry PI e não tem estado inicial. A Segunda entrada representa o Buzzer que será acionado do Arduino conectado na segunda porta do Raspberry PI.

Após é necessário criar uma entrada na tabela tarefas, condições e horários para este módulo como por exemplo.

Id	Idmódulo	Nome
103	1	Alarme

Esta tabela serve apenas para relacionar as tarefas e horários caso haja mais de um.

Idtarefa	Valor	Operador	Atuador	Comando
103	True	=	8	BUZ15500

Sempre que o tarefa 103 (Alarme) que está relacionado ao módulo 3 (Sensor de presença) retornar o valor “=” (igual) “true”, será enviado o comando “BUZ15500” para o módulo 8 (Alarme).

Id	Hora inicial	Hora final	Repetir
103	<i>Null</i>	<i>Null</i>	60

A tabela de horário serve para indicar quando a tarefa deve ser executada no caso da tabela acima diz que a tarefa de checagem deve ocorrer de 1 em 1 minuto.

Todas as representações acima citadas descrevem as características dos módulos propostos sendo eles "Módulo Anti Chuva", "Módulo Termostato", "Módulo de Iluminação", "Módulo de Alarme". Com as configurações citadas será possível reproduzi-los integralmente e com base nas mesmas regras pode integrar quaisquer outros módulos ao sistema.

Dessa maneira conseguimos demonstrar o funcionamento de todos os módulos. No próximo capítulo vamos revelar nossas conclusões acerca do sistema e seu funcionamento.

5. CONCLUSÕES

Durante nossa graduação fomos apresentados a muitos conceitos e tecnologias que nos aproximaram do cativante mundo do software e hardware livres, direcionando a escolha desse tema. Percebemos que aplicações residenciais de controle podem ser desenvolvidas utilizando-se componentes de baixo custo e programação caseira. Com isso visamos à popularização das casas inteligentes para facilitar a vida das pessoas e tornar a tecnologia acessível às classes menos favorecidas. Mas para que esta popularização se dê de forma correta, é necessário investimento na criação de componentes simples e seguros e na padronização de estruturas de rede de alto desempenho. Este documento apresentou de forma geral nosso estudo sobre automação residencial utilizando componentes livre e abertos. Demonstramos as etapas de desenvolvimento uma a uma com especificações, diagrama, circuitos e pinagens, sempre visando o detalhamento das informações para que o leitor possa recriar o sistema e ter um bom funcionamento de cada módulo, sempre buscando um produto final de qualidade e consistência.

Com o início da fase de idealização dos módulos, detectamos que o hardware escolhido para a ligação Arduino com Raspberry PI (Shield), não atenderiam aos nossos anseios iniciais para o projeto de ser barato e livre. A grande dificuldade foi à aquisição do Shield connection bridge, pois além de proprietário o seu preço pra ser adquirido no Brasil inviabilizaria o projeto. Como planejado cada módulo foi desenvolvido individualmente e tínhamos que conectar os Arduino ao Raspberry PI de alguma forma, assim de uma maneira pouco agradável aprendemos com ocorrido.

Como conseguimos descobrir ainda na fase do desenvolvimento que integração dos módulos não seria possível via shield o que simplificaria muito o projeto, medidas de contingência foram adotadas. A medida adotada foi à alteração da forma de conexão, nos forçando a pensar e pesquisar novas formas.

Após pesquisas mais aprofundadas no projeto dos hardwares que poderiam suprir as especificações do projeto, decidimos então adotar a Conexão Serial usando cabos USB. Essa mudança trouxe muitos benefícios ao projeto, um grande diferencial desta forma de conexão é que além de ser um padrão, a mesma permite interligar e prover alimentação elétrica a todo o sistema, proporcionando a

interligação de outros módulos tornando o sistema expansível, tal como visto com a utilização de Shields.

Após a definição da utilização da conexão serial, foi iniciada toda a parte de desenvolvimento do protótipo, onde conseguimos validar e testar grande parte dos itens propostos no documento. Porém ainda durante os testes iniciais constatamos um problema na utilização da conexão serial.

A dificuldade ficou por conta de contornar a característica de "auto-reset" do Arduino. O auto-reset é um reset automático do microcontrolador do AVR quando uma conexão serial é estabelecida. Ele foi introduzido no Arduino Diecimila, estando presente inclusive nos modelos Uno e Duemilanove e em alguns modelos seriais RS-232(Arduino, 2011). Basicamente o problema é que quando uma conexão serial é estabelecida, o sinal DTR da interface RS232 (ou o equivalente na interface serial simulada na USB) é acionado; o hardware do Arduino (ou o firmware do ATmega8U2 usado para implementar a USB) força o reset do microcontrolador para zero neste momento. O ideal seria o Arduino tivesse um jumper para ligar ou desligar o auto-reset. Infelizmente, nenhum modelo atual tem esta facilidade, o UNO e o Duemilanove possuem duas ilhas interligadas por uma fina trilha; cortando a trilha o auto-reset é desligado. Para ligá-lo novamente é preciso reconectar as duas ilhas, infelizmente não quisemos arrisca danificar o hardware removendo a solda dos pinos. A solução ideal foi colocar um capacitor de 10 uF entre o sinal de reset e o terra. Esse não foi o único problema, houve outro muito sério que podia ter acabado com o projeto, constatamos que não é possível haver mais de um acesso simultâneo no mesmo Arduino, pois o Arduino impede que isso ocorra. O que ocorre é que se não houvesse esse travamento poderia gerar um deadlock¹⁷, e o Arduino travar. Dessa maneira a solução foi criar uma fila de comandos a serem enviados par o Arduino.

Na etapa de conclusão do projeto, tivemos alguns imprevistos com a construção e a integração dos módulos. Pela nossa falta de experiência perdemos um tempo enorme com um motor de passo, pois o mesmo estava apresentando mau funcionamento o que gerava curto no CI (Circuito integrado). Demoramos a

¹⁷ Propriedade (indesejável) de um conjunto de processos que estão bloqueados indefinidamente esperando uns pelos outros.

Mais em: <http://homepages.dcc.ufmg.br/~scampos/cursos/so/aulas/aula9.html> Acesso: 21 de Nov. 2013

identificar que o defeito era do motor, então ao descobrir mudamos para o servo motor.

Outro imprevisto que é muito importante citar foi o problema de alimentação no Raspberry PI, quando iniciamos a sua utilização por vezes o sistema se mostrava lento e quando um periférico USB era conectado o mesmo desligava. Ao utilizarmos um Multímetro vimos que não passava energia suficiente para alimentar o Raspberry PI, pois o Cabo USB utilizado era de má qualidade e não passava a corrente necessária de 700mAh, a solução foi à troca do cabo por um de melhor qualidade e a substituição da fonte por uma de 2000mAh¹⁸. Ainda relacionado à fonte perdemos alguns CI que foram substituídos prontamente.

Algumas melhorias futuras seriam a criação de uma interface para o cadastramento de usuários e demais informações no banco de dados, a criação de um Appliance¹⁹, com uma placa de circuito impresso eliminando os fios e tornando o sistema pronto para ser comercializado ou distribuído em forma de crowdfunding (financiamento coletivo)²⁰. Almejamos também a criação de outros módulos para a automação de mais áreas e tarefas dentro de uma casa.

Consideramos o presente projeto concluído, o mesmo atingiu os objetivos propostos e declarados no escopo do projeto, claro com algumas alterações necessárias observadas durante o processo, mas de maneira geral bem próximo ao idealizado.

Esse projeto serviu como uma grande experiência e aprendizado para os membros da equipe. Aprendemos a lidar com prazos, riscos funcionais do projeto, medidas de contingência e o mais importante de tudo entender que planejar um projeto desse porte, requer muito estudo e conhecimento das tecnologias existentes, e, além disso, gera uma imensa responsabilidade para os desenvolvedores.

Face ao exposto documento, terminamos este projeto com a sensação e o sentimento de dever cumprido.

¹⁸ **mAh** é uma unidade de carga elétrica. Um ampère-hora é igual a 3600 Coulomb (ampère-segundo), e é a quantidade de carga elétrica transferida por uma corrente estável de um ampère durante uma hora.

¹⁹ No mundo da informática, as Appliance são computadores pré-configurados para executar tarefas específicas, como compartilhar a conexão ou um firewall para a rede, como um quiosque multimídia, um sistema de caixa registradora, um centro de multimídia, ou um centro de controle de automatização doméstica e assim por diante. (Morimoto, 2005)

²⁰ Maiores detalhes disponível em: <http://crowdfunding.com/> Acesso em: 21 de Nov. 2013.

REFERÊNCIAS

- Alves, J. A., & Mota, J. (2003). Casas inteligentes. Revista New World (p. 141). Lisboa: Centro Atlântico.
- Angerhausen, M. (1984). THE ANATOMY OF THE COMMODORE 64 (1st ed., p. 304). Grand Rapids: Abacus Software.
- Arduino, T. (2011). Disabling Auto Reset On Serial Connection. Retrieved from <http://playground.arduino.cc/Main/DisablingAutoResetOnSerialConnection>
- Deboni, M. L., Engel, C., Alvarez, D., & Bissoli, M. (2011). Automação sustentável : uma nova visão do emprego de tecnologias na construção civil. ELECS - Encontro Latinoamericano de Edificações e Comunidades Sustentáveis, 1(1), 10.
- Grego, M. (2009). O hardware em “código aberto.” Info Exame.
- Morimoto, C. E. (2005). Appliance. Retrieved from <http://www.hardware.com.br/termos/appliance>
- Schmidt, M. (2011). Arduino: A Quick-Start Guide (p. 263). Pragmatic Bookshelf.
- DIAS, C. L. A.; PIZZOLATO, N. D. DOMÓTICA Aplicabilidade e Sistemas de Automação Residencial. Vértices Publicação Técnico-científica do CEFET, Campos dos Goytacazes, v.6, n.3, p.9-32, 2004.
- FILHO, C. F. História da Computação: O Caminho Do Pensamento E Da Tecnologia. Porto Alegre, RS - BRASIL: EDIPUCRS, 2007. 205p.
- LAKATOS, E. M.; MARCONI, M. A. Metodologia Científica. 6ª Edição. ed. São Paulo: Atlas, 2011. 312p.
- LÜCKMANN, L. C.; ROVER, A.; VARGAS, M. Diretrizes para elaboração de trabalhos científicos. 4ª Edição. ed. Joaçaba: Unoesc, 2010. 0p. Caderno 1 da série: Metodologia do trabalho científico

MAZZAROPPI, M. Sensores de movimento e presença. 2007. 54f. Monografia (Bacharelado em Engenharia Elétrica) - UFRJ, Rio de Janeiro, 2007.

PUDENTE, F. Automação predial e residencial: uma introdução. Rio de Janeiro: Livros Técnicos e Científicos Editora Ltda., 2011. 211p.

SGARBI, J. A.; TONIDANDEL, F. Domótica Inteligente: Automação Residencial baseada. 2011. 10f. Monografia (Tecnologia em Mecatrônica) - Centro Universitário da FEI – UniFEI, São Bernardo do Campo, 2011.

SILVEIRA, J. A. Experimentos com Arduino: monte seus próprios projetos com o Arduino utilizando as linguagens C e Processing. São Paulo: Ensino Profissional, 2011. 199p.

A Computer in the Basement? (Um Computador no Porão?). Disponível em: <http://books.google.com.br/books?id=AtQDAAAAMBAJ&pg=PA77&source=gbbs_toc_r&redir_esc=y#v=onepage&q&f=false> Acesso em: 14 set. 2013

Arduino. Disponível em: <<http://arduino.cc>> Acesso em: 14 set. 2013

Arduino Playground. Disponível em: <<http://playground.arduino.cc/Code/2B>> Acesso em: 15 set. 2013

Bruce Perens. Disponível em: <<http://perens.com/>> Acesso em: 15 set. 2013

Commodore 64 Vs. ZX Spectrum. Disponível em: <<http://www.c64vsspectrum.com/>> Acesso em: 12 out. 2013

Dispositivos X10. Disponível em: <[http://en.wikipedia.org/wiki/X10_\(industry_standard\)](http://en.wikipedia.org/wiki/X10_(industry_standard))> Acesso em: 15 set. 2013

Freetronics.com. Disponível em: <<http://www.freetronics.com/>> Acesso em: 15 set. 2013

Nest. Disponível em: <<http://nest.com/>> Acesso em: 15 set. 2013

O que é domótica. Disponível em: <<http://www.din.uem.br/ia/intelige/domotica/int.htm>> Acesso em: 14 set. 2013

Raspberry PI Faq's. Disponível em: <<http://www.raspberrypi.org/faqs>> Acesso em: 15 set. 2013

Raspbian. Disponível em: <<http://www.raspbian.org/>> Acesso em: 13 out. 2013

SABER ELETRÔNICA. A onda do Arduino. Disponível em: <<http://www.sabereletronica.com.br/artigos/2096-a-onda-do-arduino>> Acesso em: 14 set. 2013

Shield Arduino para Raspberry PI. Disponível em: <<http://www.cooking-hacks.com/index.php/documentation/tutorials/raspberry-pi-to-arduino-shields-connection-bridge>> Acesso em: 15 set. 2013

Shields Arduino. Disponível em: <<http://tronixstuff.com/>> Acesso em: 15 set. 2013

Significado de escopo. Disponível em: <<http://www.significados.com.br/escopo/>> Acesso em: 13 out. 2013

Sistemas de Arquivo. Disponível em: <<http://www.clubedohardware.com.br/printpage/313>> Acesso em: 13 out. 2013

APÊNDICE A – CÓDIGO ARDUINO SENSORES

```

#define CHUVA A0 // entrada A0 analogica
#define LDR A2 // entrada A2 analogica
#define MOVIMENTO 3// entrada 3 digital
#define TERMOSTATO1 A1 // entrada A1 analogica
#define TERMOSTATO2 A3 // entrada A3 analogica

void Status ();
void Execulte(String str);

float chuva =0;
float ldr =0;
int movimento =0;
float termostato1 =0;
float termostato2 =0;

String readString;

void setup ()
{
  pinMode ( CHUVA, INPUT ); // inicia sensor chuva
  pinMode ( LDR, INPUT ); // inicia sensor ldr
  pinMode( MOVIMENTO, INPUT ); // inicia sensor de movimento
  pinMode ( TERMOSTATO1, OUTPUT ); //inicia sensor termostato
  pinMode ( TERMOSTATO2, OUTPUT); //inicia sensor termostato
  Serial.begin( 9600 ); // porta serial
}

void loop(){

  while(Serial.available()){

    //Espera 3 milesegundos
    delay(3);

    //Caputra char da entrada
    char c = Serial.read();
    //Adciona no final da String readString.
    readString +=c;

  }
  if(readString.length()>1){
    //Execulta o metodo "Execulte"
    Execulte(readString);
    //Limpa o conteudo do readString

    readString="";

  }

}

void Execulte(String str){

  //verfica se o commando enviado é valido
  int valid=0;

  //Define que a string value deve possuir penas 2 caracteres
  String value = str.substring(0,5);

  //Cria um array de char de 2 caracteres para armazenar a String value
  char charBuf[str.length()];

```

```

//Adciona a String value dentro o array charBuf
value.toCharArray(charBuf, 5);

if(strcmp(charBuf,"STUS")==0){ Status();valid=1;}

if(valid==0)Serial.println("CMD: Invalido");
}

void Status ()
{
    // lê os valores obtidos pelo sensor de chuva
    chuva = analogRead (CHUVA);

    // lê os valores obtidos pelo sensor ldr
    ldr = analogRead (LDR);

    // lê os valores obtidos pelo sensor de movimento
    movimento = digitalRead (MOVIMENTO);

    // lê os valores obtidos pelo sensor termostato1
    termostato1 = analogRead (TERMOSTATO1);

    // lê os valores obtidos pelo sensor termostato2
    termostato2 = analogRead (TERMOSTATO2);

    Serial.print ("CHV0:");
    Serial.println(chuva);

    Serial.print ("LDR0:");
    Serial.println(ldr);

    Serial.print ("TER0:");
    Serial.println(termostato1);

    Serial.print ("TER1:");
    Serial.println(termostato2);

    Serial.print ("MOV0:");
    Serial.println(movimento);
}

```

APÊNDICE B – CÓDIGO ARDUINO ATUADORES

```
//Biblioteca necessaria dos C
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <Servo.h>

////Define todos os pinos a serem usados

//--- LED RGB 0
#define RED0 13 //Definindo o pino do led vermelho
#define GREEN0 12 //Definindo o pino do led verde
#define BLUE0 11 //Definindo o pino do led azul

//--- LED RGB 1
#define RED1 10 //Definindo o pino do led vermelho
#define GREEN1 9 //Definindo o pino do led verde
#define BLUE1 8 //Definindo o pino do led azul

//--- LED RGB 2
#define RED2 7 //Definindo o pino do led vermelho
#define GREEN2 6 //Definindo o pino do led verde
#define BLUE2 5 //Definindo o pino do led azul

//--- LED RGB 3
#define RED3 4 //Definindo o pino do led vermelho
#define GREEN3 3 //Definindo o pino do led verde
#define BLUE3 2 //Definindo o pino do led azul

//-- BUZZER
#define ABU A0 //Definindo o pino do buzzer

//-- SERVO MOTOR
#define SRM A5 //Definindo o pino do servo motor

//-- FAN0
#define FAN0 A2

//-- FAN1
#define FAN1 A3

////Declara todas as variaveis globais

//readString recebe um valor temporario para o input de dados.
String readString;

//cmdBuzzer recebe um valor temporario para o input de dados na função OnOffABU0
String cmdBuzzer = "";

//servoMain recebe o valor temporario da função OnOffSRM0, para executar.
Servo servoMain;

//abuzzer recebe da função OnOffABU0, um valor booleano que ativa ou nao o alarme.
boolean abuzzer = false;

////Declaração previamente das funções usadas.

void Execulte(String str);
void OnOffLEDRGB0(String str);
void OnOffLEDRGB1(String str);
void OnOffLEDRGB2(String str);
void OnOffLEDRGB3(String str);
void OnOffABU0(String str, boolean r);
```

```

void OnOffSRM0(String str);
void OnOffFAN0(String str);
void OnOffFAN1(String str);
void OnRepeat();

//Setup Inicia todos os pinos necessarios no bootloader
void setup(){

    //Inicia serial com velocidade 9600
    Serial.begin(9600);

    //Printa em tela assim que e iniciado o Serial Monitor
    // Serial.println("ARDCODE 1.0.1");

    //Inicia os pinos definido para o conjunto de cores do LED0
    pinMode(RED0, OUTPUT);
    pinMode(GREEN0, OUTPUT);
    pinMode(BLUE0, OUTPUT);

    //Inicia os pinos definido para o conjunto de cores do LED1
    pinMode(RED1, OUTPUT);
    pinMode(GREEN1, OUTPUT);
    pinMode(BLUE1, OUTPUT);

    //Inicia os pinos definido para o conjunto de cores do LED2
    pinMode(RED2, OUTPUT);
    pinMode(GREEN2, OUTPUT);
    pinMode(BLUE2, OUTPUT);

    //Inicia os pinos definido para o conjunto de cores do LED3
    pinMode(RED3, OUTPUT);
    pinMode(GREEN3, OUTPUT);
    pinMode(BLUE3, OUTPUT);

    //Inicia o pino definido para o BUZZER
    pinMode(ABU, OUTPUT);

    //Inicia o pino definido para o SERVO MOTOR
    servoMain.attach(SRM);

    //Inicia o pino definido para o primeiro FAN
    pinMode(FAN0, OUTPUT);

    //Inicia o pino definido para o segundo FAN
    pinMode(FAN1, OUTPUT);

}

//Exculta em loop todas as funções.
void loop(){

    //Re executa ativos, no caso usados na funcao OnOffABU0
    OnRepeat();

    //http://arduino.cc/en/Serial/Available
    //Verica se a serail está disponivel.
    while(Serial.available()){

        //Espera 3 milesegundos
        delay(3);

        //Caputra char da entrada
        char c = Serial.read();
        //Adciona no final da String readString.
        readString +=c;
    }
}

```

```

    }
    //Se a String readString for conter mais de 0 caracteres
    if(readString.length()>0){
        //Execulta o metodo "Execulte"
        Execulte(readString);
        //Limpa o conteudo do readString
        readString="";
    }
}

//Metodo que trada a entrada de dados Capturados no metodo LOOP
void Execulte(String str){

    //verfica se o commando enviado é valido
    int valid=0;

    //Define que a string value deve possuir penas 2 caracteres
    String value = str.substring(0,5);

    //Cria um array de char de 2 caracteres para armazenar a String value
    char charBuf[str.length()];

    //Adciona a String value dentro o array charBuf
    value.toCharArray(charBuf, 5);

    //Compara se a entrada de value é igual a LED0 se for igual execulta o metodo
    OnOffLEDRGB0 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"LED0")==0){ OnOffLEDRGB0(str);valid=1;}

    //Compara se a entrada de value é igual a LED1 se for igual execulta o metodo
    OnOffLEDRGB1 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"LED1")==0){ OnOffLEDRGB1(str); valid=1;}

    //Compara se a entrada de value é igual a LED2 se for igual execulta o metodo
    OnOffLEDRGB2 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"LED2")==0){ OnOffLEDRGB2(str); valid=1;}

    //Compara se a entrada de value é igual a LED3 se for igual execulta o metodo
    OnOffLEDRGB3 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"LED3")==0){ OnOffLEDRGB3(str); valid=1;}

    //Compara se a entrada de value é igual a ABU0 se for igual execulta o metodo
    OnOffABU0 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"ABU0")==0){ OnOffABU0(str, true); valid=1;}

    //Compara se a entrada de value é igual a SRM0 se for igual execulta o metodo
    OnOffSRM0 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"SRM0")==0){ OnOffSRM0(str); valid=1;}

    //Compara se a entrada de value é igual a FAN0 se for igual execulta o metodo
    OnOffFAN0 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"FAN0")==0){ OnOffFAN0(str); valid=1;}

    //Compara se a entrada de value é igual a FAN1 se for igual execulta o metodo
    OnOffFAN1 passando como parametro toda a string recebida no inicio do metodo Execute.
    if(strcmp(charBuf,"FAN1")==0){ OnOffFAN1(str); valid=1;}

    //Ser valor for falso responde comando invalido!
    if(!valid) Serial.println("CMD: INVALID");

}

//Função responsavel pelo conjunto de LED0
void OnOffLEDRGB0(String str){

```

```

//Verifica o tamanho da variavel str.
if(str.length()){

    char charBufR[4]; //Cria a variavel charBufR com tamanho para quatro caracteres
    sendo tres caracteres validos e do tipo char.
    char charBufG[4]; //Cria a variavel charBufG com tamanho para quatro caracteres
    sendo tres caracteres validos e do tipo char.
    char charBufB[4]; //Cria a variavel charBufB com tamanho para quatro caracteres
    sendo tres caracteres validos e do tipo char.

    //Recebe o conteúdo dentro da variavel str como por exemplo LED0255000255,
    //pegando os primeiros sete caracteres, sendo estes LED0255, e depois dividu-os
    em duas partes
    //os primeiros quatro caracteres são reconhecidos como um comando, como por
    exemplo LED0,
    //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
    esta divisão os caracteres 255 são armazenados dentro
    //da variavel charBufR.
    str.substring(4,7).toCharArray(charBufR, 4);

    //Recebe o conteúdo dentro da variavel str como por exemplo LED0255000255,
    //pegando os primeiros dez caracteres, sendo estes LED0255000, e depois dividu-
    os em duas partes
    //os primeiros sete caracteres são reconhecidos como um comando, como por
    exemplo LED0255,
    //a segunda parte são os parâmetros, que no caso são os caracteres 000. Após
    esta divisão os caracteres 000 são armazenados dentro
    //da variavel charBufG.
    str.substring(7,10).toCharArray(charBufG, 4);

    //Recebe o conteúdo dentro da variavel str como por exemplo LED0255000255,
    //pegando os 13 caracteres, sendo estes LED0255000255, e depois dividu-os em
    duas partes
    //os primeiros 10 caracteres são reconhecidos como um comando, como por exemplo
    LED0255000,
    //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
    esta divisão os caracteres 255 são armazenados dentro
    //da variavel charBufB.
    str.substring(10,13).toCharArray(charBufB, 4);

    int iR = atoi(charBufR); //atoi converte os caracteres 255 em inteiro da
    variavel charBufR, passando para a variavel iR de tipo inteiro.
    int iG = atoi(charBufG); //atoi converte os caracteres 000 em inteiro da
    variavel charBufG, passando para a variavel iG de tipo inteiro.
    int iB = atoi(charBufB); //atoi converte os caracteres 255 em inteiro da
    variavel charBufB, passando para a variavel iB de tipo inteiro.

    analogWrite(RED0,iR); //Liga a Cor vermelha "RED0" do LED0 com valor de 0 a 255
    que esta dentro da variavel iR.

    Serial.print("LED0:"); //Printa em tela LED0:.
    Serial.print(iR); //Printa em tela o valor dentro da Variavel iR, podendo ser
    de 0 a 255.
    Serial.print(" "); //Printa em tela um espaço em branco.

    analogWrite(GREEN0,iG); //Liga a Cor verde "GREEN0" do LED0 com valor de 0 a 255
    que esta dentro da variavel iG.
    Serial.print(iG); //Printa em tela o valor dentro da Variavel iG, podendo ser
    de 0 a 255.
    Serial.print(" "); //Printa em tela um espaço em branco.

    analogWrite(BLUE0, iB); //Liga a Cor verde "BLUE0" do LED0 com valor de 0 a 255
    que esta dentro da variavel iB.

```



```

        Serial.println(iB); //Printa em tela o valor dentro da Variavel iB, podendo ser
de 0 a 255.

    }
}

//Função responsável pelo conjunto de LED1
void OnOffLEDRGB1(String str){

    //Verifica o tamanho da variavel str.
    if(str.length()){

        char charBufR[4]; //Cria a variavel charBufR com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.
        char charBufG[4]; //Cria a variavel charBufG com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.
        char charBufB[4]; //Cria a variavel charBufB com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.

        //Recebe o conteúdo dentro da variavel str como por exemplo LED1255000255,
        //pegando os primeiros sete caracteres, sendo estes LED1255, e depois dividu-os
em duas partes
        //os primeiros quatro caracteres são reconhecidos como um comando, como por
exemplo LED1,
        //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
esta divisão os caracteres 255 são armazenados dentro
        //da variavel charBufR.
        str.substring(4,7).toCharArray(charBufR, 4);

        //Recebe o conteúdo dentro da variavel str como por exemplo LED1255000255,
        //pegando os primeiros dez caracteres, sendo estes LED1255000, e depois dividu-
os em duas partes
        //os primeiros sete caracteres são reconhecidos como um comando, como por
exemplo LED1255,
        //a segunda parte são os parâmetros, que no caso são os caracteres 000. Após
esta divisão os caracteres 000 são armazenados dentro
        //da variavel charBufG.
        str.substring(7,10).toCharArray(charBufG, 4);

        //Recebe o conteúdo dentro da variavel str como por exemplo LED1255000255,
        //pegando os 13 caracteres, sendo estes LED1255000255, e depois dividu-os em
duas partes
        //os primeiros 10 caracteres são reconhecidos como um comando, como por exemplo
LED1255000,
        //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
esta divisão os caracteres 255 são armazenados dentro
        //da variavel charBufB.
        str.substring(10,13).toCharArray(charBufB, 4);

        int iR = atoi(charBufR); //atoi converte os caracteres 255 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.
        int iG = atoi(charBufG); //atoi converte os caracteres 000 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.
        int iB = atoi(charBufB); //atoi converte os caracteres 255 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.

        analogWrite(RED1,iR); //Liga a Cor vermelha "RED1" do LED1 com valor de 0 a 255
que esta dentro da variavel iR.

        Serial.print("LED1:"); //Printa em tela LED1:.
        Serial.print(iR); //Printa em tela o valor dentro da Variavel iR, podendo ser
de 0 a 255.
        Serial.print(" "); //Printa em tela um espaço em branco.

        analogWrite(GREEN1,iG); //Liga a Cor verde "GREEN1" do LED1 com valor de 0 a 255

```

```

que esta dentro da variavel iG.
    Serial.print(iG); //Printa em tela o valor dentro da Variavel iG, podendo ser
de 0 a 255.
    Serial.print(" "); //Printa em tela um espaço em branco.

    analogWrite(BLUE1, iB); //Liga a Cor verde "BLUE1" do LED1 com valor de 0 a 255
que esta dentro da variavel iB.
    Serial.println(iB); //Printa em tela o valor dentro da Variavel iB, podendo ser
de 0 a 255.

    }
}

//Função responsável pelo conjunto de LED2
void OnOffLEDRGB2(String str){

    //Verifica o tamanho da variavel str.
    if(str.length()){

        char charBufR[4]; //Cria a variavel charBufR com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.
        char charBufG[4]; //Cria a variavel charBufG com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.
        char charBufB[4]; //Cria a variavel charBufB com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.

        //Recebe o conteúdo dentro da variavel str como por exemplo LED2255000255,
        //pegando os primeiros sete caracteres, sendo estes LED2255, e depois dividu-os
em duas partes
        //os primeiros quatro caracteres são reconhecidos como um comando, como por
exemplo LED2,
        //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
esta divisão os caracteres 255 são armazenados dentro
        //da variavel charBufR.
        str.substring(4,7).toCharArray(charBufR, 4);

        //Recebe o conteúdo dentro da variavel str como por exemplo LED2255000255,
        //pegando os primeiros dez caracteres, sendo estes LED2255000, e depois dividu-
os em duas partes
        //os primeiros sete caracteres são reconhecidos como um comando, como por
exemplo LED2255,
        //a segunda parte são os parâmetros, que no caso são os caracteres 000. Após
esta divisão os caracteres 000 são armazenados dentro
        //da variavel charBufG.
        str.substring(7,10).toCharArray(charBufG, 4);

        //Recebe o conteúdo dentro da variavel str como por exemplo LED2255000255,
        //pegando os 13 caracteres, sendo estes LED1255000255, e depois dividu-os em
duas partes
        //os primeiros 10 caracteres são reconhecidos como um comando, como por exemplo
LED2255000,
        //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
esta divisão os caracteres 255 são armazenados dentro
        //da variavel charBufB.
        str.substring(10,13).toCharArray(charBufB, 4);

        int iR = atoi(charBufR); //atoi converte os caracteres 255 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.
        int iG = atoi(charBufG); //atoi converte os caracteres 000 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.
        int iB = atoi(charBufB); //atoi converte os caracteres 255 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.

        analogWrite(RED2,iR); //Liga a Cor vermelha "RED2" do LED2 com valor de 0 a 255
que esta dentro da variavel iR.

```

```

        Serial.print("LED2:"); //Printa em tela LED2:.
        Serial.print(iR); //Printa em tela o valor dentro da Variavel iR, podendo ser
de 0 a 255.
        Serial.print(" "); //Printa em tela um espaço em branco.

        analogWrite(GREEN2,iG); //Liga a Cor verde "GREEN2" do LED2 com valor de 0 a 255
que esta dentro da variavel iG.
        Serial.print(iG); //Printa em tela o valor dentro da Variavel iG, podendo ser
de 0 a 255.
        Serial.print(" "); //Printa em tela um espaço em branco.

        analogWrite(BLUE2, iB); //Liga a Cor verde "BLUE2" do LED2 com valor de 0 a 255
que esta dentro da variavel iB.
        Serial.println(iB); //Printa em tela o valor dentro da Variavel iB, podendo ser
de 0 a 255.

    }
}

//Função responsável pelo conjunto de LED3
void OnOffLEDRGB3(String str){

    //Verifica o tamanho da variavel str.
    if(str.length()){

        char charBufR[4]; //Cria a variavel charBufR com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.
        char charBufG[4]; //Cria a variavel charBufG com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.
        char charBufB[4]; //Cria a variavel charBufB com tamanho para quatro caracteres
sendo tres caracteres validos e do tipo char.

        //Recebe o conteúdo dentro da variavel str como por exemplo LED3255000255,
        //pegando os primeiros sete caracteres, sendo estes LED3255, e depois dividindo-os
em duas partes
        //os primeiros quatro caracteres são reconhecidos como um comando, como por
exemplo LED3,
        //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
esta divisão os caracteres 255 são armazenados dentro
        //da variavel charBufR.
        str.substring(4,7).toCharArray(charBufR, 4);

        //Recebe o conteúdo dentro da variavel str como por exemplo LED3255000255,
        //pegando os primeiros dez caracteres, sendo estes LED3255000, e depois dividindo-
os em duas partes
        //os primeiros sete caracteres são reconhecidos como um comando, como por
exemplo LED3255,
        //a segunda parte são os parâmetros, que no caso são os caracteres 000. Após
esta divisão os caracteres 000 são armazenados dentro
        //da variavel charBufG.
        str.substring(7,10).toCharArray(charBufG, 4);

        //Recebe o conteúdo dentro da variavel str como por exemplo LED3255000255,
        //pegando os 13 caracteres, sendo estes LED3255000255, e depois dividindo-os em
duas partes
        //os primeiros 10 caracteres são reconhecidos como um comando, como por exemplo
LED3255000,
        //a segunda parte são os parâmetros, que no caso são os caracteres 255. Após
esta divisão os caracteres 255 são armazenados dentro
        //da variavel charBufB.
        str.substring(10,13).toCharArray(charBufB, 4);

        int iR =  atoi(charBufR); //atoi converte os caracteres 255 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.
        int iG =  atoi(charBufG); //atoi converte os caracteres 000 em inteiro da

```

```

variavel charBufR, passando para a variavel iR de tipo inteiro.
    int iB = atoi(charBufB); //atoi converte os caracteres 255 em inteiro da
variavel charBufR, passando para a variavel iR de tipo inteiro.

    analogWrite(RED3,iR); //Liga a Cor vermelha "RED3" do LED3 com valor de 0 a 255
que esta dentro da variavel iR.

    Serial.print("LED3:"); //Printa em tela LED3:.
    Serial.print(iR); //Printa em tela o valor dentro da Variavel iR, podendo ser
de 0 a 255.
    Serial.print(" "); //Printa em tela um espaço em branco.

    analogWrite(GREEN3,iG); //Liga a Cor verde "GREEN3" do LED3 com valor de 0 a 255
que esta dentro da variavel iG.
    Serial.print(iG); //Printa em tela o valor dentro da Variavel iG, podendo ser
de 0 a 255.
    Serial.print(" "); //Printa em tela um espaço em branco.

    analogWrite(BLUE3, iB); //Liga a Cor verde "BLUE3" do LED3 com valor de 0 a 255
que esta dentro da variavel iB.
    Serial.println(iB); //Printa em tela o valor dentro da Variavel iB, podendo ser
de 0 a 255.

    }
}

//Função responsavel pelo Buzzer
void OnOffABU0(String str, boolean r){

    //Se o comando possui caracteres.
    if(str.length()){

        //Torna a string cmdBuzzer iguai a str
        cmdBuzzer = str;

        //Define charBufHtz uma variavel char com tamando de cinco catacteres sendo
quatro validos
        char charBufHtz[5];

        //Define charBufDelay uma variavel char com tamando de cinco catacteres sendo
quatro validos
        char charBufDelay[5];

        //Recebe o conteúdo dentro da variavel str como por exemplo ABU0150002000,
//pegando os 9 caracteres, sendo estes ABU015000, e depois divindu-os em duas
partes
        //os primeiros quatro caracteres são reconhecidos como um comando, como por
exemplo ABU0,
        //a segunda parte são os parâmetros, que no caso são os caracteres 15000. Após
esta divisão os caracteres 15000 são armazenados dentro
        //da variavel charBufHtz.
        str.substring(4,9).toCharArray(charBufHtz, 5);

        //Recebe o conteúdo dentro da variavel str como por exemplo ABU0150002000,
//pegando os quatorze caracteres, sendo estes ABU0150002000, e depois divindu-os
em duas partes
        //os primeiros nove caracteres são reconhecidos como um comando, como por
exemplo ABU015000,
        //a segunda parte são os parâmetros, que no caso são os caracteres 2000. Após
esta divisão os caracteres 2000 são armazenados dentro
        //da variavel charBufG.
        str.substring(9,14).toCharArray(charBufDelay, 5);

        int H = atoi(charBufHtz); //atoi converte os caracteres 15000 em inteiro da

```

```

variavel charBufHtz, passando para a variavel H de tipo inteiro.
    int D = atoi(charBufDelay); //atoi converte os caracteres 2000 em inteiro da
variavel charBufDelay, passando para a variavel D de tipo inteiro.

    // Liga Buzzer
    if(charBufHtz, H){ //Verifica os valores das variaveis charBufHtz e H.

        abuzzer=true; //Indica que abuzzer é igual a false.

        tone(ABU, H); //Ativa o Buzzer ABU com a indicação de Hertz dentro da variavel
H.
        delay(D); //Ativa delay de tempo em milisegundos com a indicação de tempo
dentro da variavel D.

        noTone(ABU); //Desativa o Buzzer ABU
        delay(D); //Ativa delay de tempo em milisegundos com a indicação de tempo
dentro da variavel D.

        if (r){

            Serial.print("ABU0: "); //Escreve ABU0:
            Serial.print(H); //Escreve o conteudo da variavel H
            Serial.print(" "); //Escreve um espaço
            Serial.print(D); //Escreve o conteudo da variavel H
            Serial.println(" "); //Escreve um espaço

        }

        return; //Retorno.
    }

    // Desliga Buzzer
    if(charBufHtz, '0000'){ //Verifica o valor da variavel charBufHtz.

        abuzzer=false; //Indica que abuzzer é igual a false.

        //Desliga buzzer
        noTone(ABU); //Desativa o Buzzer ABU

        Serial.print("ABU0: "); //Escreve ABU0:
        Serial.print(H); //Escreve o conteudo da variavel H
        Serial.print(" "); //Escreve um espaço
        Serial.print(D); //Escreve o conteudo da variavel D

    }
}

//Função responsável pelo Servo Motor.
void OnOffSRM0(String str){

    //Verifica o tamanho da variavel str.
    if(str.length()){

        //Cria a variavel charBufGraus com tamanho para quatro caracteres sendo tres
caracteres validos e do tipo char.
        char charBufGraus[4];

        //Recebe o conteúdo dentro da variavel str como por exemplo SRM0180,
        //pegando os sete caracteres, sendo estes SRM0180, e depois dividindo-os em duas
partes
        //os primeiros quatro caracteres são reconhecidos como um comando, como por
exemplo SRM0,

```

```

        //a segunda parte são os parâmetros, que no caso são os caracteres 180. Após
esta divisão os caracteres 180 são armazenados dentro
        //da variavel charBufGraus.
        str.substring(4,7).toCharArray(charBufGraus, 4);

        int G = atoi(charBufGraus); //atoi converte os caracteres 180 em inteiro da
variavel charBufGraus, passando para a variavel G de tipo inteiro.

        if(charBufGraus, G){ //Verifica os valores das variaveis charBufGraus e G.

            servoMain.write(G); //Ativa o Servo Motor usando o valor dentro da variavel
G, para fazer a rotação, que seria 180 graus.

            Serial.print("SRM0:"); //Imprime em tela SRM0:.
            Serial.println(G); //Imprime em tela o valor da variavel G.

            return;
        }
    }
}

//Função responsável pelo FAN0
void OnOffFAN0(String str){

    //Se o comando possui mais de 5 caracteres.
    if(str.length()>5){

        String value = str.substring(4,str.length());
        char charBuf[value.length()];

        value.toCharArray(charBuf, value.length());

        //verifica se charBuf é igual a "ON".
        if(strcmp(charBuf,"ON")==0) {

            //Se verdadeiro liga o pino contido na variavel FAN0
            digitalWrite(FAN0,HIGH);

            //escreve FAN:ON
            Serial.println("FAN0:ON");

            //Retorna
            return;
        }

        //verifica se charBuf é igual a "OFF".
        if(strcmp(charBuf,"OFF")==0){

            //Se verdadeiro desliga o pino contido na variavel FAN0
            digitalWrite(FAN0,LOW);

            //escreve FAN0:OFF
            Serial.println("FAN0:OFF");

            //Retorna
            return;
        }
    }
}

//Função responsável pelo FAN1
void OnOffFAN1(String str){

    //Se o comando possui mais de 5 caracteres.
    if(str.length()>5){

```

```

String value = str.substring(4,str.length());
char charBuf[value.length()];

value.toCharArray(charBuf, value.length());

//verifca se charBuf é igual a "ON".
if(strcmp(charBuf,"ON")==0) {

    //Se verdadeiro liga o pino contido na variavel FAN1
    digitalWrite(FAN1,HIGH);

    //escreve FAN1:ON
    Serial.println("FAN1:ON");

    //Retorna
    return;
}

//verifca se charBuf é igual a "OFF".
if(strcmp(charBuf,"OFF")==0){

    //Se verdadeiro desliga o pino contido na variavel FAN1
    digitalWrite(FAN1,LOW);

    //escreve FAN1:OFF
    Serial.println("FAN1:OFF");

    //Retorna
    return;
}
}

//Repete Todas as funções necessarias.
void OnRepeat(){

    if(abuzzer)OnOffABU0(cmdBuzzer, false);

}

```

APÊNDICE C – CÓDIGO CENTRAL RASPBERRY PI

Código fonte da interface em Python 2.6.7.

Lista de arquivos.

```
|.
|..
|banco_de_dados/
|    |pi.sql
|css/
|    |site.css
|scripts/
|    |__init__.py
|    |CommandSerial.py
|    |DataBase.py
|    |filas.py
|    |filas_startup.sh
|    |functions.js
|    |json.py
|    |tarefas.py
|    |tarefas_startup.sh

|__init__.py
|home.py
|index.py
```

Aquivo banco_de_dados/pi.sql

```
-- MySQL dump 10.13  Distrib 5.5.31, for debian-linux-gnu (armv7l)
--
-- Host: localhost    Database: pi
-- -----
-- Server version    5.5.31-0+wheezy1

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `condicoes`
--

DROP TABLE IF EXISTS `condicoes`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `condicoes` (
  `idtarefas` int(11) NOT NULL,
  `valor` varchar(10) COLLATE latin1_general_ci NOT NULL,
  `operador` varchar(2) COLLATE latin1_general_ci NOT NULL,
  `atudador` int(11) NOT NULL,
  `comando` varchar(255) COLLATE latin1_general_ci DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `condicoes`
--
```



```

LOCK TABLES `condicoes` WRITE;
/*!40000 ALTER TABLE `condicoes` DISABLE KEYS */;
/*!40000 ALTER TABLE `condicoes` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `filas`
--

DROP TABLE IF EXISTS `filas`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `filas` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `usuario` varchar(20) COLLATE latin1_general_ci NOT NULL,
  `data` datetime NOT NULL,
  `comando` varchar(255) COLLATE latin1_general_ci NOT NULL,
  `tipo` varchar(2) COLLATE latin1_general_ci NOT NULL,
  `porta` varchar(255) COLLATE latin1_general_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `filas`
--

LOCK TABLES `filas` WRITE;
/*!40000 ALTER TABLE `filas` DISABLE KEYS */;
/*!40000 ALTER TABLE `filas` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `horarios`
--

DROP TABLE IF EXISTS `horarios`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `horarios` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `datainicial` datetime DEFAULT NULL,
  `datafinal` datetime DEFAULT NULL,
  `repetir` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `horarios`
--

LOCK TABLES `horarios` WRITE;
/*!40000 ALTER TABLE `horarios` DISABLE KEYS */;
/*!40000 ALTER TABLE `horarios` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `logs`
--

DROP TABLE IF EXISTS `logs`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `logs` (

```

```

`id` int(11) NOT NULL AUTO_INCREMENT,
`idtarefa` int(11) NOT NULL,
`data` datetime NOT NULL,
`usuario` varchar(20) COLLATE latin1_general_ci NOT NULL,
`texto` varchar(255) COLLATE latin1_general_ci NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `logs`
--

LOCK TABLES `logs` WRITE;
/*!40000 ALTER TABLE `logs` DISABLE KEYS */;
/*!40000 ALTER TABLE `logs` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `modulos`
--

DROP TABLE IF EXISTS `modulos`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `modulos` (
  `idmodulo` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(50) COLLATE latin1_general_ci NOT NULL,
  `comando` varchar(4) COLLATE latin1_general_ci NOT NULL,
  `porta` varchar(255) COLLATE latin1_general_ci NOT NULL,
  `estado` varchar(20) COLLATE latin1_general_ci NOT NULL,
  PRIMARY KEY (`idmodulo`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `modulos`
--

LOCK TABLES `modulos` WRITE;
/*!40000 ALTER TABLE `modulos` DISABLE KEYS */;
/*!40000 ALTER TABLE `modulos` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `tarefas`
--

DROP TABLE IF EXISTS `tarefas`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `tarefas` (
  `idtarefa` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(50) COLLATE latin1_general_ci NOT NULL,
  PRIMARY KEY (`idtarefa`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `tarefas`
--

LOCK TABLES `tarefas` WRITE;
/*!40000 ALTER TABLE `tarefas` DISABLE KEYS */;
/*!40000 ALTER TABLE `tarefas` ENABLE KEYS */;
UNLOCK TABLES;

```

```
--
-- Table structure for table `usuarios`
--

DROP TABLE IF EXISTS `usuarios`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `usuarios` (
  `usuario` varchar(20) COLLATE latin1_general_ci NOT NULL,
  `senha` varchar(100) COLLATE latin1_general_ci NOT NULL,
  `nome` varchar(50) COLLATE latin1_general_ci DEFAULT NULL,
  PRIMARY KEY (`usuario`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `usuarios`
--

LOCK TABLES `usuarios` WRITE;
/*!40000 ALTER TABLE `usuarios` DISABLE KEYS */;
INSERT INTO `usuarios` VALUES ('pi','pi','Home');
/*!40000 ALTER TABLE `usuarios` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2013-11-17  4:44:04
```

Arquivo css/site.css

```
body{
    background: #000 url(../images/bg.jpg) no-repeat center bottom fixed;
        -webkit-background-size: cover;
        -moz-background-size: cover;
        -o-background-size: cover;
        background-size: cover;
    font-family: "Segoe UI", "Helvetica", "Arial", San-serif;
}

.hidden{
    position:absolute;
    top:-100px;
    left:-100px;
}

.top{
    background: rgba(255,255,255,0.50) url(../images/logo.png) no-repeat ;
    border: solid 2px #FFF;
    border-radius: 10px;
    margin: 50px auto;

    padding: 10px;
    padding-left: 100px;
    width:874px;
```

```

    -moz-box-shadow: 5px 5px 2px #000;
    -webkit-box-shadow: 5px 5px 2px #000;
    box-shadow: 5px 5px 2px #000;
}

.top .sub{
    font-size: 12px;
}

.top .right{
    float:right;
}

.top .right .auth{
    padding:5px;
    padding-right: 10px;
    margin-top:-30px;
    text-align:right;
    font-size:12px;
}

.top .right .auth span{
    letter-spacing:5px;
    font-weight:bold;
}

.top .right .menu{
    background:#FFF;
    border-radius: 10px;
    padding: 10px;
    margin-top:5px;
}

.top .right .menu a{
    font-size: 14px;
    font-weight:bold;
    color: #333;
    text-decoration: none;
}

.top .right .menu a:hover{
    color: #999;
}

.top .right .menu .active{
    color: #999;
}

.center{
    margin: 0px auto;
    width:1000px;
}

.center .left{

    width:700px;
    float:left;
    text-align:center;
}

.center .left .box0{
    display: block;

```

```

        width: 710px;
        text-align: left;
        margin-left: -13px;
        padding:10px;
    }

    .center .right .box0{
        display: block;
        text-align: left;
        margin-left: -13px;;
    }

    .box0 .box{

        display: inline-block;
        background: rgba(255,255,255,0.50);
        border: solid 2px #FFF;

        border-radius: 5px;
        width: 312px;
        height: 150px;

        margin-left:15px;
        margin-bottom: 25px;
        padding:10px;

        -moz-box-shadow: 5px 5px 2px #000;
        -webkit-box-shadow: 5px 5px 2px #000;
        box-shadow: 5px 5px 2px #000;
    }

    .auth0{
        margin: 10px auto;
        background: rgba(255,255,255,0.50);
        border: solid 2px #FFF;

        border-radius: 5px;
        width: 312px;
        height: 150px;
        padding:10px;

        -moz-box-shadow: 5px 5px 2px #000;
        -webkit-box-shadow: 5px 5px 2px #000;
        box-shadow: 5px 5px 2px #000;
    }

    .auth0 table{
        margin: 20px auto;
    }

    .box0 .reta{
        display: inline-block;
        background: rgba(255,255,255,0.50);
        border: solid 2px #FFF;

        border-radius: 5px;
        width: 670px;
        height: 100px;

        margin-left:15px;
        margin-bottom: 25px;
        padding:10px;

        -moz-box-shadow: 5px 5px 2px #000;
        -webkit-box-shadow: 5px 5px 2px #000;
    }

```

```

        box-shadow: 5px 5px 2px #000;
    }

    .right .box0 .box{
        margin-left:45px;
        margin-top:10px;
        width: 230px;
        height: 350px;
    }

    .text-tile{
        margin-top: -20px;
        font-size:12px;
        font-weight:bold;
        background: rgba(255,255,255,1);
        border: solid 2px #FFF;
        border-radius: 5px;
        padding: 5px;
        width:100px;
    }

    .Icon{
        background-position: top center;
        border-radius: 5px;
        border: solid 1px #CCC;
        width: 60px;
        height: 60px;
        opacity: 0.9;
        -moz-box-shadow: 1px 1px 1px #000;
        -webkit-box-shadow: 1px 1px 1px #00;
        box-shadow: 1px 1px 1px #000;
        margin: 5px;
    }

    .Icon:hover{
        opacity: 1;
    }

    .IconRGB{
        background: #CCC url('../images/icon-light.png');
    }

    .IconSRM{
        background: #CCC url('../images/icon-windows.png');
    }

    .IconLOCK{
        background: #CCC url('../images/icon-unlock.png')
    }

    .IconFAN{
        background: #CCC url('../images/icon-fan.png');
    }

    .IconSHIELD{
        background: #CCC url('../images/icon-shield.png');
    }

    .IconCLOUD{
        background: #CCC url('../images/icon-cloud.png');
    }

    .IconFIRE{
        background: #CCC url('../images/icon-fire.png');
    }

```

```

    }

    .IconMEGA{

        background: #CCC url('../images/icon-megaphone.png');

    }

    .IconWIFI{

        background: #CCC url('../images/icon-wifi.png');

    }

    .text{
        margin-left:-5px;
        margin-top:10px;
        padding:10px;
    }

    #NavHelp{
        position:absolute;
        left: 100px;
        width:100px;
        height:100px;

        background: rgba(255,255,255,0.90);
        border: solid 2px #FFF;
        border-radius: 10px;
        margin: 50px auto;
        padding: 10px;
        padding-left: 100px;

        -moz-box-shadow: 5px 5px 2px #000;
        -webkit-box-shadow: 5px 5px 2px #000;
        box-shadow: 5px 5px 2px #000;
        visibility:hidden;
        z-index:100;
    }

    #NavHelp .Mark{

        position:absolute;
        width: 1px;
        height: 1px;
        border: solid 20px transparent;
        border-bottom: solid 20px #FFF;
        border-top: none;
        margin-top:-33px;
        margin-left:-80px;
    }

```

Arquivo scripts/__init__.py

```
# EM BRANCO
```

Arquivo scripts/CommandSerial.py

```
python
# importa a classe serial instalada anteriormente, e a classe time nativa do
import serial, time

# Cria a class CommandSerial
class CommandSerial:

    #Cria o metodo Send que envia o comando para o arduino
    def SendByPort(self,cmd,port):
```

```

        try:
            ser = serial.Serial(port, 9600)
            # Seta o timeout para dois minutos aguardando resposta.
            ser.timeout=1
            # Envia o comando para o arduino
            ser.write(cmd)
            # Inicia variavel ret que recebera a resposta do arduino
            #espera 700 millesegundos pela resposta.
            time.sleep(0.7)
            # pega resposta do Arudino
            ret = ser.readline()
            ser.close()
            # retorna resposta do arduino.
            return ret
        except Exception:
            return "ERROR"
    def Send(self,cmd):
        return this.SendByPort(cmd,'/dev/ttyACM0')

```

Arquivo scripts/DataBase.py

```

# importa biblioteca do Mysql
import MySQLdb

# Cria a class DataBase
class DataBase:

    # Cria o metodo dbOne retorna a primeira linha na tabela
    def dbOne(self,sql):
        # Conecta no banco
        con = MySQLdb.connect('127.0.0.1','root','kirk')
        # Seleciona banco de dados
        con.select_db('pi')
        # Abre cursor
        cursor = con.cursor()
        # Execulta query
        cursor.execute(sql)
        # Retorna resultado do cursor
        return cursor.fetchone()

    #Cria o metodo dbAll retorna todas as linhas da tabela
    def dbAll(self,sql):
        # Conecta no banco
        con = MySQLdb.connect('127.0.0.1','root','kirk')
        # Seleciona banco de dados
        con.select_db('pi')
        # Abre cursor
        cursor = con.cursor()
        # Execulta query
        cursor.execute(sql)
        # Retorna resultado do cursor
        return cursor.dictfetchall()

    def dbExe(self,sql):
        # Conecta no banco
        con = MySQLdb.connect('127.0.0.1','root','kirk')
        # Seleciona banco de dados
        con.select_db('pi')
        # Abre cursor
        cursor = con.cursor()
        # Execulta query
        cursor.execute(sql)
        # confirma operação
        con.commit()

```


Arquivo scripts/filas.py

```
#!/usr/bin/python

import datetime

from DataBase import *
from CommandSerial import *
from time import gmtime, strftime

def log(str):
    print strftime("%Y-%m-%d %H:%M:%S", gmtime()) + "\t" + str

while True:
    SQL = "SELECT * FROM filas order by data asc"
    try:
        Data = DataBase()
        ret = Data.dbAll(SQL)

        for row in ret:
            try:
                ser = CommandSerial()
                try:
                    resp = ser.SendByPort(row[3], row[5])
                except ValueError:
                    resp = "error"

                SQLinsert = "insert into logs\n"
                (id, idtarefa, data, usuario, texto) values (null, 0, NOW(), '" + row[1] + "', '" + row[3] + " para\n"
                "+row[5] + " RESP: " + resp + " ');"
                SQLdelete = "delete from filas where\n"
                id = " + str(row[0]) + ";"

                Data.dbExe(SQLinsert)
                Data.dbExe(SQLdelete)

                LOG = "ID: " + str(row[0]) + "\t" + row[1] + "\t" +
                row[3] + " para " + row[5] + " RESP: " + resp

                log(LOG)
            except Exception:
                print "Erro no envio de comando"

        except Exception:
            ret = "error"
```

Arquivo scripts/filas_startup.sh

```
#!/bin/bash
LOG=`date +%Y%m%d`
DIR="./"
./filas.py >> fila-$LOG.log &
```

Arquivo scripts/functions.js

```
$(document).ready(function() {

    $("#RGB0").dblclick(function(){RGB0()});
    $("#RGB1").dblclick(function(){RGB1()});
    $("#RGB2").dblclick(function(){RGB2()});
    $("#RGB3").dblclick(function(){RGB3()});
```

```

$("#RGB0").click(function(){RGBX('#FFFFFF','0')});
$("#RGB1").click(function(){RGBX('#FFFFFF','1')});
$("#RGB2").click(function(){RGBX('#FFFFFF','2')});
$("#RGB3").click(function(){RGBX('#FFFFFF','3')});

$("#SRM0").click(function(){SRM0()});

$(".Icon").mouseout(
    function(event){
        $("#NavHelp").css("visibility","hidden");
    }
);

$(".Icon").mouseover(
    function(event){
        $("#NavHelp").css("visibility","hidden");
        var v = function(){

            //var par = "";
            //$.getJSON( par, function( data ) {
            //
            $("#NavHelp").css("visibility","visible");
            //      $("#NavHelp").css("left",(event.pageX-
35)+"px");
            //      $("#NavHelp").css("top",(event.pageY-
50)+"px");
            //      $("#NavHelpText").html("TESTE ");
            //});

        }
        setTimeout(v,3000);
    }
);

function LEDON(OBJ){

    sendAction(OBJ,"FFF");
    $("#"+OBJ).css("background-color","#FFF");

}

function SRM0(){

    var img = $("#SRM0").css("background-image");
    var parameter="180";

    if(img.indexOf("-open")>0){
        parameter="001";
        img = "url(images/icon-windows.png)";
    }else{
        img = "url(images/icon-windows-open.png)";
    }

    var par = "scripts/json.py?cmd=SRM0"+parameter+ ".";
    $.getJSON( par, function( data ) {
        console.log(data.cmd)
        $("#SRM0").css("background-image",img);
    });

}

function RGB0(){

```

```

        $("#favcolor0")[0].click();
        $("#favcolor0").on("change",function(){
            var cor = $("#favcolor0").val();
            sendAction("LED0",RGBHexDec(cor));
            $("#RGB0").css("background-color",cor);
            return;
        }
    );
}

function RGB1(){
    $("#favcolor1")[0].click();
    $("#favcolor1").on("change",function(){
        var cor = $("#favcolor1").val();
        sendAction("LED1",RGBHexDec(cor));
        $("#RGB1").css("background-color",cor);
    }
);
}

function RGB2(){
    $("#favcolor2")[0].click();
    $("#favcolor2").on("change",function(){
        var cor = $("#favcolor2").val();
        sendAction("LED2",RGBHexDec(cor));
        $("#RGB2").css("background-color",cor);
    }
);
}

function RGB3(){
    $("#favcolor3")[0].click();
    $("#favcolor3").on("change",function(){
        var cor = $("#favcolor3").val();
        sendAction("LED3",cor);
        $("#RGB3").css("background-
color",RGBHexDec(cor));
    }
);
}

function RGBX(cor,obj){
    var excor = $("#RGB"+obj).css("background-color")

    if(excor.indexOf("255, 255, 255") >-1)excor="#000000";
    else excor = cor;

    var dcor = RGBHexDec(excor);
    $("#RGB"+obj).css("background-color",excor);
    sendAction("LED"+obj,dcor);
}

function RGBHexDec(parmeter){
    parmeter=parmeter.replace("#","");

    var v1 = parmeter.substring(0,2);
    var v2 = parmeter.substring(2,4);

```

```

        var v3 = parameter.substring(4,6);

        var vi1= parseInt(v1,16);
        var vi2= parseInt(v2,16);
        var vi3= parseInt(v3,16);

        if(parseInt(vi1)<10)vi1="0"+vi1;
        if(parseInt(vi1)<100)vi1="0"+vi1;

        if(parseInt(vi2)<10)vi2="0"+vi2;
        if(parseInt(vi2)<100)vi2="0"+vi2;

        if(parseInt(vi3)<10)vi3="0"+vi3;
        if(parseInt(vi3)<100)vi3="0"+vi3;

        parameter = vi1+" "+vi2+" "+vi3;
        //alert(vi1+" - "+vi2+" - "+vi3)
        return parameter;
    }

    function sendAction(cmd, parameter){

        var par = "scripts/json.py?cmd="+cmd+parameter+ ".";

        $.getJSON( par, function( data ) {
            console.log(data.cmd)
        });

        return 0;
    }

```

Archivo scripts/json.py

```

#!/usr/bin/python

import cgi, cgitb, serial

from DataBase import *

print "Content-type: application/json"
print
try:

    form = cgi.FieldStorage()
    if not form.has_key("cmd") : ret = "Not Command"
    else: ret = form.getvalue("cmd");

    if not ret == "Not Command":
        conn = DataBase()
        ret = conn.dbExe("insert into filas
(id,usuario,data,comando,tipo,porta)
(null,'INTERFACE',NOW(),'+ret+', 'IN', '/dev/ttyACM0');")

except ValueError:
    ret = "error";

print "{ \"cmd\": \""+ ret+"\" }"

```

Arquivo home.py

```
#!/usr/bin/env python

import cgi
import cgitb
import Cookie
import datetime
import random
from scripts.DataBase import *

#Habilita o CGI
cgitb.enable()

# Request do dados do formulario
form = cgi.FieldStorage()

Data = DataBase()

usuario=""
senha=""
if "usu" in form or "pass" in form:
    usuario = form["usu"].value
    senha = form["pass"].value

SQL = "select * From usuarios where usuario = '"+ usuario +"'" and
senha = '"+senha+"'"
ret = Data.dbOne(SQL);
if not ret :
    print "location: index.py?erro=invalido"

else:
    print "location: index.py?erro=falha"

print "Content-type: text/html"
print

print "<!DOCTYPE HTML>"
print "<html lang='pt-br'"

    <head>
        <meta charset="UTF-8">
        <title>Home</title>
        <link href="css/site.css" rel="stylesheet" type="text/css">
        <link rel="shortcut icon" href="images/logo.png" type="image/x-icon"/>
        <link href="scripts/css/ui-lightness/jquery-ui-1.10.3.custom.css"
rel="stylesheet" type="text/css">
        <script src="scripts/js/jquery-1.9.1.js"></script>
        <script src="scripts/functions.js"></script>
    </head>
    <body>
        <div class="top">

            <b>HOME</b><br>
            <span class="sub">Domtioc System. </span>
            <div class="right">
                <div class="auth">
                    <span>Bem vindo!</span><br/>
                    Alexandre Biondi Ermetti!!! <a href="#">Sair</a>
```

```

        </div>
        <div class="menu">
             <a href="#"
class="active">Home</a> |
             <a
href="#">Sobre</a>
        </div>
    </div>
</div>
<div class="hidden">
    <input type="color" id="favcolor0" >
    <input type="color" id="favcolor1" >
    <input type="color" id="favcolor2" >
    <input type="color" id="favcolor3" >
    <input type="color" id="favcolorX" >
</div>
<div id="NavHelp">
    <div class="Mark"></div>
    <div id="NavHelpText" class="text"></div>
</div>
<div class="center">
    <div class="left">
        <div class="box0">
            <div class="box">
                <div class="text-tile">Dormitorio 1</div>
                <div class='text'>
                    <button id="RGB0" class="Icon
IconRGB"></button>
                    <button id="SRM0" class="Icon
IconSRM"></button>
                </div>
            </div>
            <div class="box">
                <div class="text-tile">Dormitorio 2</div>
                <div class='text'>
                    <button id="RGB1" class="Icon
IconRGB"></button>
                    <button id="FAN0" class="Icon
IconFAN"></button>
                </div>
            </div>
            <div class="box">
                <div class="text-tile">Cozinha</div>
                <div class='text'>
                    <button id="RGB2" class="Icon
IconRGB"></button>
                    <button id="FAN1" class="Icon
IconFAN"></button>
                </div>
            </div>
            <div class="box">
                <div class="text-tile">Sala</div>
                <div class='text'>
                    <button id="RGB3" class="Icon
IconRGB"></button>
                </div>
            </div>
        </div>
    </div>
    <div class="right">
        <div class="box0">
            <div class="box">
                <div class="text-tile">Ult.
Eventos</div>

```

```

        </div>
    </div>
</div>

</body>
</html>""""

```

Arquivo index.py

```

#!/usr/bin/env python
import cgi

print "Content-type: text/html"
print

print "<!DOCTYPE HTML>"
print """"<html lang="pt-br">

    <head>
        <meta charset="UTF-8">
        <title>Home</title>
        <link href="css/site.css" rel="stylesheet" type="text/css">
        <link rel="shortcut icon" href="images/logo.png" type="image/x-icon"/>
        <link href="scripts/css/ui-lightness/jquery-ui-1.10.3.custom.css"
rel="stylesheet" type="text/css">
        <script src="scripts/js/jquery-1.9.1.js"></script>
        <script src="scripts/functions.js"></script>
    </head>
    <body>
        <div class="top">

            <b>HOME</b><br>
            <span class="sub">Domtioc System. </span>

        </div>
        <div class="center">
            <form action="home.py" method="POST">
                <div class="box0">
                    <div class="auth0">
                        <div class="text-tile">Login </div>
                        <table>
                            <tr>
                                <td><b>Usuario:</b></td>
                                <td><input type="text" id="usu" name="usu" /></td>

                            </tr>
                            <tr>
                                <td><b>senha:</b></td>
                                <td><input type="password" id="pass"
name="pass" /></td>

                            </tr>
                            <tr>
                                <td colspan="2">
                                    <input type="submit" value="Entrar" />
                                </td>
                            </tr>
                        </table>
                    </div>
                </div>
            </form>
        </div>

```

```

        </body>
    </html>"""

Arquivo tarefas.py
#!/usr/bin/python

import datetime

from DataBase import *
from CommandSerial import *
from time import gmtime, strftime

def getValue(comando,lista):
    for row in lista:
        var = row.split(":")
        if var[0] == comando:
            return int(var[1])
    return 0

def log(str):
    print strftime("%Y-%m-%d %H:%M:%S", gmtime()) + "\t" + str

def operacao(valor,operador,comando,entrada,porta,idtarefa):
    #print str(valor)+"|"+operador+"|"+comando+"|"+str(entrada)+"|"+porta

    db = DataBase()
    SQL="insert into filas (id,usuario,data,comando,tipo,porta) values
(null,'SYS',NOW(), '"+comando+"', 'IN', '"+porta+"');"
    SQL1="UPDATE horarios SET espera=ADDDATE(NOW(), interval repetir
second) where idtarefa="+idtarefa

    LOG="CMD: "+comando+" PORT:"+porta

    if operador=='=':
        if(int(valor)==entrada):

            db.dbExe(SQL1)
            db.dbExe(SQL)
            log(LOG)

    elif operador=='>':
        if(int(valor)>entrada):

            db.dbExe(SQL1)

            db.dbExe(SQL)
            log(LOG)

    elif operador=='<':
        if(int(valor)>entrada):

            db.dbExe(SQL1)
            db.dbExe(SQL)
            log(LOG)

    elif operador=='<>':
        if(int(valor)>entrada):

            db.dbExe(SQL1)
            db.dbExe(SQL)
            log(LOG)

    while True:

        SQL = """select h.idtarefa,(select comando from modulos where
id=idmodulo) from tarefas as t inner join horarios as h on h.idtarefa = t.idtarefa where

```



```

        (convert(datainicial,date) < now() or convert(datainicial,date)
='0000-00-00') and
        (convert(datafinal,date) > now() or convert(datafinal,date) ='0000-
00-00') and
        (convert(substring(datainicial,11,6),time) < CURTIME() or
convert(datainicial,time) ='00:00') and
        (convert(substring(datafinal,11,6),time) > CURTIME() or
convert(datafinal,time) ='00:00')
        and espera < NOW()
        """

        data = DataBase()
        resp = data.dbAll(SQL)

        ser = CommandSerial()
        try:
            comando
            =
ser.SendByPort("STUS.", '/dev/ttyACM0').splitlines()

        except ValueError:
            comando = """TES0: 100
TES1: 200
TES3: 500
""".splitlines()

        for row in resp:
            SQLCondicoes = "SELECT * FROM condicoes as c inner join modulos
as m on c.atuador = m.idmodulo where idtarefas = "+str(row[0])
            #print SQLCondicoes
            cond = data.dbAll(SQLCondicoes)
            for rowcond in cond:
                var=getValue(row[1],comando)

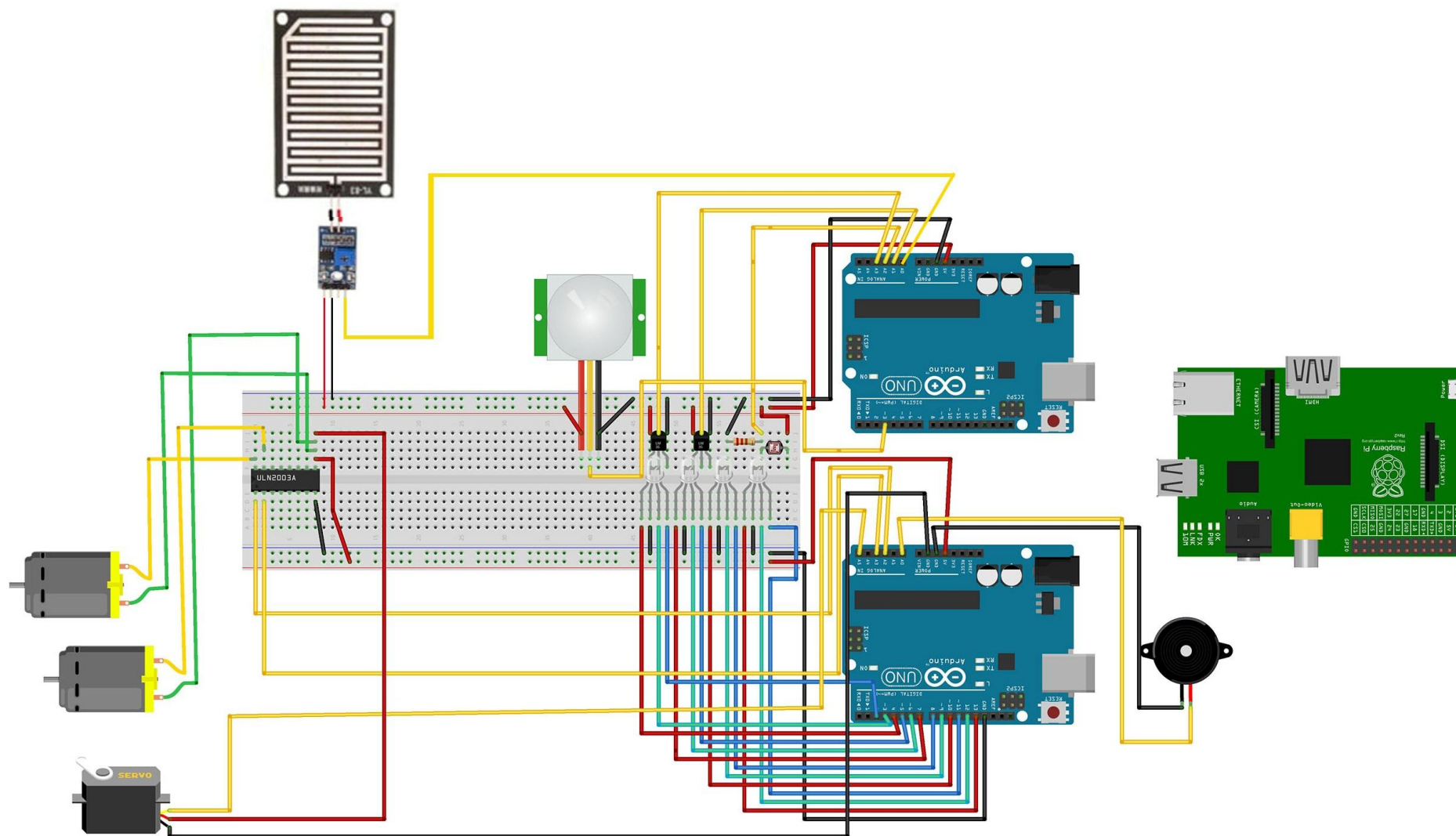
operacao(rowcond[1],rowcond[2],rowcond[4],var,rowcond[8],str(row[0]))

Arquivo tarefas_startup.sh

#!/bin/bash
LOG=`date +%Y%m%d`
DIR="./"
./tarefas.py >> tarefas-$LOG.log &

```

APÊNDICE D – CIRCUITO COMPLETO



APÊNDICE E – LISTA DE PREÇOS PARA CONSTRUÇÃO DO PROTOTIPO

Qtd.	Componente	Valor
2x	Arduinos Uno R3	R\$ 140.00
1x	Raspberry PI B	R\$ 150.00
1x	FAN 5x5 cm	R\$ 7.00
1x	FAN 12x12 cm	R\$ 14.00
1x	Buzzer com Oscilador	R\$ 3.00
1x	Cartão Micro SDHC c/ Adapt. 4Gb	R\$ 15.00
1x	Cabo USB to Micro USB Sony	R\$ 30.00
1x	Fonte de Alimentação 5V	R\$ 10.00
1x	Servo Motor Hextronik Hxt900 9g	R\$ 14.00
4x	LEDs RGB	R\$ 7.20
1x	Sensor de movimento PIR	R\$ 36.00
1x	Sensor de Luminosidade LDR	R\$ 0.80
2x	Sensores de Temperatura	R\$ 4.40
1x	Sensor de Chuva	R\$ 32.00
1x	CI UNL 2003A	R\$10.00
1x	Capacitor 18k	R\$ 2.00
5x	Metros de fio	R\$ 4.00
1x	Protoboard 600 Pinos	R\$ 19.00
		TOTAL: R\$ 498.40