UNIVERSIDADE FEEVALE

MARCO MARCELO MOREIRA

ESTUDO TEÓRICO E PRÁTICO SOBRE O USO DE SISTEMAS EMBARCADOS E INTERNET DAS COISAS EM SEGURANÇA DOMICILIAR

Novo Hamburgo

2017

MARCO MARCELO MOREIRA

ESTUDO TEÓRICO E PRÁTICO SOBRE O USO DE SISTEMAS EMBARCADOS E INTERNET DAS COISAS EM SEGURANÇA DOMICILIAR

Trabalho de conclusão de curso apresentado como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação pelo Centro Universitário Feevale

Orientador: Vandersilvio da Silva

Novo Hamburgo

2017

AGRADECIMENTOS

Agradeço...

... a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial à minha família que ao meu lado me apoiaram de todas as formas, apesar das dificuldades.

... aos amigos e colegas do Centro Universitário Feevale, aos professores, e em especial, ao meu orientador Prof. Me. Vandersilvio da Silva, pela sua disponibilidade e dedicação que me permitiram concluir este desafio.

... a meus filhos que me nortearam nessa jornada.

... a minha esposa, pela compreensão irrestrita.

RESUMO

Os avanços da microeletrônica permitiram o uso indiscriminado de objetos com tecnologia embarcada, comunicáveis entre si. A isso deu-se o nome de Internet das Coisas (*Internet of Things - IoT*), que possibilita acesso remoto nos mais diversos ambientes e funcionalidades. Este trabalho aborda o uso da *IoT*, para atuação remota em sistemas de segurança domiciliar. Foi desenvolvido um protótipo para aferir a viabilidade de implementação de um sistema de segurança domiciliar, utilizando sistemas embarcados e Internet das Coisas. A aplicação implementada monitora o ambiente e quando detecta movimentos e/ou vazamento de gás, notifica os usuários por uma interface desenvolvida e com o envio de e-mail. Após a realização de testes, foi possível constatar que os objetivos propostos foram alcançados, visto que o protótipo capturou todos os eventos simulados nos sensores, mantendo um histórico e dando ciência para os dispositivos dos usuários.

Palavras-chave: Sistemas embarcados; Internet das Coisas; Raspberry Pi; Segurança residencial.

ABSTRACT

The advances in microelectronics allowed the indiscriminate use of objects with embedded technology, communicable to each other. This was called Internet of Things (IoT), which allows remote access in the most diverse environments and functionalities. This work addresses the use of IoT, for remote performance in home security systems. A prototype was developed to measure the feasibility of implementing a home security system using Embedded Systems and Internet of Things. The implemented application monitors the environment and when it detects movements and / or gas leaks, it notifies the users through a developed interface and the sending of e-mail. After the tests, it was possible to verify that the proposed objectives were reached, since the prototype captured all the simulated events in the sensors, maintaining a history and giving science to the devices of the users.

Keywords: Embedded systems; Internet of Things; Raspberry Pi; Residential security.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ARM	*Advanced RISC Machine*
BCM	*Broadcom*
CPU	*Central Processing Unit*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
FGPA	*Field Programmable Gate Array*
FTP	*File Transfer Protocol*
GLP	Gás Liquefeito de Petróleo
GND	*Ground*
GPIO	*General Purpose Input/Output*
HDMi	*High-Definition Multimedia Interface*
HTTP	*Hypertext Transfer Protocol*
I2C	*Inter-Integrated Circuit*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IoT	*Internet of Things* - Internet das Coisas
IPV6	*Internet Protocol Version 6*
MQTT	*Message Queuing Telemetry Transport*
NFC	*Near Field Communication*
NPM	*Node Package Manager*
OSI	*Open Systems Interconnection*
PC	*Personal Computer*
PIR	*Passive Infrared*
QoS	*Quality of Service*
RAM	*Random Access Memory*
RCA	*Radio Corporation of America*
RFID	*Radio Frequency Identification*

ROM	*Read Only Memory*
Rpi	Raspberry Pi
SE	Sistema Embarcado
SFTP	*Secure File Transfer Protocol*
SoC	*System on a Chip*
SPI	*Serial Peripheral Interface*
UART	*Universal Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
XP	*Extreme Programming*

SUMÁRIO

INTRODUÇÃO

Atualmente, é factível monitorar sua residência mesmo estando a quilômetros de distância, pois há a possibilidade de uma central de alarme avisá-lo por celular que o alarme disparou. Caso não tenha deixado a chave da casa com alguém de confiança, para verificar o que aconteceu, poderá sentir-se inclinado a deslocar-se até lá. Difícil acreditar que, com a facilidade de comunicação e a rapidez na transmissão da informação, ainda haja necessidade de deslocamento físico para resolução de alarme falso em sua residência.

A nova era da informação conduzida pela Internet das Coisas traz consigo recursos novos e interessantes como processos de controle das coisas. Além disso, o usuário não se preocupa com os processos de desenvolvimento em si, apenas com os resultados. Com isso é possível obter respostas rápidas para que se tenha maior segurança e acesso à informação instantaneamente. Assim, o desenvolvimento de novas tecnologias permite a utilização de dispositivos inteligentes que trazem muitas vantagens para a vida cotidiana. Os dispositivos inteligentes comandam o meio ambiente que nos rodeia, livrando as pessoas de resolverem tarefas comuns e do desperdício de tempo (INTERNET OF THINGS IN 2020, 2008).

A Internet das Coisas (do inglês *Internet of Things*) – *IoT*, surgiu com os avanços de várias áreas como sistemas embarcados, microeletrônica, comunicação e tecnologia de informação. A *IoT* é um paradigma que preconiza um mundo de objetos físicos embarcados com sensores conectados por redes sem fio e que se comunicam, usando a Internet, moldando uma rede de objetos inteligentes capazes de realizar variados processamentos, capturar variáveis ambientais e reagir a estímulos externos (ATZORI et al., 2010).

A partir de 2005, a discussão sobre a Internet das Coisas se generalizou, começou a ganhar a atenção dos governos e aparecer relacionada a questões de privacidade e segurança de dados. Naquele ano a Internet das Coisas se tornou a pauta do *International Telecommunication Union* (ITU), agência das Nações Unidas para as tecnologias da informação e da comunicação, que publica anualmente um relatório sobre tecnologias emergentes. Assim, depois da banda larga e da internet móvel, a Internet das Coisas ganhou a atenção do órgão e passou a figurar como o próximo passo das tecnologias ‘*always on*’ [...] que prometem um mundo de dispositivos interconectados em rede (ITU, 2005).

Os Sistemas Embarcados encontram-se cada vez mais presentes em nosso dia-a-dia. Um componente embarcado é um dispositivo instalado em um produto, geralmente destinado à realização de um conjunto de tarefas predefinidas com requisitos específicos. São formados

pelas partes de hardware e software intimamente relacionadas. O hardware é a parte física do componente e o software é a parte computacional, ou seja, a parte lógica.

Segundo Spinola (1998), podemos considerar também que os hardwares, em geral, são módulos genéricos, enquanto que os softwares implementam os requisitos mais específicos.

Esse cenário avança com o início da Internet das Coisas (*IoT*), que se refere à interligação em rede de objetos do cotidiano conectados por meio de sistemas embarcados que utilizam a internet e se comunicam com os seres humanos e outros dispositivos (XIA *et al*., 2012).

Todo sistema embarcado é composto por uma unidade de processamento, que é um circuito integrado, fixado a uma placa de circuito impresso. Possuem uma capacidade de processamento de informações vinda de um software que está sendo processado internamente nessa unidade. Logo, o software está embarcado na unidade de processamento. Todo o software embarcado é classificado de firmware (BALL, 2005).

Microcomputadores têm sido assunto na categoria de inovações nos últimos tempos e o Raspberry Pi, simboliza mais um passo nessa evolução computacional e representa um esforço por parte de educadores para expandir o conhecimento de programação e eletrônica para pessoas de todas as idades.

O Raspberry Pi, é uma placa de desenvolvimento, com o tamanho similar a de um cartão de crédito, produzida no Reino Unido pela fundação de mesmo nome. Foi criada na Universidade de Cambridge pelo engenheiro britânico Eben Upton e sua equipe, com o objetivo principal de incentivar o ensino de programação crianças e por meio de um preço acessível.

A primeira versão surgiu em 2006 e, atualmente, possui diversos modelos disponíveis no mercado. Podem ter diferentes configurações com suporte a muitas linguagens de programação como Python, Java, Ruby on Rails, Perl e C++ (RASPBERRY,2016).

Este estudo se insere no conceito Internet das Coisas por meio do desenvolvimento de um protótipo de segurança residencial, composto por uma plataforma Raspberry Pi, controlando sensores conectados a ela, que possa ser gerenciado pela internet por meio de um sistema embarcado no dispositivo, que o usuário poderá controlar remotamente.

A organização deste trabalho é apresentada inicialmente com esta introdução e mais três capítulos. No próximo capítulo será apresentado o levantamento do referencial teórico, que percorre pelo surgimento da Internet das Coisas, passando por seus avanços e utilizações em

várias áreas da tecnologia da informação, assim como as plataformas embarcadas e aplicações desenvolvidas para esse fim. No segundo, será abordada a implementação, demonstrando os elementos utilizados no desenvolvimento do protótipo, no qual também são expostas a operacionalidade da aplicação e análise dos resultados. Por fim, são relatadas as conclusões e possíveis extensões.

1 REFERENCIAL TEÓRICO

Este capítulo está organizado em oito seções, tratando do embasamento teórico para melhor entendimento deste trabalho de conclusão de curso.

1.1 INTERNET DAS COISAS

No final dos anos de 1980, o filósofo Pierre Lévy, denomina de "ecologia cognitiva" a articulação de "coletividades pensantes homens-coisas". Recentemente, essa ideia deixou o campo da especulação filosófica para inspirar um relatório preparado para o *World Summit on the Information Society*, relatório que ganhou o sugestivo nome de "Internet das Coisas" (DINIZ, 2006).

A ideia por trás da *IoT* nasce de uma nova dimensão de conexão propiciada pela Internet – além de possibilitar a comunicação em qualquer tempo e em qualquer lugar, agora também considera a comunicação de qualquer coisa (DINIZ, 2006).

A incorporação de capacidades de processamento, comunicação e sensoriamento a objetos cotidianos vêm permitindo, entre outras possibilidades, a integração desses objetos (ditos objetos inteligentes) às redes de computadores. Essa perspectiva abre espaço para uma ampla integração do mundo físico ao digital, o que pode ser alcançado por meio da interação autônoma entre esses objetos – que serão maciçamente embarcados nos ambientes. Essa integração constitui a ideia central da *IoT* e deve possibilitar o desenvolvimento de uma série de aplicações caracterizadas por sua elevada mobilidade, adaptação e distribuição (XAVIER et al., s.d. apud PIREZ et al., 2015).

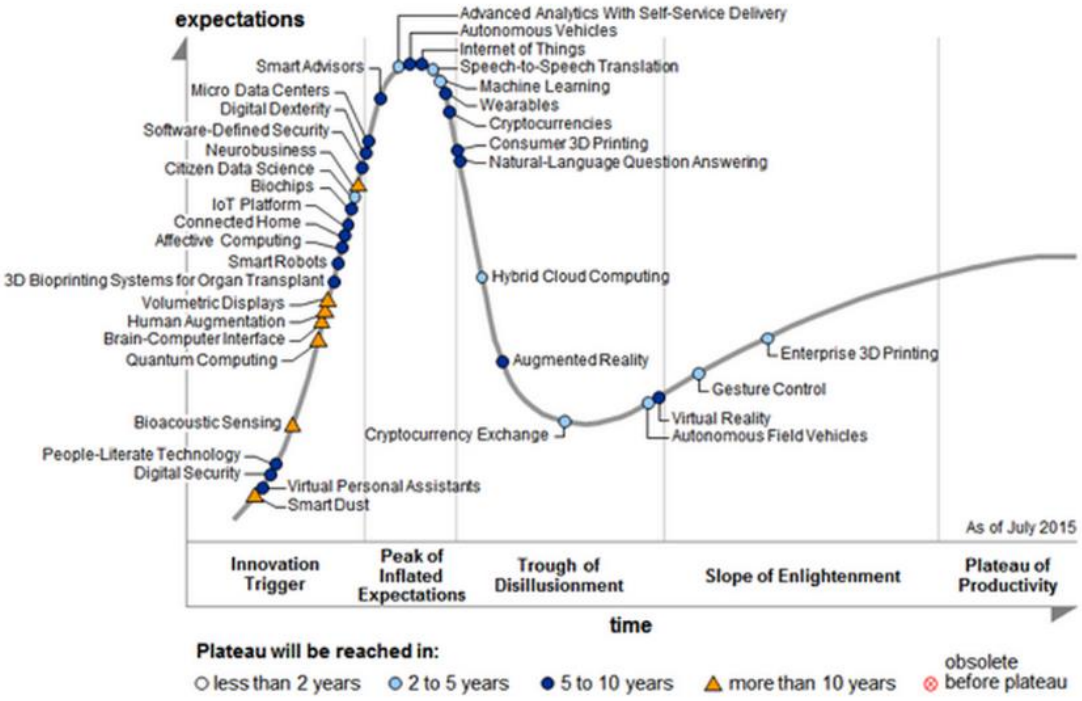
A *IoT*, em poucas palavras, nada mais é que uma extensão da Internet atual, que proporciona aos objetos do dia-a-dia (quaisquer que sejam), mas com capacidade computacional e de comunicação, se conectarem à Internet. A conexão com a rede mundial de computadores viabilizará, primeiro, controlar remotamente os objetos e, segundo, permitir que os próprios objetos sejam acessados como provedores de serviços. Essas novas habilidades, dos objetos comuns, geram um grande número de oportunidades tanto no âmbito acadêmico quanto no industrial. Todavia, essas possibilidades apresentam riscos e acarretam amplos desafios técnicos e sociais (SIQUEIRA et al., 2016).

> Existem várias barreiras que ameaçam diminuir o desenvolvimento da *IoT*. Dentre elas, pode-se citar: a transição para o protocolo IPv6, o estabelecimento de um conjunto de padrões e desenvolvimento de fontes de energia para milhões, até mesmo bilhões de sensores minúsculos. Porém, partindo do pressuposto que diversas

> empresas, organizações normativas e instituições acadêmicas trabalhem juntos para solucionar estes problemas, a *IoT* continuará a progredir. (ARANDA, 2016, p.19 apud EVANS, 2011).

A *IoT* foi identificada como uma tecnologia emergente em 2012 por especialistas da área (SIQUEIRA et al, 2016 apud Gartner 2015). A Figura 1 apresenta uma maneira de representar o surgimento, adoção, maturidade e impacto de diversas tecnologias chamada de *Hype Cycle*, ou "Ciclo de Interesse". Em 2012, foi previsto que a *IoT* levaria entre cinco e dez anos para ser adotada pelo mercado e, hoje, é vivenciado o maior pico de expectativas sobre a tecnologia no âmbito acadêmico e industrial. Também se pode notar o surgimento das primeiras plataformas de *IoT* que têm gerado uma grande expectativa de seu uso (SIQUEIRA et al., 2016).

Figura 1 - Tecnologias emergentes



Fonte: SIQUEIRA et al (2016)

Há poucos anos, era possível apontar o fator custo como limitante para adoção e desenvolvimento de objetos inteligentes. Entretanto, hoje, é possível encontrar soluções de *IoT* disponíveis no mercado de baixo custo. Para esse segmento, é possível afirmar que o custo do hardware já é acessível, se analisarmos o preço de produtos como o Raspberry Pi, Arduino e similares que permitem desde a prototipagem até a produção final de soluções de *IoT*. Por exemplo, é possível encontrar o Raspberry Pi ao custo de US$ 35 (SIQUEIRA et al., 2016).

1.2 DOMÓTICA

O termo domótica é originado da junção das palavras *Domus*, que em latim significa casa, e robótica, que representa uma tecnologia capaz de controlar todos os ambientes de uma residência por meio de um só equipamento, incluindo temperatura, luminosidade, som, segurança, dentre outros, ou seja, automação residencial (QUINDERÉ, 2009 apud BOLZANI et al., 2004).

Domótica é um processo ou sistema que prioriza a melhoria do estilo de vida das pessoas, do conforto, da segurança e da economia da residência, usando um controle centralizado das suas funções, como água, luz, telefone e sistemas de segurança, entre outros (QUINDERÉ, 2009 apud ANGEL, 1993 e NUNES, 2002).

Para corresponder às exigências, a domótica faz uso de vários equipamentos distribuídos pela residência de acordo com as necessidades dos moradores. Estes equipamentos podem ser divididos em três principais grupos (QUINDERÉ, 2009 apud TAKIUCHI et al, 2004).

Atuadores: controlam os aparelhos da residência como, por exemplo, luz e ventilador;

Sensores: capturam informações do ambiente como, por exemplo, luminosidade, umidade e presença;

Controladores: são responsáveis pela administração dos atuadores e sensores, ou seja, coordenam todos os aparelhos e equipamentos da residência que fazem parte da automação.

1.3 APLICAÇÕES DE IoT

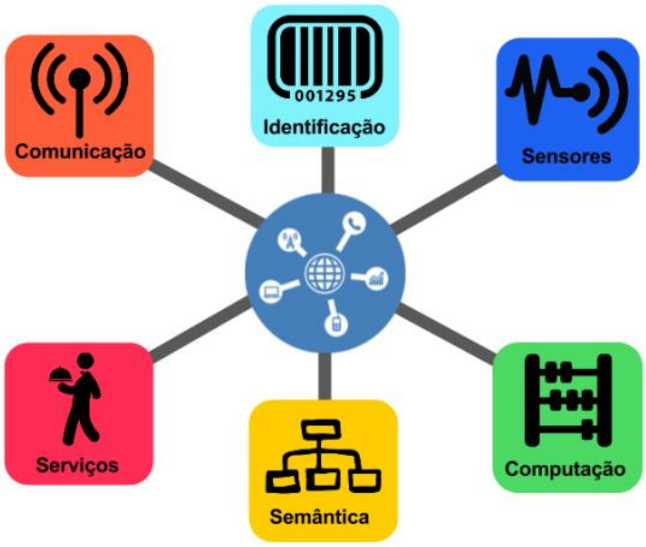
Ao conectar objetos com diferentes recursos a uma rede, potencializa-se o surgimento de novas aplicações. Nesse sentido, conectar esses objetos à Internet significa criar a Internet das Coisas. Na *IoT*, os objetos podem prover comunicação entre usuários e dispositivos. Com isso, emerge uma nova gama de aplicações, tais como coleta de dados de pacientes e monitoramento de idosos, sensoriamento de ambientes de difícil acesso e inóspitos, entre outras. (SUNDMAEKER et al., 2010, tradução nossa).

Conforme Santos et al. (s.d.) os dados providos pelos objetos podem apresentar imperfeições (calibragem do sensor), inconsistências (fora de ordem, *outliers*) e serem de diferentes tipos (gerados por pessoas, sensores físicos, fusão de dados). Assim, as aplicações e algoritmos devem ser capazes de lidar com esses desafios sobre os dados. Outro exemplo diz respeito ao nível de confiança sobre os dados obtidos dos dispositivos da *IoT* e como se pode empregar esses dados em determinados cenários.

Desse modo, os desafios impostos por essas novas aplicações devem ser explorados e soluções devem ser propostas para que a *IoT* contemple as expectativas em um futuro próximo como previsto na Figura 1.

A *IoT* pode ser vista como a combinação de diversas tecnologias, as quais são complementares no sentido de viabilizar a integração dos objetos no ambiente físico ao mundo virtual. A Figura 2 apresenta os blocos básicos de construção da *IoT*.

Figura 2 - Blocos básicos da *IoT*



Fonte: SIQUEIRA et al (2016)

A seguir será explicado em detalhe cada um dos blocos da Figura 2.

Identificação: tecnologias como *RFID*, *NFC* e endereçamento IP podem ser empregados para identificar os objetos.

Sensores/Atuadores: sensores coletam informações sobre o contexto onde os objetos se encontram e, em seguida, armazenam e ou encaminham esses dados para data *warehouse*, *clouds* ou centros de armazenamento. Atuadores podem manipular o ambiente ou reagir de acordo com os dados lidos.

Comunicação: diz respeito às diversas técnicas usadas para conectar objetos inteligentes. Algumas das tecnologias usadas são *WiFi*, *Bluetooth*, IEEE 802.15.4 e *RFID*.

Computação: inclui a unidade de processamento como, por exemplo, microcontroladores, processadores e *FPGAs*, responsáveis por executar algoritmos locais nos objetos inteligentes.

Serviços: a *IoT* pode prover diversas classes de serviços, como por exemplo os Serviços de Ubiquidade que visam prover serviços de colaboração e inteligência em qualquer momento e qualquer lugar em que eles sejam necessários.

Semântica: refere-se à habilidade de extração de conhecimento dos objetos na *IoT*.

1.4 SISTEMAS EMBARCADOS

Os sistemas embarcados estão cada vez mais presentes em nosso dia a dia, mesmo que por vezes passem despercebidos. Aplicações como o controle de um portão eletrônico ou até mesmo o controle de um aparelho de ar condicionado em uma residência podem possuir um Sistema Embarcado (SE).

Segundo dados estimados por pesquisas em alta tecnologia, mais de 90% dos microprocessadores fabricados mundialmente são destinados a máquinas que usualmente não são chamadas de computadores (CHASE E ALMEIDA, 2007 apud REIS, 2004).

Um sistema é classificado como embarcado quando é dedicado a uma única tarefa e interage continuamente com o ambiente a sua volta por meio de sensores e atuadores (CHASE E ALMEIDA, 2007 apud BALL, 2005).

Estamos cercados por sistemas embarcados, alguns estão cada vez mais presentes em nosso dia-a-dia: máquinas de lavar, televisões, eletrodomésticos em geral possuem algum tipo de processamento, automóveis, caixas de banco eletrônicos, equipamentos de comunicação como modens, roteadores, etc.

Conforme Taurion (2005), após o advento do mercado de massa, quando os computadores chegaram às casas dos consumidores finais, vemos a possibilidade da computação onipresente, em que o software estará embarcado em praticamente todos os artefatos fabricados pela sociedade humana, sejam eles automóveis, celulares, TVs, geladeiras, relógios, roupas e assim por diante. A distribuição de um software, nesse cenário não se medirá em milhões de cópias, mas em bilhões. É um cenário onde os padrões abertos potencializam o efeito de rede, e os mecanismos atuais de licenciamento de software proprietário podem ser um grande entrave.

Nesse mercado, o modelo de software livre e o Linux, por excelência, têm um papel de extrema importância. Os sistemas operacionais são o cerne deste cenário, pois em cima deles é que serão construídas as funcionalidades dos artefatos (TAURION, 2005). E quando falamos em sistema operacional para sistemas embarcados, uma das melhores opções para desenvolvimentos de produtos é o Linux.

Linux Embarcado é a aplicação e uso do *kernel* Linux em uma placa eletrônica cujo principal elemento é o SoC (*System on a Chip*). SoC é um chip eletrônico que reúne todos os elementos básicos de um sistema computacional como CPU, memória principal (RAM), memória secundária (ROM/*FLASH*) e alguns periféricos. Atualmente, um SoC contém internamente diversos periféricos dos mais variados tipos como I2C, SPI, UART.

O *kernel* Linux em conjunção com outros softwares são escritos na memória *flash* ou outra mídia de armazenamento presente na placa, como, por exemplo, um cartão microSD. Essa combinação forma um sistema operacional completo e funcional.

O mesmo Linux que roda em um supercomputador pode rodar também em uma simples placa eletrônica. O que torna isso possível é que o Linux suporta uma grande variedade de arquiteturas e processadores. No entanto, nem todas essas arquiteturas são de fato usadas em sistemas embarcados. As mais comumente usadas em sistemas embarcados são ARM, PowerPC e MIPS.

Muitas subcomunidades mantêm seu próprio *kernel*, com atributos novos e distintos, porém menos estáveis. Além disso, em alguns casos, os fabricantes mantêm versões derivadas do Linux oficial para dar suporte para seu hardware específico. Citando alguns exemplos, a *Beaglebone Black* mantém seu *kernel* no endereço https://github.com/beagleboard/linux e a Raspberry Pi, no endereço https://github.com/raspberrypi/linux.

1.5 O RASPBERRY PI

Este capitulo fornece uma visão geral sobre a plataforma Raspberry PI, em que são apresentadas as definições, características técnicas, uma breve comparação entre os modelos 2 e o 3 escolhido para o desenvolvimento do protótipo, interface GPIO (*General Purpose Input/Output*) e suas aplicações.

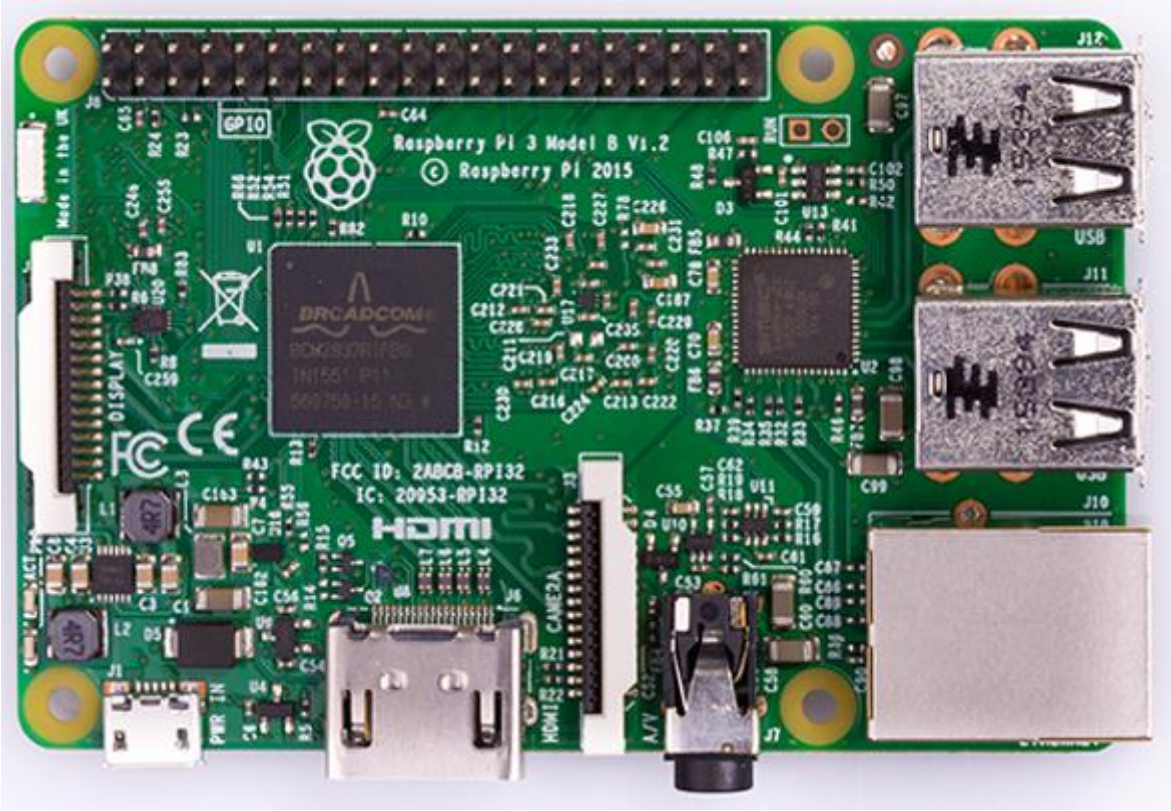
1.5.1 Conceito

Conforme Scheidemantel (2015), o Raspberry Pi é um computador de baixo custo com as dimensões físicas de um cartão de crédito. Assim como os computadores desktop modernos,

possui saída de vídeo, saída de áudio, interface *Universal Serial Bus* (USB), interface *Ethernet*, interface *Wireless* e *Bluetooth*. O Raspberry Pi será, neste texto, doravante referido como Raspberry, Raspberry Pi e Rpi.

A Figura 3 ilustra o Raspberry Pi em sua versão mais recente, o Rpi 3 modelo B.

Figura 3 - Raspberry Pi 3 Modelo B



Fonte: RaspberryPI.org (2017)

O Rpi, considerado o menor computador do mundo, possui o tamanho de um cartão de crédito, conexões USB para conectar o teclado e mouse utilizado em computadores de mesa. É possível conectá-lo a TVs com saída RCA ou HDMI. Além dessas vantagens, pode-se destacar o baixo custo do hardware (em torno de US$35), além do custo zero do software embarcado, baseado no Linux. Todo o hardware é integrado em uma única placa. O principal objetivo dos desenvolvedores foi promover o ensino em Ciência da Computação básica em escolas, principalmente públicas. É um pequeno dispositivo que permite que pessoas de todas as idades possam explorar a computação para aprender a programar em linguagens como Python. É capaz de desenvolver tudo que um computador convencional faz, como navegar na internet, reproduzir vídeo de alta definição, fazer planilhas, processar texto, além de jogar *games* (RASPBERRY, 2017).

> O Raspberry Pi foi idealizado no ano de 2006, nos laboratórios de ciência da computação da Universidade de Cambridge, localizada no Reino Unido. Os professores Eben Upton, Rob Mullins, Jack Lang, e Alan Mycroft deparavam-se com um cenário acadêmico desanimador, pois as notas de programação dos alunos de ciência da computação decaíam gradativamente, ano a ano. Mais preocupante que a progressiva queda de notas, havia uma mudança de paradigma no ambiente universitário. Na década de 1990, os alunos de ciência da computação ingressavam na universidade com amplo conhecimento em programação, conhecimento este decorrente da imensa experiência que adquiriram por hobby. No entanto, os alunos ingressantes após a década de 2000, embora entusiasmados com o curso, demonstravam um conhecimento prático bastante limitado, ou até inexistente. Independentemente das causas que levaram à mudança de mentalidade dos alunos, era preciso reverter o fluxo, e reacender o interesse pela programação por conta própria. A reação materializou-se no Raspberry Pi, mas sua influência rapidamente excedeu o ambiente da ciência da computação e da programação, criando fortes raízes no mundo da eletrônica. Pouco depois do lançamento, surgia uma enxurrada de diferentes projetos e aplicações do pequeno computador. Alguns exemplos da ampla gama de experimentos realizados com o Raspberry Pi são: controlador de aparelho micro-ondas, aparelhos de videogame, telefones celulares controlados por tela LCD sensível ao toque, rádio definido por software (*software-defined radio*), servidores de dados, e, até mesmo um trabalho de pós-graduação em processamento de sinais (SCHEIDEMANTEL, 2015).

Embora venha sem sistema operacional, o Rpi é compatível com várias distribuições Linux, incluindo o Debian, Arch Linux e Fedora. Diferente do que se tem na plataforma PC, não existe uma imagem única para dispositivos ARM[1], já que a plataforma carece de BIOS, enumeração de dispositivos *plug-and-play* e outras funções, o que torna quase impossível para o sistema detectar o hardware automaticamente durante o boot. No mundo ARM é necessário que uma imagem específica do sistema seja desenvolvida para o dispositivo, compilada com as opções corretas. A boa notícia é que vários voluntários já estão fazendo isso, disponibilizando imagens que estão disponíveis para download (PEREIRA E JUCÁ, s.d. apud RASPBERRY, 2015). A instalação destas imagens é muito simples, consistindo apenas em gravar a imagem no cartão SD usando um software ou outro utilitário de cópia bit a bit, plugar o cartão no Rpi e dar boot (PEREIRA E JUCÁ, s.d.).

[1] Arquitetura ARM (primeiramente Acorn RISC Machine, posteriormente Advanced RISC Machine) é uma arquitetura de processador de 32 bits usada principalmente em sistemas embarcados (WIKIPEDIA, 2017).

1.5.2 Sistema operacional

A Raspiberry Pi 3 suporta uma gama de sistemas operacionais. Dentre as opções listadas no site oficial, estão o Windows 10 *IoT* Core, Ubuntu e Raspbian (distribuição dedicada à Raspberry Pi, baseada no Debian).

Dentre os sistemas operacionais com Rpi, o mais utilizado é o Raspbian (necessário cartão SD de 4 GB). Para projetos de eletrônica, robótica, servidores e outros já que deixa livres no Rpi o máximo de recursos possível como memória, processador e consumo de corrente.

Raspian

O Raspbian2 é o sistema operacional oficial do Rpi distribuído pela Raspberry Pi Foundation. Essa é a distribuição ideal para quem tem menos conhecimentos dos sistemas Linux ou simplesmente queira um sistema pronto para funcionar. O Raspbian é um sistema quase completo, já que vem com diversas aplicações pré-instaladas, os drivers mais usuais, ferramentas para facilitar algumas configurações necessárias, entre outros. Muitas aplicações e módulos dedicados à programação já vêm incluídos na imagem do Raspbian, bastando iniciar o sistema para acessá-las. Para iniciantes com o Rpi, que desejam experimentar as potencialidades ou começar a programar e desenvolver projetos de sistemas embarcados, o Raspbian é mais recomendável (PEREIRA E JUCÁ, s.d.).

1.5.3 Modelos

O novo Rpi 3 Model B possui Wi-Fi embutido e Bluetooth 4.1 que consome pouca energia, abrindo um mundo de possibilidades na área de educação e na *IoT*. Em modelos anteriores, podia-se adicionar Bluetooth ou Wi-Fi, usando um *dongle*, mas isso aumenta o custo, o tamanho, e ocupa uma porta USB (RASPBERRY, 2017).

O Rpi 3 traz um processador atualizado da *Broadcom*, com quatro núcleos de 1,2 GHz, suporte a 64 bits e arquitetura ARM Cortex A53.

Assim como o Rpi 2, ele conta com 1 GB de RAM e chip gráfico VideoCore IV. Conta também com quatro portas USB, entrada para cartão microSD, porta Ethernet, saída de áudio e HDMI, e 40 pinos GPIO para conectar a placa a outros dispositivos.

[2] Raspbian OS é uma distribuição Linux criada para rodar nos Raspberry Pi (RASPBERRYPI, 2017)

Quadro 1 - Modelos Rpi

	Raspberry Pi A+	Raspberry Pi model B	Raspberry Pi 2	Rasberry Pi Zero	Raspberry Pi Zero W	Raspberry Pi 3
Lançamento	10/11/2014	15/02/2012	01/02/2015	30/11/2015	28/02/2017	29/02/2016
Preço US$	US$20.00		US$35.00	US$5.00	US$10.00	US$35.00
Tipo Chip	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2837
Tipo Core	ARM1176JZF-S	ARM1176JZF-S	Cortex-A7	ARM1176JZF-S	ARM1176JZF-S	Cortex-A53 64-bit
Nº Core	1	1	4	1	1	4
Clock CPU	700 MHz	700 MHz	900 MHz	1 GHz	1 GHz	1.2 GHz
GPU	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV
RAM	256 MB	512 MB (256 - model A)	1 GB	512 MB	512 MB	1 GB
Wireless	X	X	X	X	802.11n	802.11n
Bluetooth	X	X	X	X	4.1	4.1
Consumo	200 mA	700 mA	800 mA	160 mA	180 mA	800 mA

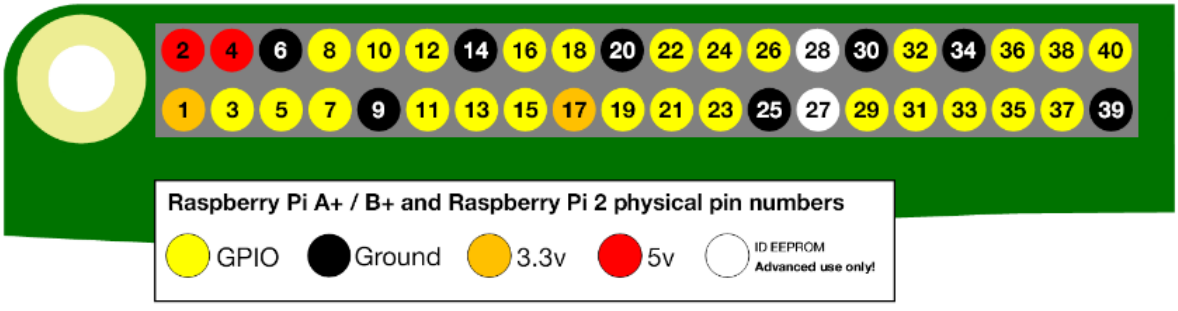
Fonte: wikipedia.org (2017)

1.5.4 GPIO (General Purpose Input/Output)

O GPIO, é basicamente um conjunto de pinos responsável por fazer a comunicação de entrada e saída de sinais digitais. Ele é composto por 26 pinos no Raspberry Pi B, e 40 pinos no Raspberry Pi B+. Com esses pinos, é possível acionar LEDs, motores, relês, fazer leitura de sensores entre outras funcionalidades. Todo pino que possui as funcionalidades gerais de entrada e saída e não possui atribuições dedicadas, pode ser classificado como pino de General Purpose Input/Output. Estes pinos podem ser configurados como saída (estado lógico alto 1 ou baixo 0) (RASPBERRY, 2017).

O nível de tensão do GPIO é de 3.3V e não de 5V. Não existe qualquer tipo de proteção de sobrecarga. A intenção é que pessoas interessadas no uso intensivo desses pinos utilizem uma placa externa ao invés da ligação direta no Rpi, porém nada impede o uso direto, desde que sejam tomadas as devidas precauções (RASPBERRY, 2017).

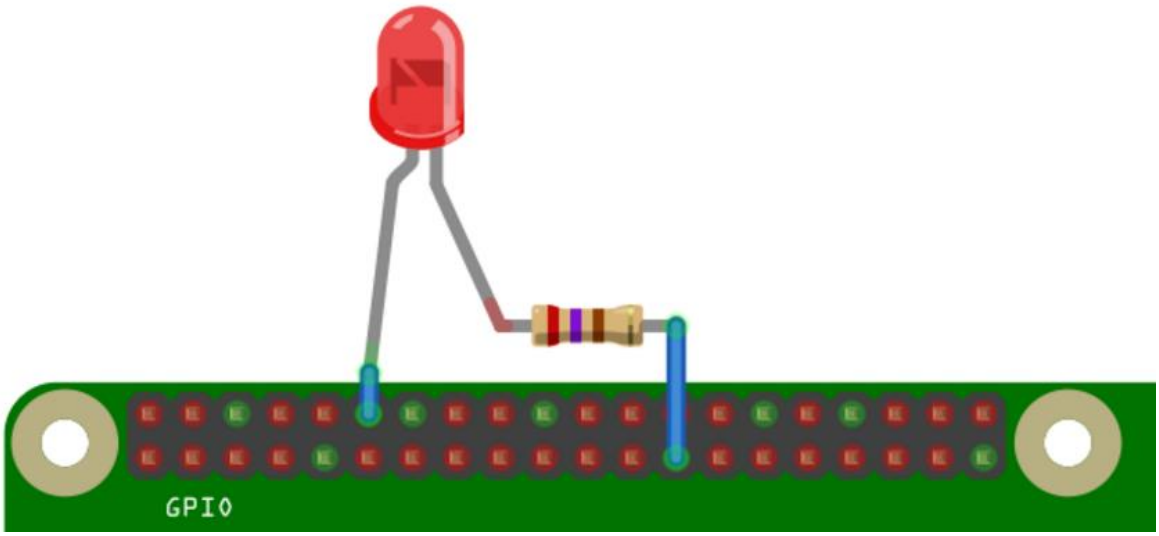
Figura 4 - GPIO



Fonte: RaspberryPI.org (2017)

Segundo Scheidemantel (2015) os pinos de GPIO do Raspberry Pi podem servir para diversas funções. O exemplo mais simples que vem à tona é o comando de um LED via um script escrito em Python. Para tanto, basta conectar os terminais do LED conforme ilustra o diagrama apresentado na Figura 5. O LED, então, pode ser facilmente controlado por meio de um código programado pelo usuário. O fato de o Rpi ter as funções de um computador agora se torna vantajoso. A depender do código escrito pelo usuário, o controle do LED pode ser feito até mesmo via Internet, para sinalizar eventos como a chegada de uma mensagem eletrônica, por exemplo.

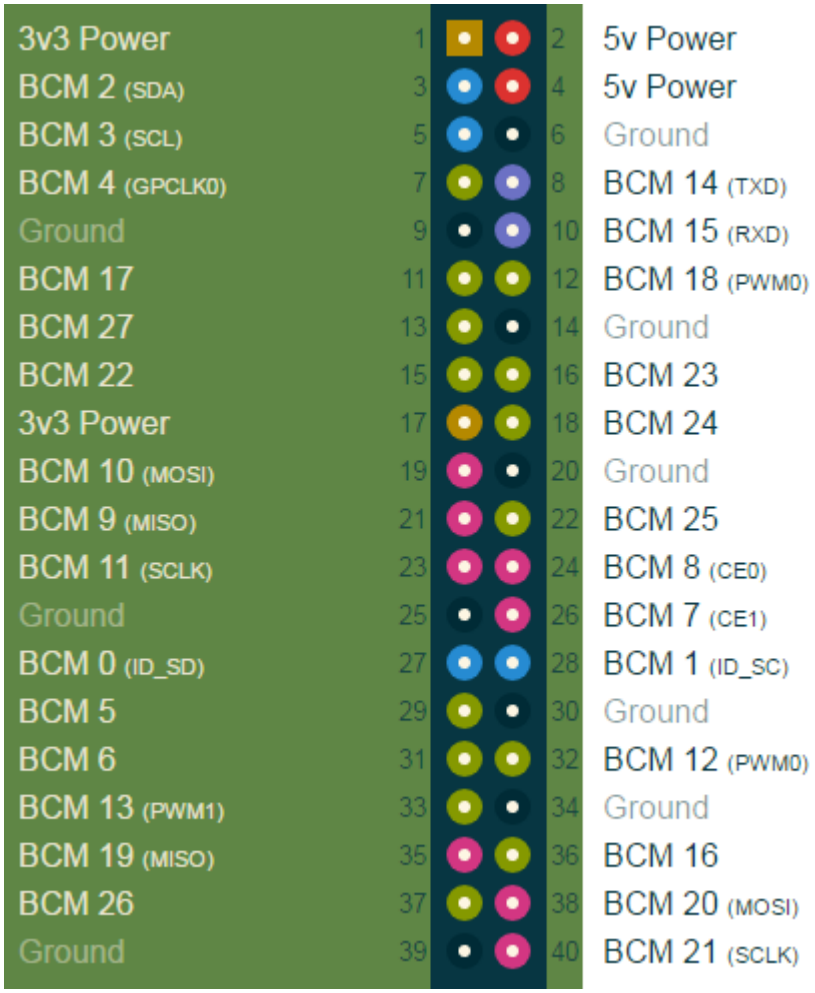
Figura 5 - LED controlado via pino GPIO



Fonte: RaspberryPI.org (2017)

É interessante notar, também, que os pinos possuem dois padrões de numeração, sendo um deles de acordo com a organização espacial dos pinos *BOARD* e outro de acordo com o circuito integrado controlador *Broadcom* da interface GPIO que é representada pela numeração antecedida pelo prefixo BCM na Figura 6 (RASPBERRY, 2017).

Figura 6 - Numeração dos pinos GPIO e suas funções específicas



Fonte: RaspberryPI.org (2017)

Para que se possa ter um conhecimento mais aprofundado dos pinos de GPIO, é necessário primeiro saber qual é a função de cada um; embora qualquer pino GPIO possa servir para leitura e escrita de bits, existem pinos especificamente projetados para determinados protocolos de comunicação, como pode ser visto na Figura 6 (RASPBERRY, 2017).

Os pinos na Figura 6 são diferenciados por cores e seus significados é apresentado na sequência.

Pinos vermelhos: Esta é uma saída para alimentação, e possui uma tensão de 5V.

Pinos laranjas: Esta também é uma saída para alimentação, porém com uma tensão de 3.3V.

Pinos pretos: Estas são simplesmente as portas Terra (*GROUND*), e, nela, não existe tensão.

Pinos azuis: Essas duas portas podem ser programadas para interface I2C (Circuito Inter-integrado). Este é um protocolo criado pela Philips em 2006, para fazer conexões entre periféricos de baixa velocidade.

Pinos roxos: Estas são as portas seriais, que utilizam o protocolo RS-232 para o envio e recebimento de sinal digital.

Pinos verdes: Aqui estão os pinos GPIO que falamos anteriormente. Eles servem para fazer envio e recebimento de dados digitais.

Pinos rosas: Estes pinos são também para entrada e saída de dados digitais. Porém, eles possuem uma característica a mais. Com estes pinos, é possível fazer uma comunicação serial Full Duplex síncrono, que permite o processador do Rpi comunicar com algum periférico externo de forma bidirecional. Mas essa comunicação só acontece se o protocolo for implementado.

Pinos azuis (Pinos 27 e 28): Essas são as portas do ID EEPROM (*Electrically-Erasable Programmable Read-Only Memory*). É um tipo de memória que pode ser programada e apagada várias vezes , por meio de uma tensão elétrica interna ou externa.

1.5.5 Aplicações do Raspberry

A Raspberry Pi, é uma plataforma interessante para ser usada em diversos tipos de projetos computacionais, sendo que se destaca pelo seu baixo custo e seu significativo poder de processamento comparado ao seu tamanho. Assim, a Rpi apresenta um desempenho considerável em diversas tarefas executadas, como por exemplo, na implantação de um servidor para monitoramento de redes IPv6 com Zabbix, conforme projeto desenvolvido por (LIMA et al., s.d. apud LEMES, 2013).

Em outro projeto foi usado a Rpi juntamente com RFID (*Radio Frequency Identification*), onde o autor (LIMA et al., s.d. apud RAULINO, 2013) desenvolveu um sistema computacional que faz monitoramento constante de atividades para natação, a Rpi foi responsável por calcular e disponibilizar a quantidade de voltas percorridas, o tempo e a velocidade na piscina para apenas um ou diversos nadadores ao mesmo tempo, gerando assim dados para análises do desempenho físico dos atletas.

1.6 SENSORES

Um sensor, conforme descrito por Tooley (2007), “é um tipo especial de transdutor utilizado para gerar um sinal de entrada para um sistema de medição, instrumentação ou controle”. Com a utilização de transdutores em circuitos eletrônicos obtêm-se leituras de diversas grandezas, tais como: velocidade, pressão, temperatura, luminosidade, nível, fluxo, peso, vibração.

Um sensor nem sempre possui as características elétricas necessárias para ser utilizado em um sistema de controle. Normalmente, o sinal de saída deve ser manipulado antes da sua leitura no sistema. Isso geralmente é realizado com um circuito de interface para produção de um sinal que possa ser lido (WENDLING, 2010).

A seguir são apresentados os sensores escolhidos para uso no protótipo deste TCC.

1.6.1 Sensor de presença

O sensor PIR mostrado na Figura 7 é um módulo automático de controle, que utiliza um sensor piroelétrico capaz de detectar movimentos baseados na variação da luz infravermelha emitida pela radiação do corpo humano.

Festo (2010) descreve esse tipo de sensor piroelétrico como um “sensor que percebe a presença de pessoas no ambiente por meio da radiação de calor emitida pelo corpo em movimento”.

Figura 7 - Sensor PIR

Fonte: O autor

É possível ajustar a duração do tempo de espera para estabilização do PIR por meio do potenciômetro amarelo embaixo do sensor bem como sua sensibilidade. A estabilização pode variar entre 5-200 seg (FESTO, 2010).

1.6.2 Sensor de gás

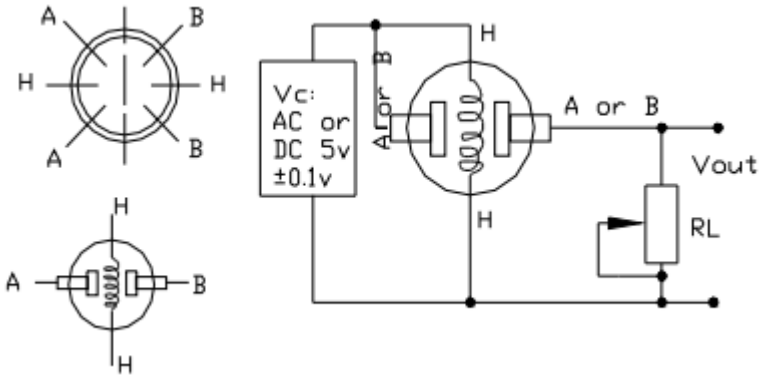
Composto por micro tubo AL2O3 de cerâmica e uma camada sensível de dióxido de estanho (SnO2), precisa um aquecimento dos componentes sensíveis para o funcionamento. O sensor de gás MQ-5 conforme ilustrado na Figura 8, é de resposta rápida e alta sensibilidade para GLP (gás de cozinha) e gás natural, e baixa sensibilidade para álcool e fumaça. Tem uma saída com uma tensão analógica e outra digital. Pode medir concentrações na faixa de 200 a 10.000ppm, opera entre -20 e 50ºC e consome menos de 800mA a 5V (HANWEI ELECTRONICS, s.d.).

Figura 8 - Sensor de gás MQ-5

Fonte: HANWEI ELECTRONICS (s.d.)

Para um funcionamento preciso e otimizado devemos conectar 5V nos pinos do aquecedor, que são os pinos H, para que o sensor possa chegar à temperatura de funcionamento. Conectando uma tensão aos pinos A faz o sensor emitr uma tensão analógica nos outros pinos B. Uma resistência entre a saída do sensor e o GND determina o grau de sensibilidade do conjunto. O sensor tem 6 pinos cada, 4 deles são usados para buscar sinais, e outros 2 são usados para corrente de aquecimento, como mostrado na Figura 9 (HANWEI ELECTRONICS, s.d.).

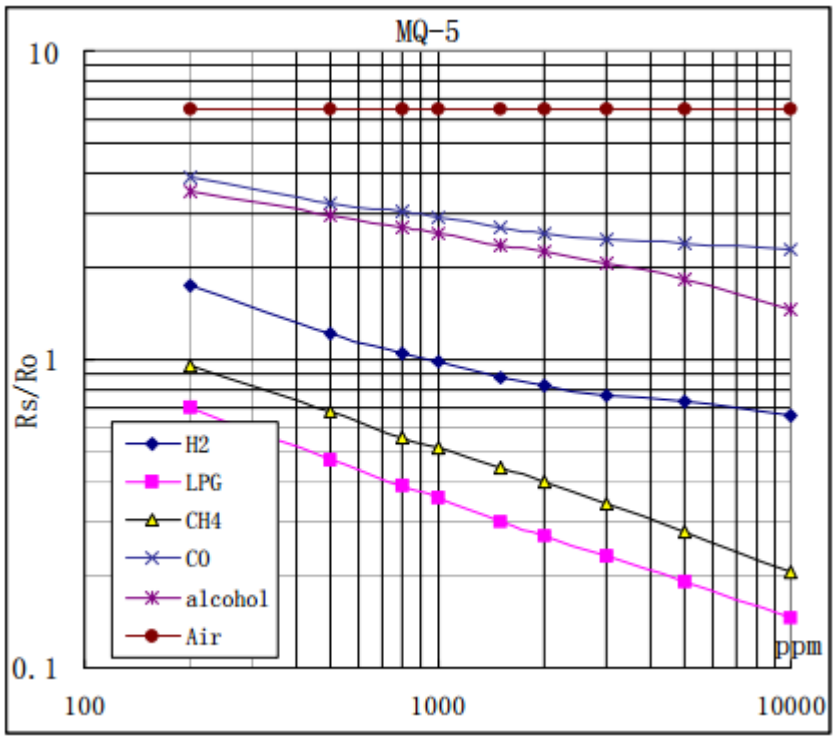
Figura 9 - Esquemático do sensor MQ-5



Fonte: HANWEI ELECTRONICS (s.d.)

A Figura 10 mostra a características de sensibilidade do MQ-5 para vários gases a uma temperatura de 20 ℃ e usando um resistor RL de 20KΩ. R0 é a resistência do sensor em 1000ppm de GLP e Rs é a resistência do sensor. No eixo X temos as concentrações em PPM e no eixo Y os valores de Rs/R0 (HANWEI ELECTRONICS, s.d.).

Figura 10 - Característica sensitiva do sensor MQ-5



Fonte: HANWEI ELECTRONICS (s.d.)

1.7 LINGUAGEM DE PROGRAMAÇÃO PYTHON

Criado por Guido Van Rossum a linguagem Python foi utilizada nesse projeto pela facilidade de integração com a plataforma Rpi, pela sua facilidade de codificação devido a sua estrutura orientada a objeto, e por ser uma linguagem com variáveis dinâmicas, ou seja, uma mesma variável pode receber diversos tipos de valores. Exemplo disso é que uma variável pode ser uma *string* e numa linha seguinte pode ser um inteiro (PYTHON, 2015).

Outra facilidade está na disponibilidade de duas versões, Python 2 e Python 3 no SO Raspian, que pode ser amplamente utilizada para desenvolver e depurar os códigos na própria plataforma Rpi.

1.8 PROTOCOLO MQTT

Na internet 'tradicional', existem diversos protocolos de comunicação responsáveis por gerenciar a transferência de dados em uma rede que conecta dois ou mais computadores. Para citar alguns exemplos: HTTP, FTP e SFTP. Considerando o ambiente da *IoT*, com capacidade computacional limitada e rede com perdas, esses protocolos não são indicados.

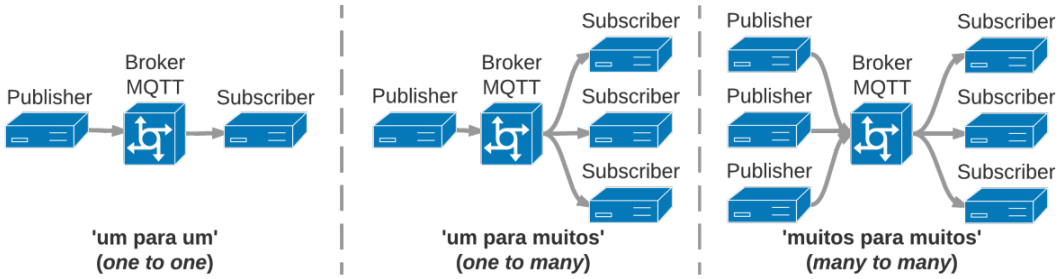
Devido sua versatilidade e compatibilidade, o protocolo de mensagens MQTT (*Message Queuing Telemetry Transport*) torna-se uma interessante escolha para comunicação de dispositivos *IoT*, pois seus princípios de design são para um baixo consumo de banda de rede e requisitos de hardware sendo extremamente simples e leve.

O MQTT foi desenvolvido pela IBM e Eurotech e é projetado para enviar dados através de redes intermitentes ou com baixa banda de dados. Para isso o protocolo é desenvolvido em cima de vários conceitos que garantem uma alta taxa de entrega das mensagens (VICENZI apud CHEN et al., 2014).

Conforme Desai (2015), o MQTT utiliza o paradigma *publish*/*subscribe* (pub/sub) para a troca de mensagens. O paradigma pub/sub implementa um middleware chamado de broker. O broker é responsável por receber, enfileirar e disparar as mensagens recebidas dos *publishers* para os *subscribers*.

Essa estrutura permite desacoplar o produtor do cliente e, assim, apenas o endereço do broker precisa ser conhecido, possibilitando a comunicação de um para um, um para muitos ou muitos para muitos, conforme Figura 11.

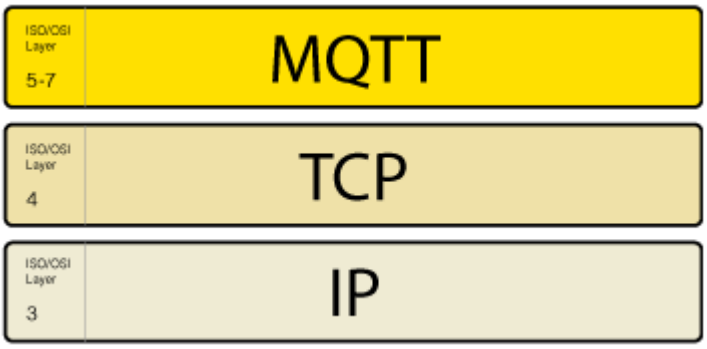
Figura 11 - Tipos de distribuição de mensagens suportados pelo MQTT



Fonte: TORRES et al (2016)

O protocolo MQTT é baseado no TCP/IP e ambos, cliente e broker, necessitam da pilha TCP/IP para o seu funcionamento (VICENZI apud CHEN et al., 2014). Na Figura 12 é exibido o modelo do protocolo no padrão OSI.

Figura 12 - Modelo OSI do protocolo MQTT



Fonte: VICENZI (2015)

Cada uma das chamadas mensagens de comando MQTT possui um cabeçalho fixo composto de dois bytes, onde o primeiro byte contém o campo que identifica o tipo da mensagem e também campos marcadores (DUP, nível QoS e RETAIN) (IBM; EUROTECH, 2010). O Quadro 2 mostra o formato do cabeçalho fixo das mensagens.

Quadro 2 - Cabeçalho fixo de uma mensagem MQTT

bit	7	6	5	4	3	2	1	0
byte 1	Tipo da mensagem				DUP flag	Nível QoS		RETAIN
byte 2	Largura restante							

Fonte: IBM & EUROTECH (2010)

IBM e Eurotech, (2010), esclarece que o tipo da mensagem ocupa do *bit* 7 ao 4 e ao primeiro *byte* do cabeçalho fixo e é representado por um valor não assinado. A lista de tipos definida na versão 3.1 do MQTT é descrita no Quadro 3. Os quatro *bits* restantes do primeiro *byte* dividem-se em quatro campos que servem de marcadores, para indicar preferências definidas antes do envio da mensagem. São eles:

- *Duplicate delivery* (DUP): acrônimo relativo à entrega duplicada, este marcador ocupa o *bit* 4 e é ativado quando o cliente ou o servidor tentam reenviar mensagens do tipo *PUBLISH*, *PUBREL*, *SUBSCRIBE* ou *UNSUBSCRIBE* (ver Quadro 3) que tenham *Quality of Service (*QoS) maior que 0 e requeiram *acknowlegment* (*ACK*).
- *Quality of Service (*QoS): este marcador ocupa os dois penúltimos *bits* e indica o nível de garantia da entrega de uma mensagem *PUBLISH*. Os níveis de QoS são mostrados no Quadro 4.

- *RETAIN*: quando um cliente envia uma mensagem *PUBLISH* ao servidor com este marcador ativado, ela deve ser retida no servidor, mesmo depois de ser entregue aos assinantes. No evento de uma nova subscrição a um tópico, a última mensagem retida para este tópico deve ser enviada para o novo assinante caso este marcador esteja ativado.

Quadro 3 - Tipos de mensagem

Mnemônico	Enumeração	Descrição
Reserved	0	Reservado
CONNECT	1	Requisição de conexão do cliente ao servidor
CONNACK	2	ACK de conexão
PUBLISH	3	Publicação de mensagem
PUBACK	4	ACK de publicação
PUBREC	5	Publicação recebida (garantia de entrega parte I)
PUBREL	6	Publicação liberada (garantia de entrega parte II)
PUBCOMP	7	Publicação completa (garantia de entrega parte III)
SUBSCRIBE	8	Requisição de subscrição do cliente
SUBACK	9	ACK de subscrição
UNSUBSCRIBE	10	Requisição de cancelamento de subscrição do cliente
UNSUBACK	11	ACK de cancelamento de subscrição
PINGREQ	12	Requisição PING
PINGRESP	13	Resposta PING
DISCONNECT	14	Cliente desconectando
Reserved	15	Reservado

Fonte: IBM & EUROTECH (2017)

A largura restante do cabeçalho fixo (*byte* 2) é usada para representar nada mais do que a quantidade de *bytes* remanescentes na mensagem, incluindo dados do cabeçalho variável e do *payload* (IBM; EUROTECH, 2010).

Cabeçalho variável é um componente presente em alguns tipos de mensagem MQTT e está localizado entre o cabeçalho fixo e o *payload.* Usado principalmente nas mensagens *CONNECT*, este cabeçalho possui dois campos para nome e versão do protocolo, respectivamente, e mais uma série de marcadores que definirão algumas diretivas para a conexão entre cliente e servidor (IBM; EUROTECH, 2010).

Quadro 4 - Nível de Qos

Valor QoS	Bit 2	Bit 1	Descrição		
0	0	0	Até uma vez	Disparar e esquecer	<=1
1	0	1	Ao menos uma vez	Entrega com ACK	>=1
2	1	0	Exatamente uma vez	Entrega garantida	=1
3	1	1	Reservado		

Fonte: IBM & EUROTECH (2010)

O payload pode armazenar diferentes tipos de informação, tudo vai depender do tipo da mensagem transmitida: *CONNECT* (irá conter ID do cliente), *SUBSCRIBE* (contém tópicos que o cliente pode subscrever e/ou nível QoS) ou *SUBACK* (lista de níveis de QoS garantidos pelo servidor) (IBM; EUROTECH, 2010).

Desde março de 2013, o MQTT está em processo de padronização na OASIS (um consórcio sem fins lucrativos que conduz o desenvolvimento, convergência e adoção de padrões abertos para a sociedade da informação) (VICENZI, 2014).

1.9 NODE-RED

Inventado por J. Paul Morrison na década de 1970, a programação baseada em fluxo é uma maneira de descrever o comportamento de uma aplicação como uma rede de nós (NODE-RED, 2017).

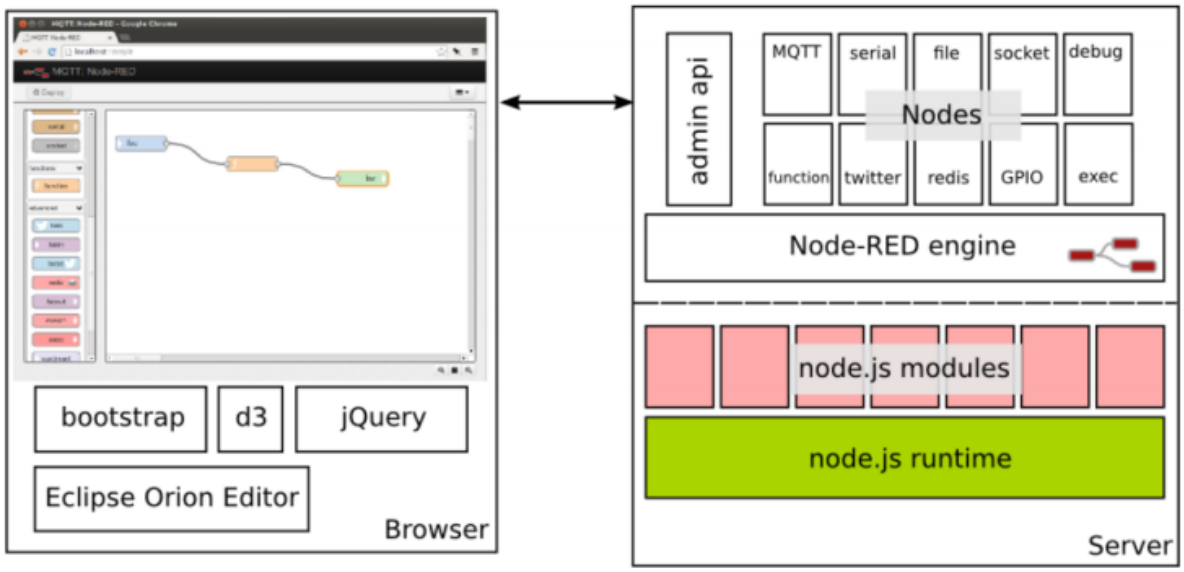
O Node-RED é um editor de *workflows* multiplataforma que possui interfaces ricas baseadas em Javascript e Node.js, que permite ao desenvolvedor modelar, e monitorar a execução desse fluxo de nós (NODE-RED, 2017).

Node-RED, (2017) afirma que é um modelo que se presta muito bem a uma representação visual e torna mais acessível a uma ampla gama de usuários. Se alguém pode quebrar um problema em passos discretos, eles podem olhar para um fluxo e ter uma ideia do que está fazendo; sem ter que entender as linhas de código individuais dentro de cada nó.

A Figura 13 mostra uma visão da estrutura do Node-RED. Pode-se observar, à direita, a estrutura no lado servidor constituída por um ambiente em tempo de execução do Node.js e um motor do Node-RED. É no servidor onde se executam os nós que tratam os fluxos de mensagens. No lado esquerdo da figura, pode-se observar o lado cliente (browser), que

proporciona um editor gráfico que permite selecionar nós de uma biblioteca e interconectá-los (MARIANO, 2016).

Figura 13 - Estrutura Node-RED



Fonte: MARIANO (2016)

Segundo Mariano (2016), cada fluxo é armazenado usando JSON, sendo possível copiar fluxos inteiros ou partes, copiando o texto JSON de outro projeto. Por rodar sobre Node.js, ter acesso as suas bibliotecas, e permitir a criação de novos nós pela comunidade (compostos de arquivos HTML e Javascript), o Node-RED permite grande flexibilidade para integrar diferentes tecnologias. Essa flexibilidade e facilidade de uso o tornou uma opção interessante para implementar a parte ativa do sistema supervisório de energia que é objetivo deste trabalho.

O Node-RED possui um módulo de protocolo MQTT capaz de interagir com o servidor Mosquitto, um servidor MQTT, permitindo, desta forma, a comunicação com plataformas e sistemas que se comunicam com esse protocolo (NODE-RED, 2017). Neste projeto, implantou-se o MQTT em uma Rpi.

Node-RED começou a vida, no início de 2013, como um projeto paralelo de Nick O'Leary e Dave Conway Jones do grupo *Emerging Technology Services* da IBM. O que começou como uma prova de conceito para visualizar e manipular mapeamentos entre tópicos do MQTT, tornou-se, rapidamente, uma ferramenta muito mais geral que poderia ser facilmente estendida em qualquer direção. Foi aberto em setembro de 2013, culminando em ser um dos projetos fundadores da Fundação JS em outubro de 2016 (NODE-RED, 2017).

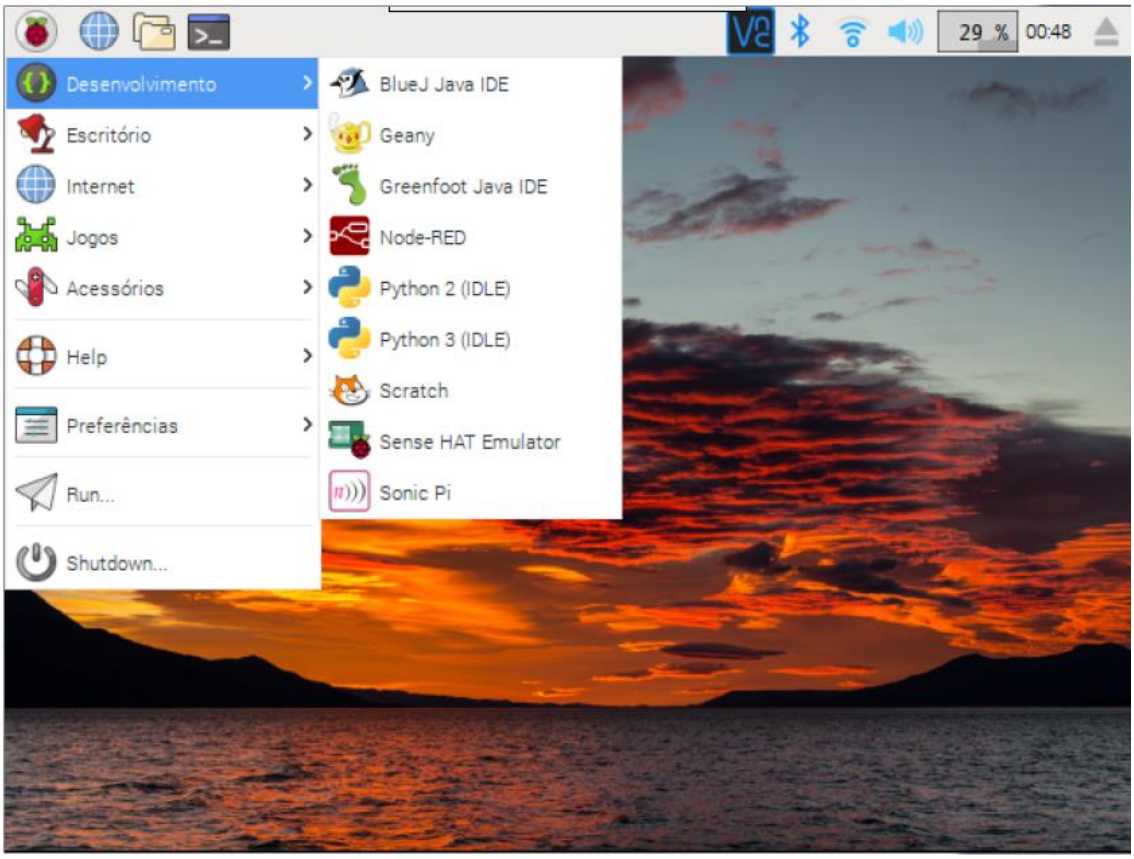
2 DESENVOLVIMENTO E RESULTADO

As seções a seguir descrevem a implementação e a operacionalidade do protótipo, abordando sucintamente as ferramentas e etapas utilizadas no desenvolvimento do projeto. No fim, são mostrados os resultados obtidos com este TCC.

2.1 AMBIENTE

A instalação do sistema operacional Raspian na placa Rpi é relativamente fácil, já que a imagem do SO já vem pré-configurada e pronta para usar. Por meio do Windows, basta fazer o procedimento de instalação em um cartão microSD. A imagem pode ser baixada do site do próprio fabricante da Rpi.

Figura 14 - Área de trabalho do Rpi



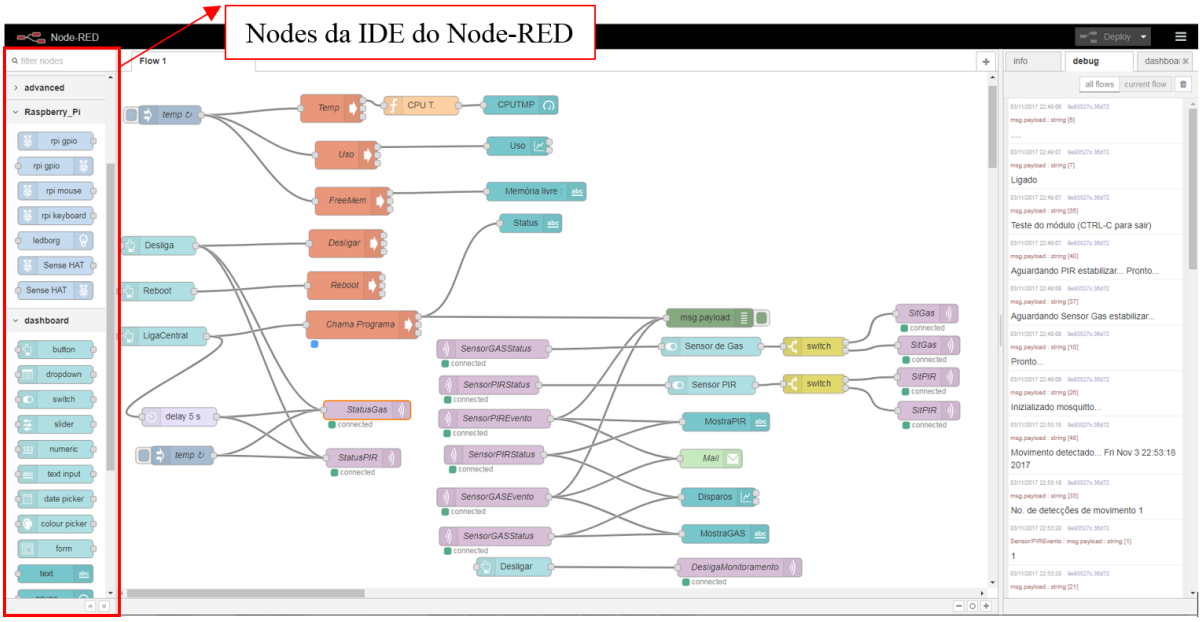
Fonte: O Autor (2017)

O servidor Node-RED foi instalado junto ao ambiente Node.js, por meio do gerenciador de pacotes NPM[3]. Utilizando o NPM também é possível adicionar nós para serem utilizados na

[3] NPM é o nome reduzido de Node Package Manager (Gerenciador de Pacotes do Node)

interface do editor de fluxos conforme a Figura 15. A configuração padrão do Node-RED disponibiliza a interface na porta 1880.

Figura 15 - IDE do Node-RED

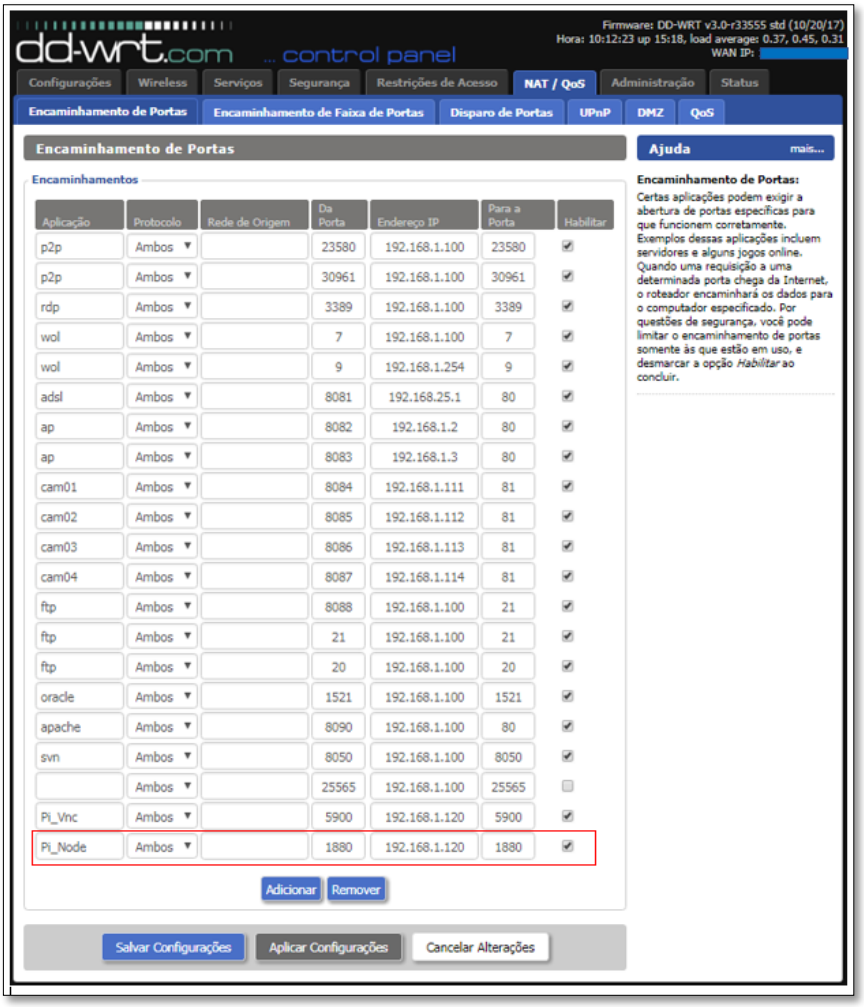


Fonte: O Autor (2017)

O servidor Mosquitto, que age como mediador MQTT, foi instalado com as configurações básicas. Ele espera mensagens na porta 1883.

A Figura 16 mostra a porta do Node-RED liberada por intermédio da NAT do roteador para acesso externo.

Figura 16 - Encaminhamento da porta Node-RED



Fonte: O Autor (2017)

2.2 ESPECIFICAÇÃO

A especificação física do trabalho foi criada utilizando as ferramentas Gliffy, para gerar o diagrama de topologia e o Fritzing, para gerar os diagramas esquemático e de protoboard. Quanto ao desenvolvimento da aplicação, foi utilizado a Metodologia Ágil *Extreme Programming* (XP).

O XP é na verdade um dos modelos de engenharia de software ágil, que segundo Pressman:

> A engenharia de software ágil combina filosofia com um conjunto de princípios de desenvolvimento. A filosofia defende a satisfação do cliente e a entrega de incremental prévio; equipes de projetos pequenas e altamente motivadas; métodos informais; artefatos de engenharia de software mínimos e, acima de tudo, simplicidade no desenvolvimento geral. Os princípios de desenvolvimento priorizam a entrega mais

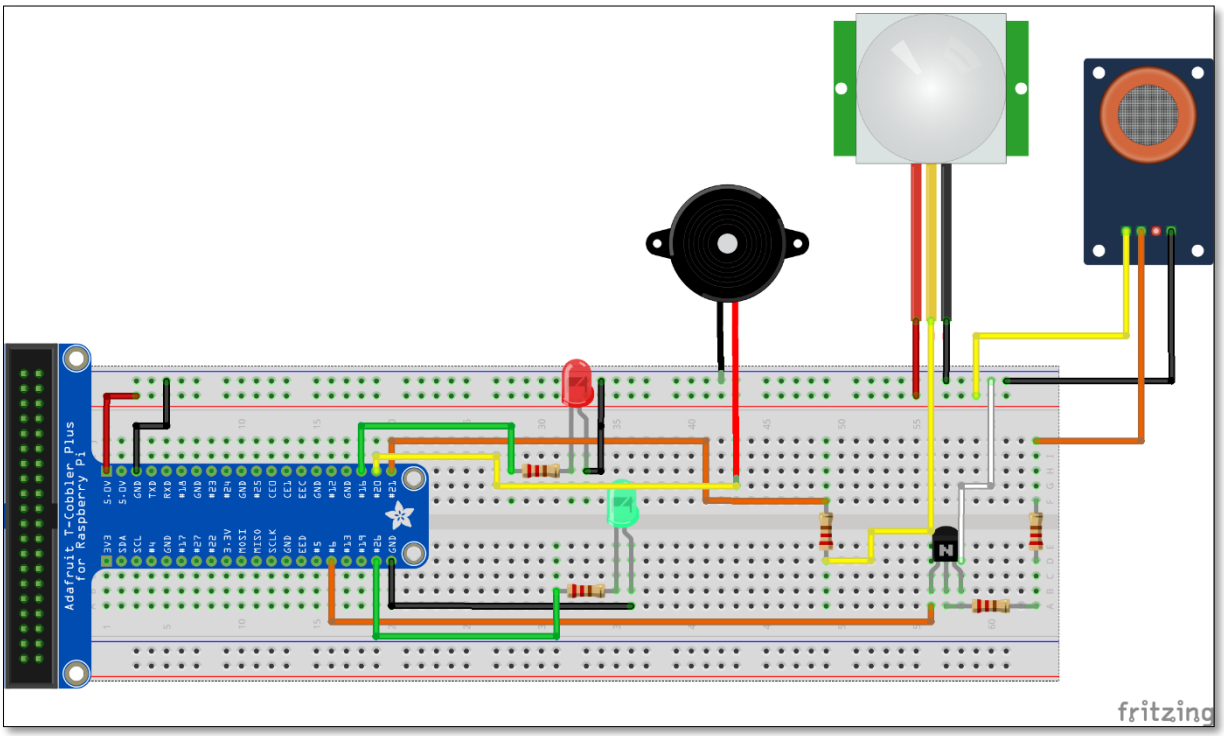
> que a análise e projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e contínua entre desenvolvedores e clientes" (Pressman, 2011).

O XP é um método sistêmico para desenvolvimento de software. O processo inicia por um plano geral, ou seja, entender o problema como um todo, mas sem muitos detalhes e, posteriormente, reparte o problema em partes, para dar início ao seu desenvolvimento.

2.2.1 Diagrama protoboard

Na Figura 17, é especificado o diagrama protoboard do protótipo construído para a captura de informações e controle dos sensores.

Figura 17 - Protoboard do protótipo

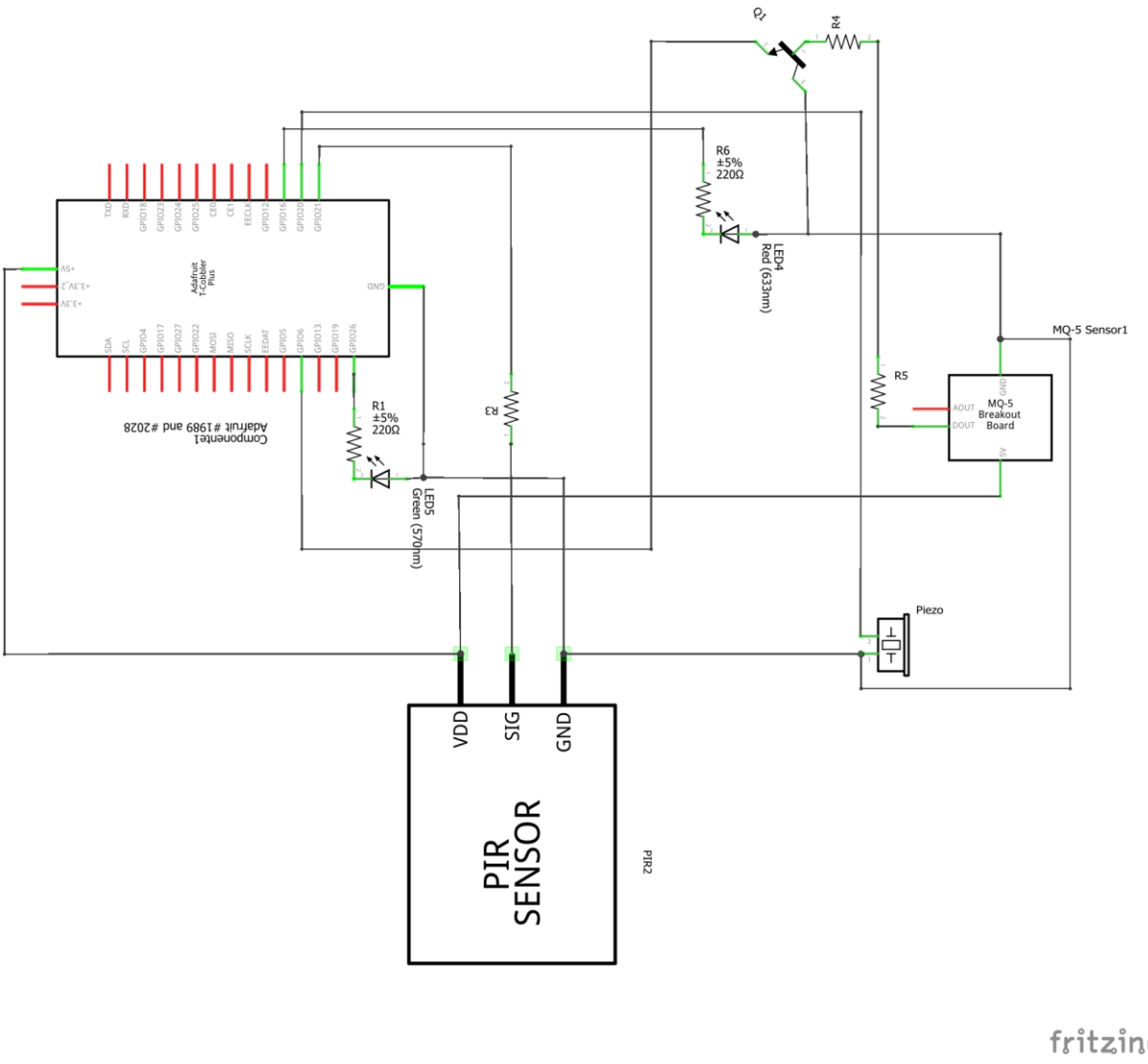


Fonte: O Autor (2017)

2.2.2 Diagrama esquemático

Na Figura 18, é especificado o diagrama esquemático do protótipo construído para a captura de informações dos sensores.

Figura 18 - Diagrama esquemático

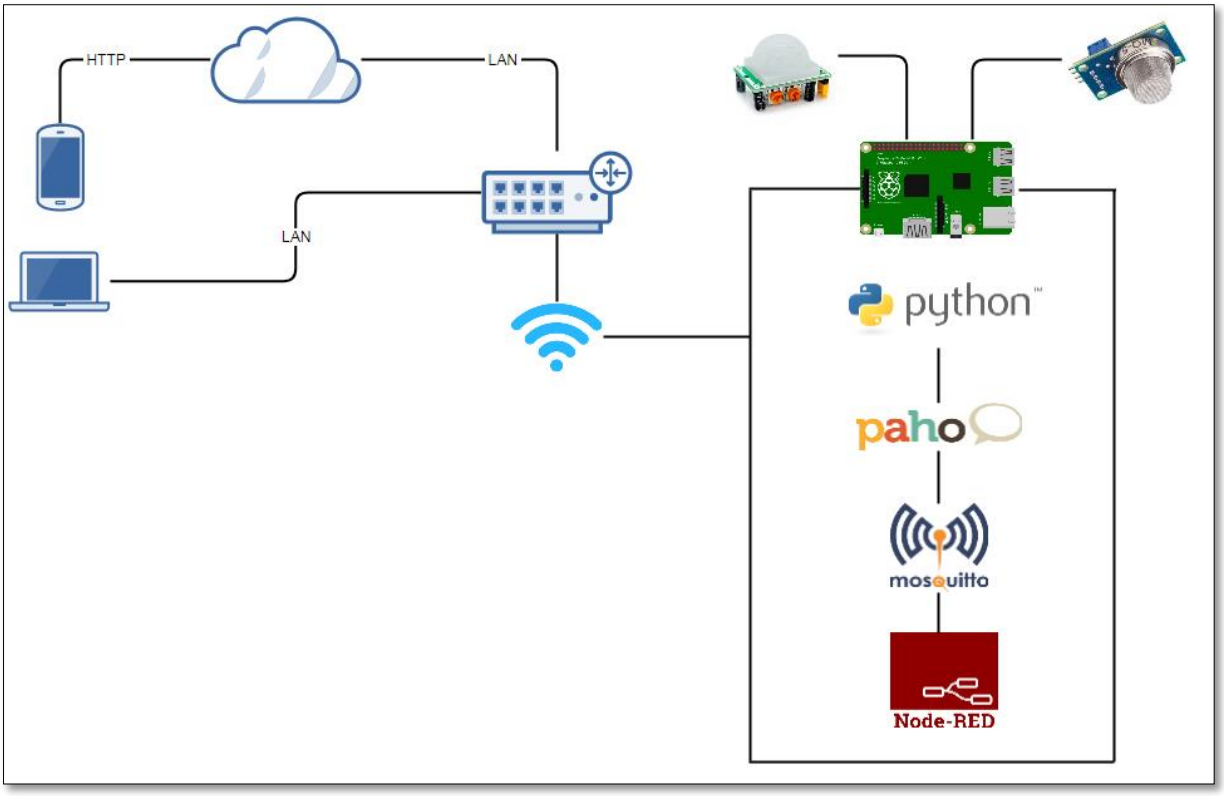


Fonte: O Autor (2017)

2.2.3 Diagrama de topologia

A Figura 19 demonstra a topologia dos sistemas desenvolvidos, funcionando em conjunto.

Figura 19 - Topologia do sistema



Fonte: O Autor (2017)

2.3 IMPLEMENTAÇÃO

A seguir, são mostradas as técnicas e ferramentas utilizadas na implementação. Para maior distinção entre as aplicações desenvolvidas elas serão nomeadas a seguir:

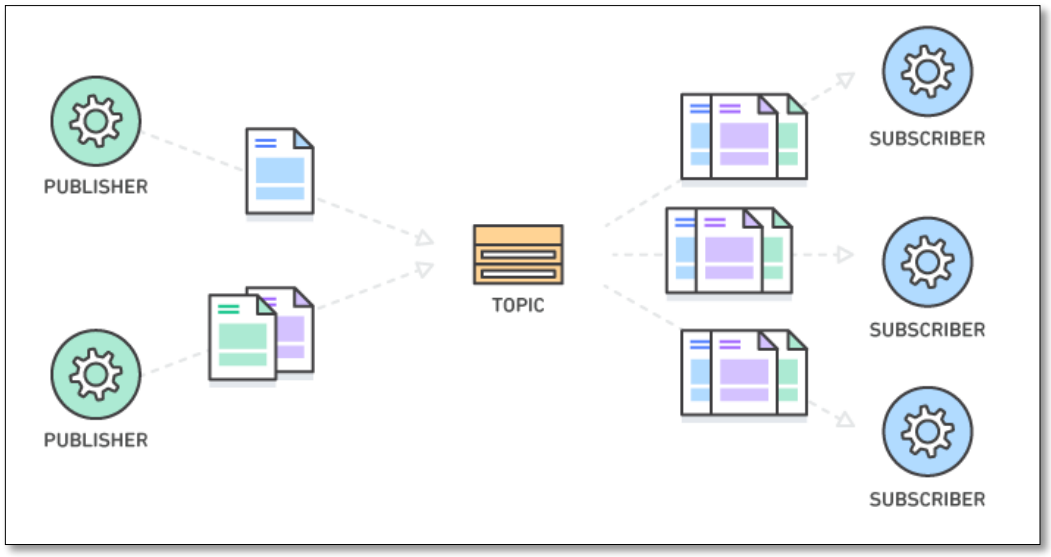
a) Módulo de controle: aplicação em Python, embarcado na Rpi, para controlar o GPIO, sensores e as interações com o *broker*;
b) Módulo de acesso: Interface desenvolvida com Node-RED, para conectar a aplicação de controle e interagir com o MQTT.

2.3.1 Técnicas e ferramentas utilizadas

Um ponto importante a ser destacado no desenvolvimento, refere-se ao *parttner Publisher/subscriber*. Este fornece notificações de eventos instantâneos para os módulos do protótipo, um padrão de mensagens em que os remetentes, chamados *publishers*, não programam as mensagens para serem enviadas diretamente para receptores específicos, chamados *subscribers*, permitindo que as mensagens sejam transmitidas para diferentes partes do sistema de forma assíncrona. Um tópico de mensagem fornece um mecanismo leve para

transmitir notificações de eventos assíncronos, permitindo que os componentes de software se conectem ao tópico para enviar e receber essas mensagens. Conforme demonstrado na Figura 20, todos os componentes que se inscrevem no tópico receberão todas as mensagens que são transmitidas, a menos que uma política de filtragem seja definida pelo assinante (AWS, 2017).

Figura 20 - *Subscribes* recebendo todas as mensagens publicadas



Fonte: AWS (2017)

Diante desse contexto, a aplicação desenvolvida é um projeto de sistema distribuído na arquitetura cliente-servidor. A camada cliente é responsável pela interface com o usuário e pela comunicação com o *broker*. A camada servidor é responsável pelo controle dos sensores e processamento das informações geradas (Módulo de controle).

2.3.2 Partes do sistema

A seguir, é detalhada a implementação de cada um dos sistemas que compõem este trabalho. Inicialmente é abordado o módulo de controle, parte principal do sistema responsável pelo processamento dos dados geradas na Rpi. Por fim, é abordado o módulo encarregado pelo fornecimento das informações aos usuários do sistema de segurança residencial.

2.3.2.1 Sistema de segurança embarcado

O módulo de controle é responsável pela operação dos sensores utilizados pelo Raspberry Pi e envio dos dados obtidos ao servidor MQTT. A partir desse protocolo, é possível fazer a comunicação com o Node-RED, no qual foi implementada a interface com os usuários.

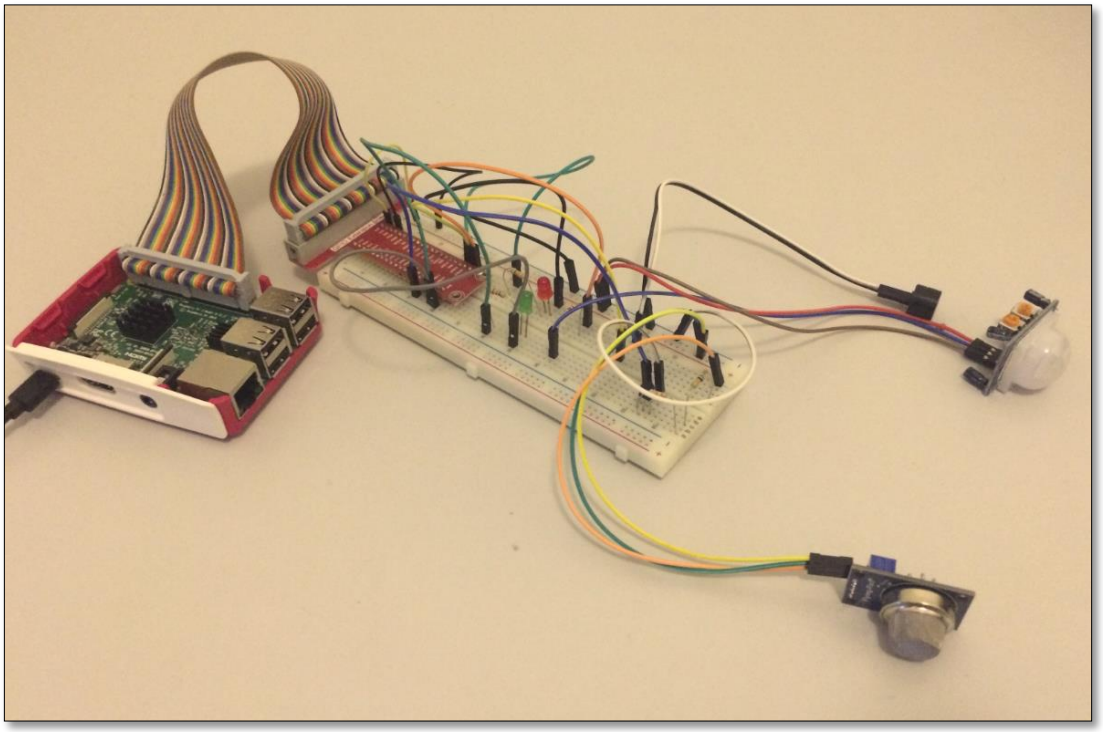
O servidor responsável por receber as mensagens do protótipo é o Mosquitto. O Mosquitto é um *broker* de código aberto que implementa as versões 3.1 e 3.1.1 do protocolo MQTT, usando o modelo *publish* / *subscribe* o que o torna adequado para a troca de mensagens entre dispositivos (MOSQUITTO, 2015).

Para o desenvolvimento do software do dispositivo embarcado, utilizou-se a linguagem de programação Python, ao qual se deu uma boa versatilidade para o acesso aos GPIOs do dispositivo e controle dos sensores.

O hardware do sistema desenvolvido pode ser observado na Figura 21. O Protótipo é composto de uma placa de prototipagem no qual foram acoplados os sensores e os componentes eletrônicos descritos no Quadro 5. A parte física do dispositivo é constituído de duas partes fundamentais: placa Raspberry Pi e sensores.

A placa Raspberry Pi é responsável por gerenciar a interface com a rede TCP/IP, acionamento das entradas e saídas, leitura dos dados coletados pelos sensores. Os sensores são conectados a Rpi por meio dos pinos de uso geral GPIO, são responsáveis por fazer leituras do ambiente, alterando seu nível lógico quando ele muda de estado;

Figura 21 - Protótipo



Fonte: O Autor (2017)

No Quadro 5, são listados os valores dos principais componentes utilizados no desenvolvimento do protótipo. A cotação foi feita em Dólar americano na loja virtual AliExpress.com.

Quadro 5 - Componentes utilizados

Componente	US$
Sensor PIR	0,69
Sensor MQ-5	1,15
Resistor 100	0,01
Resistor 270	0,01
Resistor 330	0,01
Resistor 9.85K	0,01
Resistor 38.4K	0,01
Transistor	0,61
Expansão 40 pinos	2,09
Protoboard	2,31
Kit de Jumper	3,09
Raspberry Pi	46,30

Fonte: O Autor (2017)

No trabalho, optou-se pelo Raspberry Pi 3 Model B por possuir Wi-Fi embutido, que consome pouca energia. Além dos 40 pinos GPIO, disponíveis também em versão anterior do Rpi. Os GPIOs, são necessários para controlar os LEDs de notificação e os sensores que são dois no total conforme listado abaixo:

a) Sensor PIR, responsável pela detecção de movimento;
b) Sensor MQ-5, para detectar vazamento de gás;

Na Figura 22, é possível observar o *setup* da GPIO no Python, por meio da biblioteca *RPi.GPIO* que tem duas maneiras de numerar os pinos *IO* em um Raspberry Pi.

O primeiro é o uso do sistema de numeração *BOARD*. Isso se refere aos números de pinos da placa Rpi. A vantagem de usar este sistema de numeração é que seu software sempre funcionará, independentemente da revisão da placa do Rpi.

O segundo sistema de numeração é o número de BCM. A opção *GPIO.BCM* significa que está referindo-se aos pinos pelo número *Broadcom SoC Channel*. Como neste trabalho está sendo usado apenas uma versão do Raspberry Pi, optou-se por esse sistema.

Figura 22 - Setup GPIO

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO_GAS = 6
GPIO_LED_B = 16
GPIO_BUZ = 20
GPIO_PIR = 21
GPIO_LED_V = 26
GPIO.setup(GPIO_GAS, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(GPIO_LED_B, GPIO.OUT)
GPIO.setup(GPIO_PIR, GPIO.IN)
GPIO.setup(GPIO_BUZ, GPIO.OUT)
GPIO.setup(GPIO_LED_V, GPIO.OUT)

MqttUrl = "mqtt://localhost:1883"
KeepAlive = 60
Qos = 1
```

Fonte: O Autor (2017)

O estado do protótipo é representado pelo LED verde piscando, o qual indica que o dispositivo está ligado e conectado ao servidor MQTT, já o vermelho é ligado por 3 segundos em caso de disparo de um dos sensores, além de emitir um sinal sonoro pelo *buzzer*.

O software desenvolvido para o dispositivo está divido em três classes principais, sendo elas:

a) Class SensorPIR: implementa o controle do sensor de movimento;
b) Class SensorGAS: responsável pelo sensor de gás;
c) Class Mosquito: gerencia as conexões com o MQTT.

Na Figura 23, é exibido o trecho de código que cria uma instância de *thread* para cada classe.

Figura 23 - Inicialização de Thread

```
if Continua:
    print("Ligado")
    #Criar Classe
    Operante = Inicio()
    #Criar thread
    OperaThread = Thread(target=Operante.run)
    #Start tread
    OperaThread.start()
    time.sleep(0.5)

    #Sensor de movimento PIR
    ExecPIR = SensorPIR()
    PIR_Thread = Thread(target=ExecPIR.run)
    PIR_Thread.start()
    time.sleep(0.5)

    #Sensor MQ5 GAS
    ExecGAS = SensorGAS()
    GAS_Thread = Thread(target=ExecGAS.run)
    GAS_Thread.start()
    time.sleep(0.5)

    #MQTT
    ExecMQTT = Mosquito()
    MQTT_Thread = Thread(target=ExecMQTT.run)
    MQTT_Thread.start()
    time.sleep(0.3)
```

Fonte: O Autor (2017)

Na Figura 24, é exibido a classe *SensorPIR* na qual é possível observar como é efetuada a leitura em caso de disparo do sensor, em que o comando *GPIO.input()* recebe o valor do parâmetro *GPIO_PIR* correspondendo a um dos pinos da GPIO definido conforme Figura 22.

Figura 24 - Leitura do sensor PIR

```
print("Aguardando PIR estabilizar...")
while GPIO.input(GPIO_PIR)==1:
    EstadoAtual = 0
print("Pronto...")

while self._running:
    if ContinuaPir:
        EstadoAtual = GPIO.input(GPIO_PIR)
        mqttc.loop_start()
    else:
        EstadoAtual = 0
    if EstadoAtual==1 and EstadoAnterior==0:
        GPIO.output(GPIO_LED_B,GPIO.HIGH)
        ContaPir += 1
        print("Movimento detectado...",time.asctime( time.localtime(time.time()) ))
        print("No. de detecções de movimento ", ContaPir)
        GPIO.output(GPIO_BUZ,GPIO.HIGH)
        time.sleep(1)
        GPIO.output(GPIO_BUZ,GPIO.LOW)
        EstadoAnterior = 1
        time.sleep(1)
```

Fonte: O Autor (2017)

De acordo com o valor passado à variável local *EstadoAtual*, entra na condição em que o comando *GPIO.output()* recebe os parâmetros do pino e nível lógico para acender o LED, e o mesmo acontece com o *buzzer*, para emitir um sinal sonoro.

Na Figura 25, é exibido a classe *SensorGas* que segue a mesma lógica da implementação anterior do sensor PIR;

Figura 25 - Leitura do sensor de gás

```
print("Aguardando Sensor Gas estabilizar...")
while GPIO.input(GPIO_GAS)==1:
    EstadoAtualGas = 0
print("Pronto...")

while self._running:
    if ContinuaGas:
        EstadoAtualGas = GPIO.input(GPIO_GAS)
        mqttc.loop_start()
    else:
        EstadoAtualGas = 0
    if EstadoAtualGas==1 and EstadoAnteriorGas==0:
        GPIO.output(GPIO_LED_B,GPIO.HIGH)
        ContaGas += 1
        print("Gas detectado...",time.asctime( time.localtime(time.time()) ))
        print("No. de detecções de Gas ", ContaGas)
        GPIO.output(GPIO_BUZ,GPIO.HIGH)
        time.sleep(2)
        GPIO.output(GPIO_BUZ,GPIO.LOW)
        EstadoAnteriorGas = 1
        time.sleep(1)
```

Fonte: O Autor (2017)

A implementação em Python é responsável por controlar os sensores, o MQTT e os demais módulos. Além disso, é o ponto inicial, ficando em execução durante o funcionamento do protótipo até que seja desligado.

Na Figura 26, é possível observar como é criada a conexão com o MQTT, para que seja possível enviar as mensagens.

Figura 26 - Método de conexão MQTT

```
mqttc = mosquitto.Client()

# Assign event callbacks
mqttc.on_message = on_message
mqttc.on_connect = on_connect

# Parse CLOUDMQTT_URL (or fallback to localhost)
url_str = os.environ.get('MQTT_URL', MqttUrl )
url = urlparse(url_str)

# Connect
mqttc.connect(url.hostname, url.port, KeepAlive)

rc = 0
while rc == 0 and self._running:
    rc = mqttc.loop()
```

Fonte: O Autor (2017)

Para fazer a implementação do cliente MQTT no Python, utilizou-se a biblioteca Paho. O Paho é uma biblioteca desenvolvida pela Eclipse Foundation e implementa as versões 3.1 e 3.1.1 do protocolo MQTT (ECLIPSE FOUNDATION, 2015).

A Figura 27, abaixo, apresenta a publicação de uma mensagem com o acionamento do sensor.

Figura 27 - Publish MQTT

```
        print("Gas detectado...",time.asctime( time.localtime(time.time()) ))
        print("No. de detecções de Gas ", ContaGas)
        GPIO.output(GPIO_BUZ,GPIO.HIGH)
        time.sleep(2)
        GPIO.output(GPIO_BUZ,GPIO.LOW)
        EstadoAnteriorGas = 1
        time.sleep(1)
        # Publish a message
        mqttc.publish("Sensor/GASEvento", ContaGas, Qos)
    elif EstadoAtualGas==0 and EstadoAnteriorGas==1:
        print("Estabilizando Sensor Gas...")
        EstadoAnteriorGas = 0
        time.sleep(3)
        GPIO.output(GPIO_LED_B,GPIO.LOW)
        print("Sensor Gas ok...")
```

Fonte: O Autor (2017)

O Quadro 6 apresenta os tópicos implementados na aplicação para controlar as funcionalidades dos sensores e a comunicação com a interface do usuário.

Quadro 6 - Tópicos MQTT implementados para o sensor PIR

Tópicos	Funcionalidade
Sensor/PIREvento	Evento do sensor PIR
Sensor/PIRQtd	Quantidade de eventos do sensor PIR
Sensor/PIREnable	Estado do sensor PIR
Sensor/PIRReset	Reseta eventos sensor PIR
Sensor/PIRAtiva	Ativa o sensor PIR
Sensor/PIRDesativa	Desativa o sensor PIR

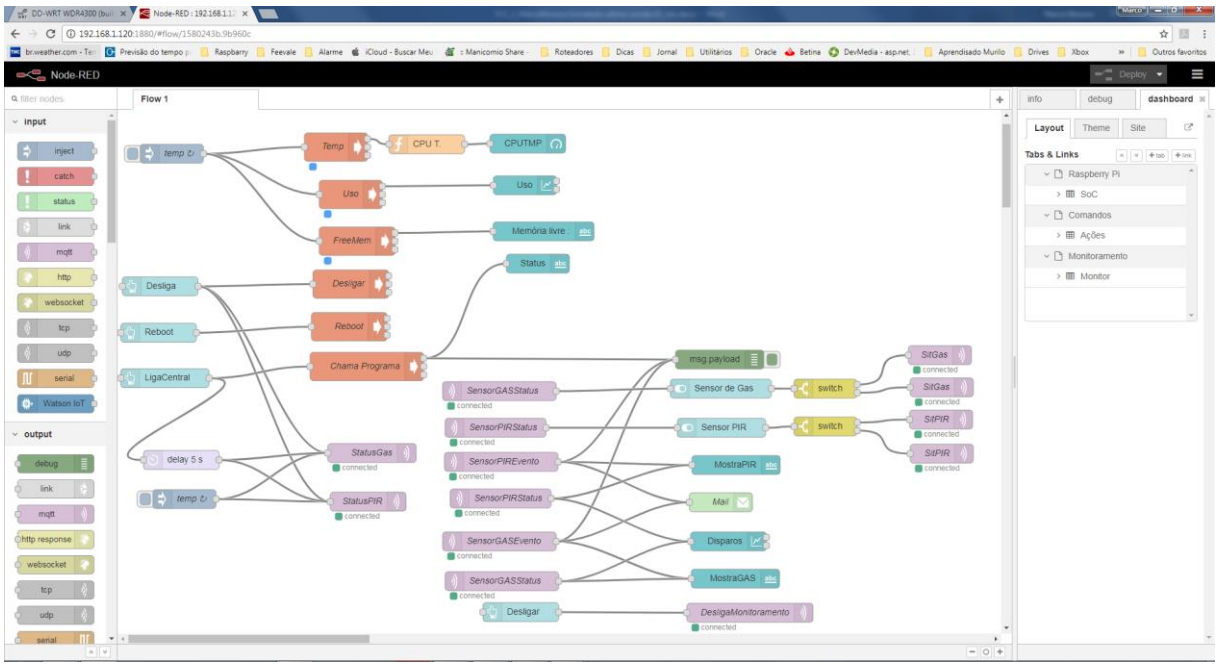
Fonte: O Autor (2017)

2.3.2.2 Interface com o usuário

Para o desenvolvimento da interface do usuário, foi utilizado o Node-RED, que é uma nova ferramenta de software livre, criada pela equipe *IBM Emerging Technology*, que permite desenvolver aplicativos simplesmente conectando nós. Esses nós podem ser dispositivos de hardware, APIs da web ou serviços online.

Na Figura 28 é possível verificar o fluxo do projeto criado.

Figura 28 - Fluxo da interface desenvolvida com Node-RED



Fonte: O Autor (2017)

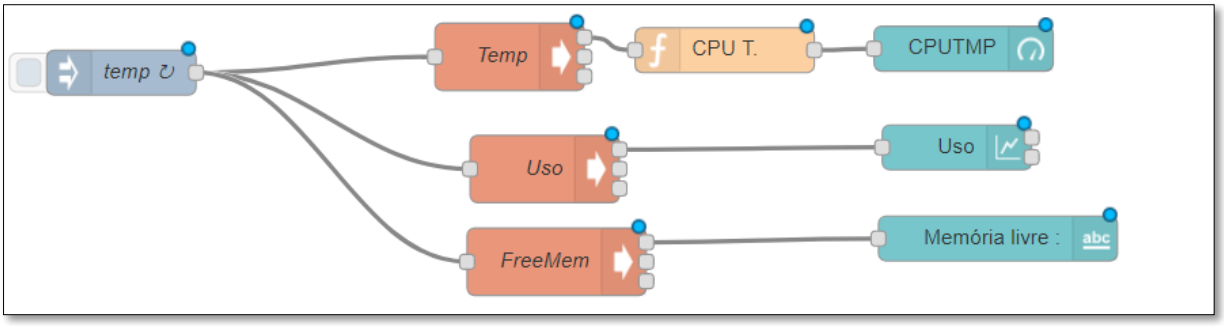
A fim de ilustrar o conceito de *IoT* sugerido, foram criadas funcionalidades em uma aplicação desenvolvida sobre o esquema de integração proposto. A ideia é usufruir do modelo de programação em fluxo para que se possa apresentar e alertar o usuário através da interface criada, envio de e-mail e se necessário, atuar no sistema desligando os sensores (via protocolo MQTT). Essas funcionalidades são apresentadas no Quadro 7.

Quadro 7 - Funcionalidades da Interface do usuário

Funcionalidades da interface implementada
Monitoramento gráfico do Raspberry Pi: Temperatura do SoC, memória Livre, taxa de utilização da CPU.
Reiniciar e desligar a Rpi;
Recebimento de movimento detectado pelo sensor PIR;
Recebimento de gás detectado pelo sensor MQ-5;
Alerta automático por meio de e-mail;
Interface de configuração dos sensores: Ativar, Desativar.
Gráfico com os históricos dos eventos detectados em uma linha de tempo;

É possível analisar as medições de alguns parâmetros de um Raspberry Pi. A Figura 29 apresenta o diagrama de fluxo responsável por esse requisito da aplicação construída:

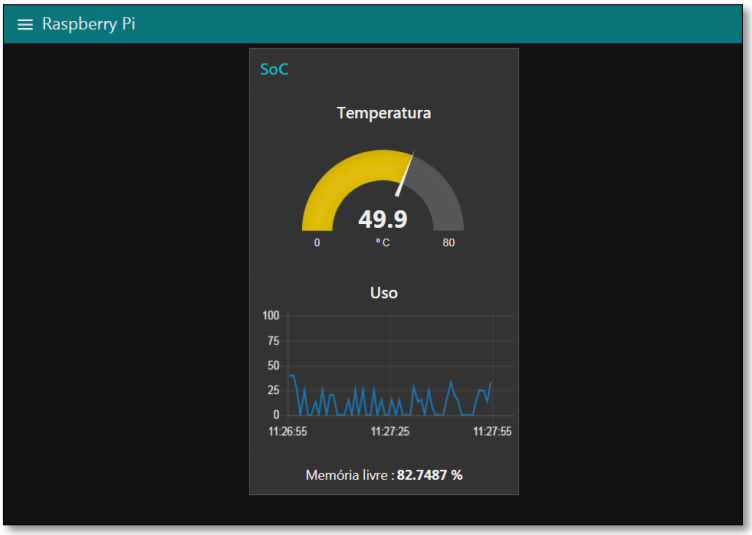
Figura 29 - Fluxo Node-RED de leitura de parâmetros Rpi



Fonte: O Autor (2017)

A Figura 30 ilustra o resultado gerado para o usuário conectado. Essa é a primeira tela apresentada ao conectar na aplicação.

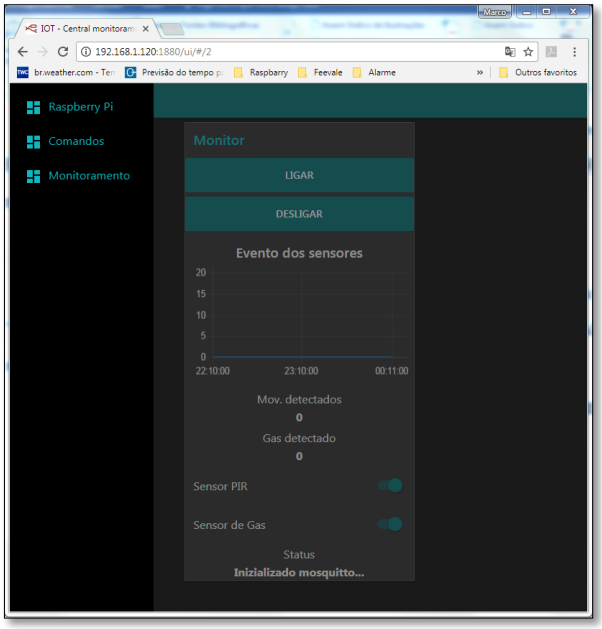
Figura 30 - Interface dos parâmetros Rpi



Fonte: O Autor (2017)

A Figura 31 mostra a interface, com a barra lateral, que apresenta o menu. O primeiro item refere-se aos parâmetros conforme Figura 30, o segundo são os botões para desligar ou resetar o Rpi e o último trata do monitoramento dos sensores.

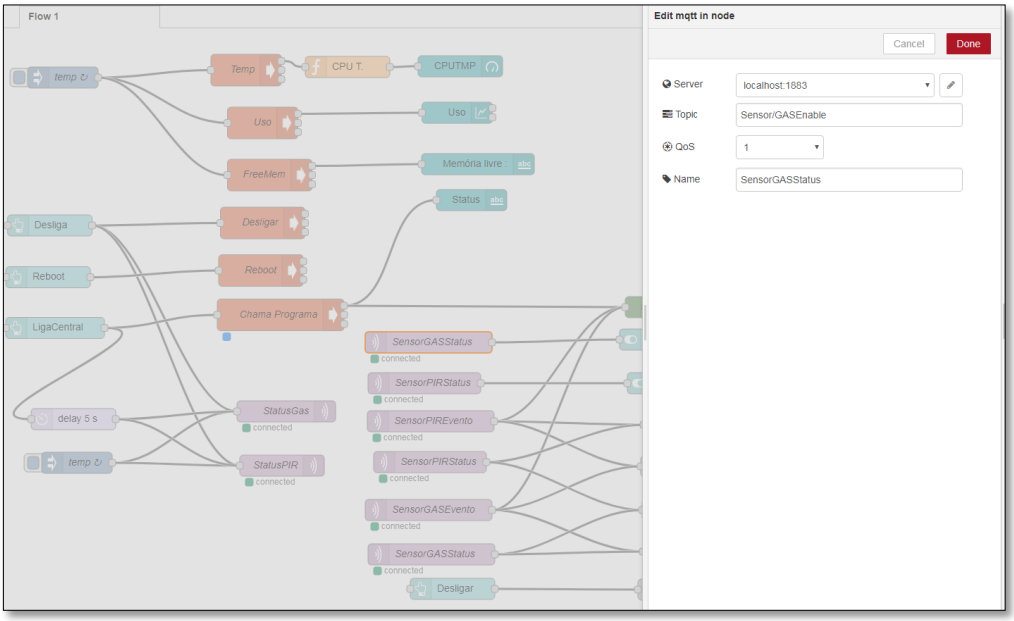
Figura 31 - Menu da interface



Fonte: O Autor (2017)

Na Figura 32, podemos observar a configuração do nó, fazendo a assinatura no tópico “Sensor/GASEnable”, que verifica se o sensor de gás está ativo, passando esse atributo para um *switch* da interface.

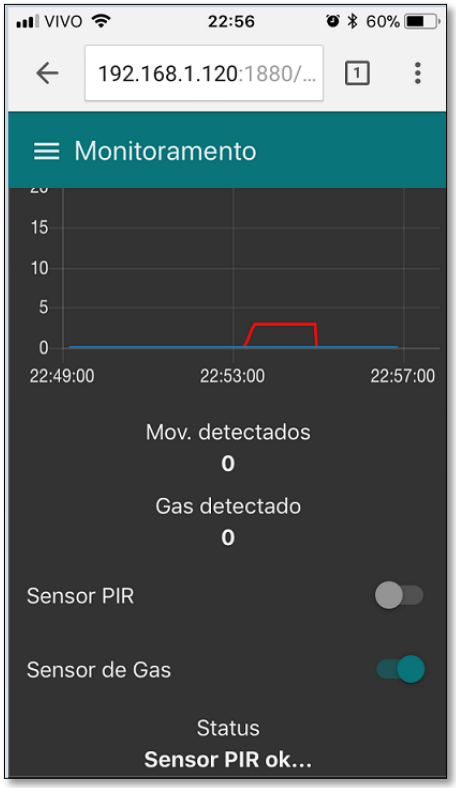
Figura 32 - Assinatura de tópico no Node-RED



Fonte: O Autor (2017)

A Figura 33 mostra a interface sendo executada no Google Chrome, rodando em um dispositivo iOS. Nela podemos verificar o horário e quantidade de eventos de cada sensor, além da opção de ativar ou desativa-los.

Figura 33 - Interface executada em ambiente IOS



Fonte: O Autor (2017)

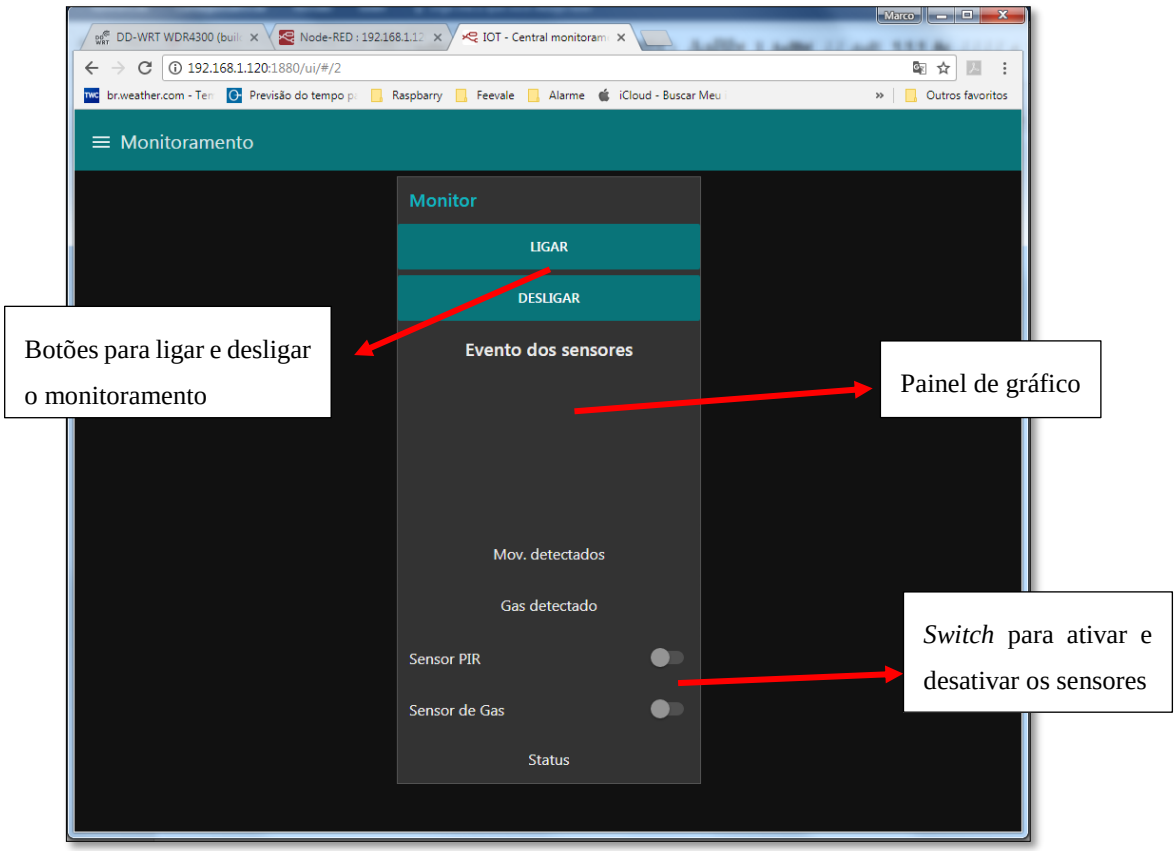
2.4 ANÁLISE DOS RESULTADOS

Durante a implementação do sistema, os testes dos sensores e as classes referentes ao seu controle foram feitos de forma isolada, garantindo, assim, o seu funcionamento a cada iteração concluída.

O teste realizado com o protótipo finalizado, foi uma prova de estresse da aplicação quanto a sua disponibilidade e monitoramento. Para tanto, o protótipo ficou ligado constantemente durante 48 horas. É importante salientar que, durante o desenvolvimento e os testes, o sistema ficou ligado a um no-break, protegendo-o de variações e falta de energia.

Depois desse intervalo, o protótipo manteve suas funcionalidades estáveis, sendo assim, considerado funcional. Na Figura 34, é possível verificar o monitoramento desligado antes do início do teste.

Figura 34 - Tela de monitoramento

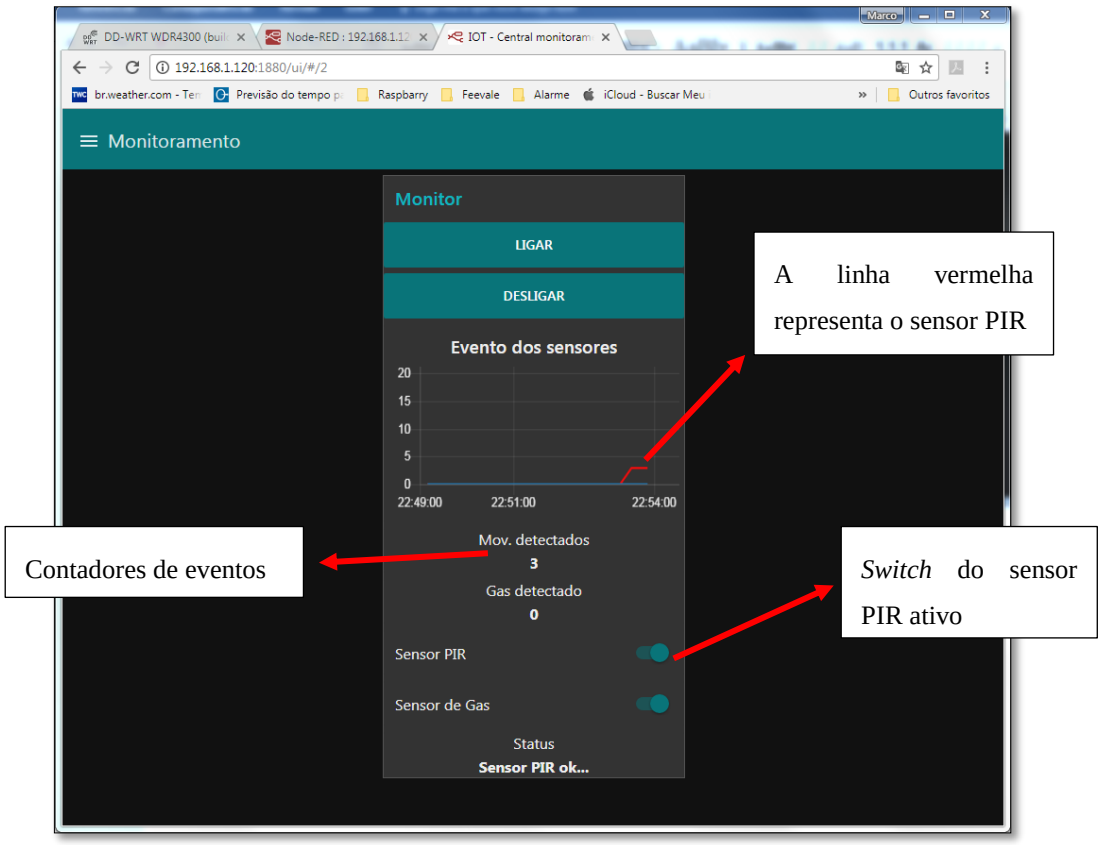


Fonte: O Autor (2017)

Conforme a Figura 34, pode-se verificar os sensores desativados, e os painéis de gráfico de eventos vazio. Por meio dos botões, o monitoramento é ligado e desligado. Aqui é importante salientar uma grande dificuldade no desenvolvimento, pois o sistema precisa executar os comandos enviados pela interface e ao mesmo tempo monitorar as portas GPIO. Como solução foi implementado Threads para os métodos trabalharem em paralelo.

Também foram realizados testes de detecção de presença pelo sensor PIR, em que a interface apresenta histórico e quantidade de eventos, seguidas de envio do e-mail de notificação, conforme Figura 35 e Figura 36 respectivamente.

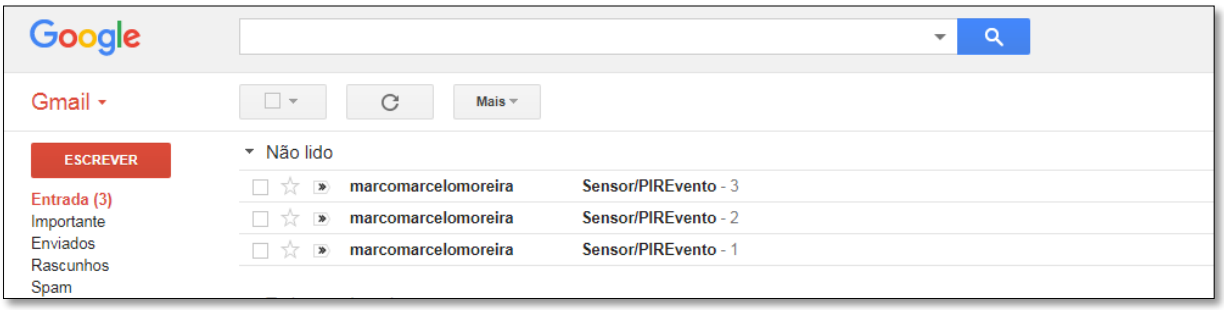
Figura 35 - Monitoramento com movimentos detectados



Fonte: O Autor (2017)

A Figura 35 ilustra os eventos do sensor de movimento em um gráfico e o contador desse evento atribuído a um *label*. O contador pode ser resetado ao desativar o sensor.

Figura 36 - Aviso de detecção recebido por e-mail

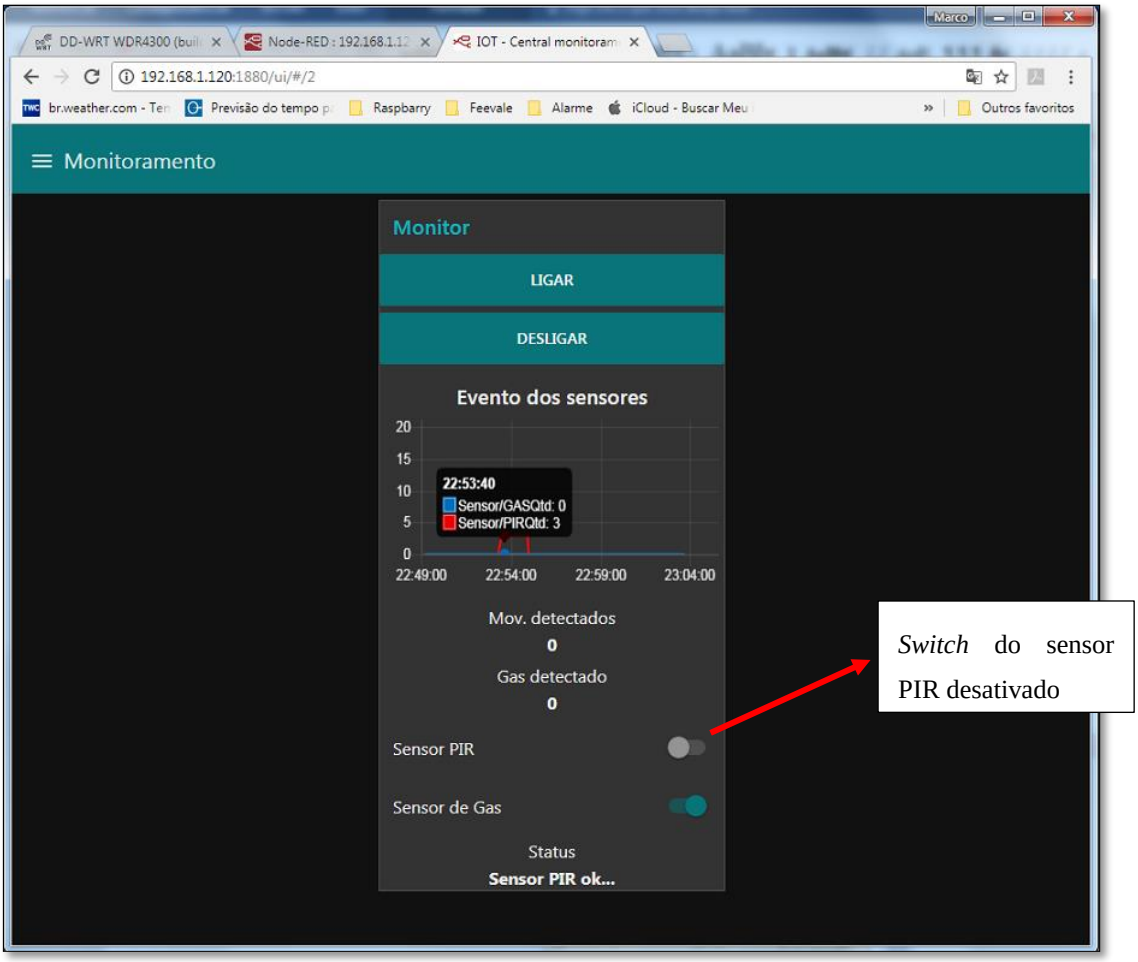


Fonte: O Autor (2017)

A Figura 36 apresenta a caixa de entrada configurada na aplicação para receber os e-mails durante os testes detecção de movimentos.

Outro ponto testado foi a atuação do usuário para desativar ou ativar os sensores remotamente. A Figura 37 ilustra o componente *switch* desabilitado referente ao sensor PIR. Ao desativar o sensor, o sistema reseta o contador, mas mantém o histórico no gráfico.

Figura 37 - Sensor desativado

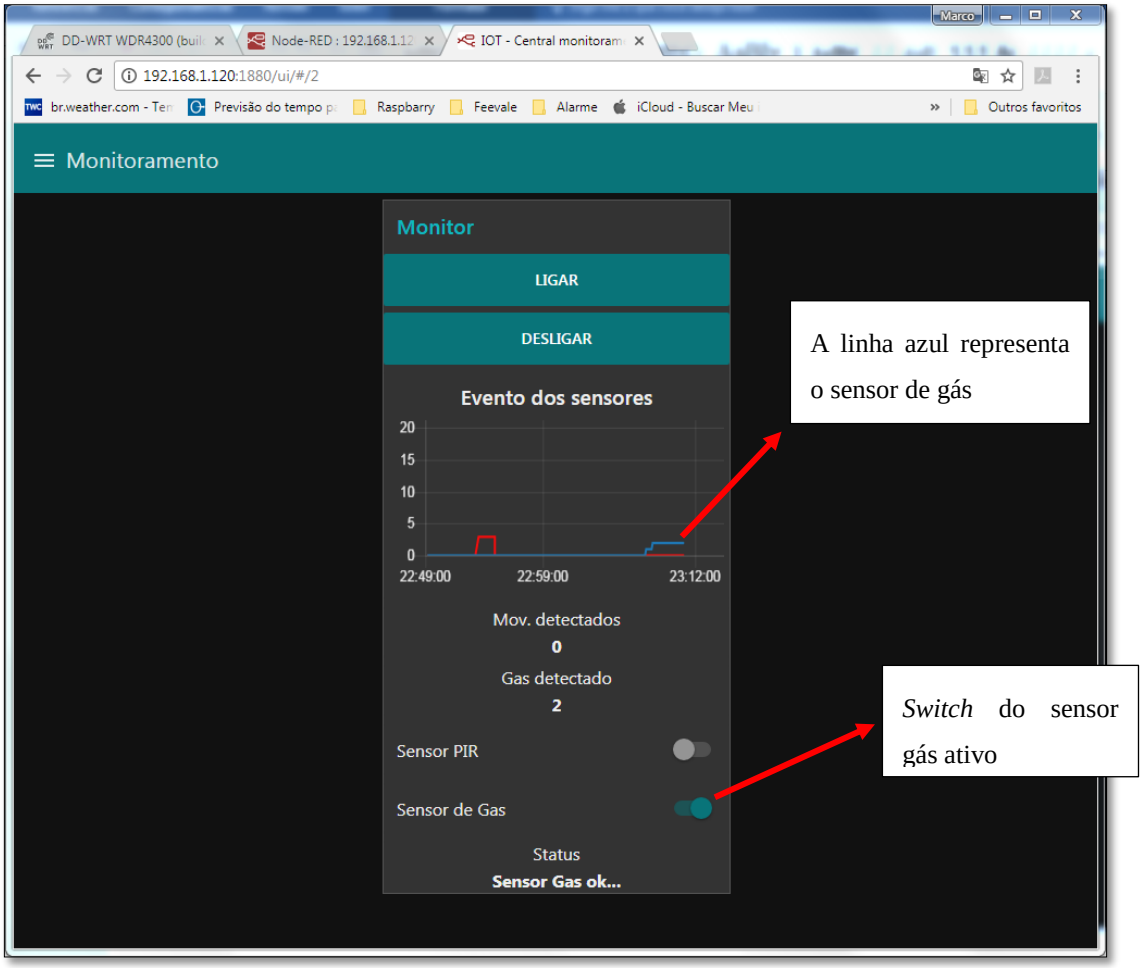


Fonte: O Autor (2017)

Com o sensor de movimento desativado, nenhum evento do mesmo foi detectado. Neste estado apenas o sensor de gás estava monitorando o ambiente.

Foi testado o funcionamento do sensor MQ-5, simulando um vazamento de gás, com o uso de um isqueiro. Na Figura 38 pode-se verificar seus eventos.

Figura 38 - Monitoramento com gás detectado



Fonte: O Autor (2017)

Na Figura 38 mostra a interface com o gráfico, onde a linha azul são os eventos de gás e as vermelhas os movimentos detectados. Podemos visualizar ainda o contador do sensor MQ-5 e seu switch habilitado demonstrando que está ativo.

O uso do Raspberry como plataforma embarcada, mostrou-se eficaz, visto que o hardware não apresentou nenhum tipo de travamento ou erro em seu funcionamento. Por meio do monitoramento implementado, pode-se verificar esses parâmetros na Figura 30 aferindo o seu desempenho. Outra vantagem refere-se a versão do Rpi utilizado, que possui uma interface *wireless* integrada.

A implementação do sensor PIR foi simples, pois sua voltagem de entrada e saída são compatíveis com a GPIO do Rpi, conforme especificação na seção 1.5.4 do referencial teórico. O sensor MQ-5 tem uma grande diferença relacionada a voltagem analógica de saída, onde a mesma pode apresentar até 5v, o que poderia danificar a GPIO. Como solução, foi usado

resitores e um *transitor* para ajustar seu nível lógico segundo o diagrama esquemático na seção 2.2.2 ilustrada pela Figura 18.

O SO Rapsbian possui uma proficiência em atualizações e instalações de biblioteca e funções em seu sistema, trazendo rapidez e praticidade para diferentes tipos de necessidades, como as linguagens de programação já instaladas em sua distribuição.

A linguagem Python simplificou a programação e controle dos sensores por meio da biblioteca *RPi.GPIO*, que possui uma sintaxe simples de ser implementada, conforme pode ser visto no apêndice A.

A utilização do Node-RED no desenvolvimento da interface com o usuário, impressionou pelo nível de abstração apresentada. Foi possível desenvolver um *front-end* com apenas uma página de nós em seu fluxo, conforme demonstrado na Figura 28.

O MQTT contempla características como segurança, qualidade de serviço e facilidade de implementação, sendo simples o seu uso como protocolo em sistemas embarcados. O broker open source Mosquitto, foi facilmente baixado e instalado no sistema operacional da plataforma Raspberry Pi. Adicionalmente, para a parte de criptografia do Mosquitto, existe a necessidade de instalação do pacote openssl, pois não foi implementado neste trabalho.

A partir dos testes executados nesta solução prototipada, verificou-se que uso de IoT na segurança domiciliar é perfeitamente viável, tendo como principal benefício a facilidade de interagir remotamente com os sensores conectados, além disto, esta implementação possui um avanço no interfaceamento com o usuário, proporcionando uma melhor experiência em relação as soluções existentes no mercado que possuem uma interface composta por apenas um teclado numérico.

CONSIDERAÇÕES FINAIS

A maioria dos dispositivos eletrônicos convencionais, usados em casas e prédios, não permite um controle personalizado e remoto (a maiores distâncias), o que pode limitar suas funções e reduzir a comodidade do usuário. Como exemplo disso, temos os alarmes, cercas elétricas e portões eletrônicos, que são acionados apenas com a presença do usuário tendo em mãos o respectivo controle. O uso da *IoT* pode trazer significativas vantagens na área de segurança residencial, com o controle e monitoramento remoto desses dispositivos.

Sendo assim, na primeira parte deste trabalho, procurou-se fazer um apanhado geral, buscando identificar as tecnologias existentes no cenário da *IoT*. Logo após foi realizado um estudo sobre a plataforma Raspberry Pi, em que se buscou quais são as suas principais características técnicas, sistemas operacionais indicados e linguagens de programação suportadas. Na sequência, um estudo sobre os sensores descrevendo brevemente suas configurações de uso.

Após a realização deste projeto, foi aferido, por meio dos testes apresentados no subcapítulo 2.4, que o sistema de segurança domiciliar desenvolvido com a utilização do Raspberry PI, sistemas embarcados e os sensores de movimento e gás se mostrou eficaz, porque o usuário pode monitorar e interagir com o protótipo, local e remotamente, por meio da interface implementada.

É possível destacar, as seguintes respostas a outros objetivos propostos:

O objetivo específico de apropriar-se dos conceitos de Internet das Coisas, foi plenamente alcançado, utilizando o referencial teórico.

A aquisição de conhecimento sobre a plataforma Raspberry Pi e a compreensão do funcionamento dos sensores, também foi plenamente atingida durante o levantamento bibliográfico do trabalho e nos testes efetuados com a protoboard.

A escolha do sistema operacional embarcado foi instalada e testada com sucesso, conforme descrição do ambiente no capítulo 2.

A linguagem de programação escolhida foi Python e a suas bibliotecas foram úteis no controle dos sensores, GPIO e protocolo de comunicação MQTT entre os módulos. Vale ressaltar a grata surpresa com o Node-RED, que, foi definido para implementação da interface com os usuários, pois surpreendeu pela simplicidade e nível de abstração no desenvolvimento.

A contribuição deste trabalho, foi provar de forma satisfatória a utilização do conceito de *IoT* em um sistema de segurança residencial por meio de um protótipo. Contudo, para o efetivo uso do sistema, é imprescindível incluir as funções de segurança do protocolo MQTT e do servidor Node-RED. Eles possuem mecanismos que, embora testados, não foram implementados. Os riscos de permitir a atuação remota sobre a aplicação, são altos e requerem um estudo mais aprofundado desse tema.

Uma proposta de trabalho futuro é expandir o sistema de monitoramento, com a inclusão do servidor Node-RED, em que várias novas funcionalidades podem ser exploradas com os serviços de nuvem oferecidos no Bluemix, entre eles o *Watson IoT Platform*.

Outra proposta de trabalho futuro, como já exposto anteriormente, é utilizar as opções de segurança oferecidas pelo protocolo MQTT, para que o sistema possa ser utilizado com segurança, nos envios e recebimentos de mensagens vitais entre os clientes conectados e o servidor Node-RED, no qual é necessário também o desenvolvimento de um método de autenticação e criptografia na comunicação com o usuário.

REFERÊNCIAS BIBLIOGRÁFICAS

ARANDA. Jorge; **Internet das Coisas: Um protótipo usando a plataforma de prototipagem Arduino e a placa Ehealth para a coleta de sinais vitais**. 2016. 106 f. Trabalho de Conclusão de Curso (Monografia) – Curso Ciências da Computação, Universidade Feevale, Novo Hamburgo, RS, 2016.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. **The Internet of Things: A survey**. 2010. Computer Networks, p. 2787–2805. Disponível em <http://www.science.smith.edu/~jcardell/Courses/EGR328/Readings/IoT%20Survey.pdf>. Acesso em: 06 jun. 2017.

AWS, 2017. **Noções básicas sobre mensagens/sub-mensagens**. Disponível em < https://aws.amazon.com/pt/pub-sub-messaging/>. Acesso em 04 nov. 2017.

BALL, Stuart. **Embedded Microprocessor Systems: Real World Design**, 3rd edition, Editora: MCPros, EUA, 2005.

CHASE, Otavio; ALMEIDA, F. J. **Sistemas embarcados**. 2007. Disponível em <http://www.academia.edu/download/42673850/Embarcados.pdf>. Acesso em: 05 jun. 2017.

DESAI, Pratik. **Python Programming for Arduino.** Birmingham: Packt Publishing Ltd, 2015.

DINIZ, Eduardo H. **Internet das coisas**. GV-executivo, v. 5, n. 1, p. 59, 2006. Disponível em <http://bibliotecadigital.fgv.br/ojs/index.php/gvexecutivo/article/download/34372/33170>. Acesso em: 04 jun. 2017.

FESTO. Sensores. 2010. Disponível em: <http://sensoreslabfisica.blogspot.com.br>. Acesso em: 09 jun. 2017.

HANWEI ELECTRONICS. **Datasheet MQ-5**. s.d. Disponível em: <http://www.usinainfo.com.br/index.php?controller=attachment&id_attachment=146>. Acesso em: 06 jun. 2017.

IBM & EUROTECH, 2010. **Especificação do protocolo MQTT V3.1**. Disponível em <http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#msg-format>. Acesso em 17 set. 2017.

INTERNET OF THINGS IN 2020. **A ROADMAP FOR THE FUTURE.** 2008. Disponível em <http://www.smart-systems-integration.org/public/documents/publications/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v3.pdf>. Acesso em 25 mar. 2017.

ITU - INTERNATIONAL TELECOMMUNICATION UNION. **ITU Internet Reports 2005: The Internet of Things.** Geneva, 2005. Disponível em <http://www.itu.int/osg/spu/publications/internetofthings/. Acesso em 25 mar. 2017.

LIMA, Victor Gutemberg Santos; Dias Wanderson R.A.; Melo José D.; Moreno, Edward D. **Análises de Sistemas Operacionais Linux usando Plataforma Embarcada**. s.d. Disponível em <https://www.researchgate.net/profile/Wanderson_Dias/publication/283018347_Anlises_de_Sistemas_Operacionais_Linux_usando_Plataforma_Embarcada/links/5626c87708aeedae57dc7c62.pdf>. Acesso em 05 jun. 2017.

MARIANO, Felipe Artur. **Integração de tecnologias da Internet das Coisas a um supervisor de energia através da programação orientada à fluxos,** 2016. 69 p. Trabalho de Conclusão de Curso (Monografia) - Instituto Federal de Santa Catarina - IFSC, 2016.

PEREIRA, Renata IS; JUCÁ, Sandro C.S. s.d. **Sistema Embarcado Linux baseado em Raspberry Pi para gravação online de Microcontroladores PIC.** Disponível em <http://www.eripi.ufpi.br/images/anais/140995.pdf>. Acesso em 05 jun. 2017.

PYTHON. **Getting started with python**. 2015. Disponível em: <http://thepythonguru.com/getting-started-with-python/>. Acesso em: 11 jun. 2017.

QUINDERÉ, Patrick R. F.: **Casa Inteligente - Um Protótipo de Sistema de Automação Residencial de Baixo Custo.** Fortaleza, 2009. 69p. Trabalho de Conclusão de Curso (Monografia) – Ciências da Computação, Faculdade Farias Brito, 2009.

RASPBERRY PI PRODUCTS. Disponível em: <https://www.raspberrypi.org/products>. Acesso em: 23 mar. 2017.

SANTOS, Bruno P. et al.s.d. **Internet das Coisas: da Teoria à Prática**. Disponível em <http://homepages.dcc.ufmg.br/~mmvieira/cc/papers/internet-das-coisas.pdf>. Acesso em 05 jun. 2017.

SCHEIDEMANTEL, Felipe Leite. **Monitoramento de vídeo por meio do computador Raspberry Pi.** Brasília, 2015. 56 p. Trabalho de Conclusão de Curso (Monografia) - Faculdade de tecnologia, Universidade de Brasília, 2015.

SIQUEIRA, Frank Augusto et al. **Livro de Minicursos SBRC 2016**. Disponível em <https://www.researchgate.net/profile/Alexandre_Heideker/publication/303810868_Computacao_Urbana_Tecnologias_e_Aplicacoes_para_Cidades_Inteligentes/links/576809b808ae8ec97a423e6b.pdf>. Acesso em: 04 jun. 2017.

SPINOLA, M. **Diretrizes para o desenvolvimento de software de sistemas embarcados**. 1998. Tese. Doutorado. Universidade de São Paulo, SP, 1998.

SUNDMAEKER, Harald et al. Vision and challenges for realising the Internet of Things. **Cluster of European Research Projects on the Internet of Things, European Commision**, 2010. Disponível em <http://www.robvankranenburg.com/sites/default/files/Rob%20van%20Kranenburg/Clusterbook%202009_0.pdf>. Acesso em: 05 jun. 2017.

TAURION, Cezar. **Software Embarcado. A nova onda da informática chips e softwares em todos os objetos.** Rio de Janeiro. Editora Brasport, 2005.

TOOLEY, M. **Circuitos Eletrônicos: Fundamentos e Aplicações**. 3. Ed. Rio de Janeiro: Elsevier, 2007.

TORRES, Andrei BB; ROCHA, Atslands R.; DE SOUZA, José Neuman. **Análise de Desempenho de Brokers MQTT em Sistema de Baixo Custo.** Disponível em <http://www.lbd.dcc.ufmg.br/colecoes/wperformance/2016/009.pdf>. Acesso em 17 set. 2017.

VICENZI, Alexandre. **BUSTRACKER: Sistema de rastreamento para transporte coletivo.** Blumenau, 2015. 61p. Trabalho de Conclusão de Curso (Monografia) – Ciências da Computação, Universidade Regional de Blumenau, 2015.

WENDLING, Marcelo, 2010. **Sensores.** Universidade Estadual Paulista. Disponível em: <http://www2.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/4---sensores-v2.0.pdf >. Acesso em: 06 jun. 2017.

XAVIER, Lucas M.S.; DAVET, Patrıcia T.; YAMIN, Adenauer C., s.d., **ubiMeter: uma Proposta de Instrumento para o Cenário da IoT**. Disponível em <http://www.lbd.dcc.ufmg.br/colecoes/erad-rs/2016/080.pdf>. Acesso em: 04 jun. 2017.

XIA, Feng, *et al*. Internet of Things**. International Journal of Communication Systems.** 25 set. 2012, p.1101.

APÊNDICE A – CLASSES DO MÓDULO DE CONTROLE

```
# -*- coding: utf-8 -*-
#!/usr/bin/python
import RPi.GPIO as GPIO
from threading import Thread
import time
import paho.mqtt.client as mosquitto
import os
from urllib.parse import urlparse

global Desliga
Desliga = False

class Inicio:
    def __init__(self):
        self._running = True

    def terminate(self):
        self._running = False

    def run(self):
        print("Teste do módulo (CTRL-C para sair)")
        while self._running:
            if Continua:
                GPIO.output(GPIO_LED_V,GPIO.HIGH)
                time.sleep(0.5)
            GPIO.output(GPIO_LED_V,GPIO.LOW)
            time.sleep(0.5)

class SensorPIR:
    def __init__(self):
        self._running = True
```

```
def terminate(self):
    self._running = False

def run(self):
    EstadoAtual = 0
    EstadoAnterior = 0
    global ContinuaPir
    ContinuaPir = True
    global ContaPir
    ContaPir = 0

    mqttc = mosquitto.Client()

    # Parse CLOUDMQTT_URL (or fallback to localhost)
    url_str = os.environ.get('MQTT_URL', MqttUrl)
    url = urlparse(url_str)

    # Connect
    mqttc.connect(url.hostname, url.port, KeepAlive)

    print("Aguardando PIR estabilizar...")
    while GPIO.input(GPIO_PIR)==1:
        EstadoAtual = 0
    print("Pronto...")

    while self._running:
        if ContinuaPir:
            EstadoAtual = GPIO.input(GPIO_PIR)
            mqttc.loop_start()
        else:
            EstadoAtual = 0
        if EstadoAtual==1 and EstadoAnterior==0:
```

```
            GPIO.output(GPIO_LED_B,GPIO.HIGH)
            ContaPir += 1
            print("Movimento detectado...",time.asctime( time.localtime(time.time()) ))
            print("No. de detecções de movimento ", ContaPir)
            GPIO.output(GPIO_BUZ,GPIO.HIGH)
            time.sleep(1)
            GPIO.output(GPIO_BUZ,GPIO.LOW)
            EstadoAnterior = 1
            time.sleep(1)
            # Publish a message
            mqttc.publish("Sensor/PIREvento", ContaPir, Qos)
        elif EstadoAtual==0 and EstadoAnterior==1:
            print("Estabilizando PIR...")
            EstadoAnterior = 0
            time.sleep(3)
            GPIO.output(GPIO_LED_B,GPIO.LOW)
            print("Sensor PIR ok...")

class SensorGAS:
    def __init__(self):
        self._running = True

    def terminate(self):
        self._running = False

    def run(self):
        EstadoAtualGas = 0
        EstadoAnteriorGas = 0
        global ContinuaGas
        ContinuaGas = True
        global ContaGas
        ContaGas = 0
```

```
mqttc = mosquitto.Client()


# Parse CLOUDMQTT_URL (or fallback to localhost)
url_str = os.environ.get('MQTT_URL', MqttUrl)
url = urlparse(url_str)

# Connect
mqttc.connect(url.hostname, url.port, KeepAlive)

print("Aguardando Sensor Gas estabilizar...")
while GPIO.input(GPIO_GAS)==1:
   EstadoAtualGas = 0
print("Pronto...")

while self._running:
   if ContinuaGas:
      EstadoAtualGas = GPIO.input(GPIO_GAS)
      mqttc.loop_start()
   else:
      EstadoAtualGas = 0
   if EstadoAtualGas==1 and EstadoAnteriorGas==0:
      GPIO.output(GPIO_LED_B,GPIO.HIGH)
      ContaGas += 1
      print("Gas detectado...",time.asctime( time.localtime(time.time()) ))
      print("No. de detecções de Gas ", ContaGas)
      GPIO.output(GPIO_BUZ,GPIO.HIGH)
      time.sleep(2)
      GPIO.output(GPIO_BUZ,GPIO.LOW)
      EstadoAnteriorGas = 1
      time.sleep(1)
      # Publish a message
      mqttc.publish("Sensor/GASEvento", ContaGas, Qos)
```

```
            elif EstadoAtualGas==0 and EstadoAnteriorGas==1:
                print("Estabilizando Sensor Gas...")
                EstadoAnteriorGas = 0
                time.sleep(3)
                GPIO.output(GPIO_LED_B,GPIO.LOW)
                print("Sensor Gas ok...")

class Mosquito:
    def __init__(self):
        self._running = True

    def terminate(self):
        self._running = False

    def run(self):

        print("Inizializado mosquitto...")

        # Define event callbacks
        def Mostra(Sensor):
            if Sensor == "PIR":
                mqttc.publish("Sensor/PIRQtd", ContaPir, Qos)
                mqttc.publish("Sensor/PIREnable", ContinuaPir, Qos)
            elif Sensor == "GAS":
                mqttc.publish("Sensor/GASQtd", ContaGas, Qos)
                mqttc.publish("Sensor/GASEnable", ContinuaGas, Qos)

        def Zera(Sensor):
            if Sensor == "PIR":
                global ContaPir
                ContaPir = 0
            elif Sensor == "GAS":
                global ContaGas
```

```
        ContaGas = 0

def Inicializa(Sensor, Estado):
    if Sensor == "PIR":
        global ContinuaPir
        ContinuaPir = Estado
        Zera("PIR")
        Mostra("PIR")
    elif Sensor == "GAS":
        global ContinuaGas
        ContinuaGas = Estado
        Zera("GAS")
        Mostra("GAS")

# Define event callbacks
def on_connect(mqttc, obj, flags, rc):
    mqttc.subscribe("Sensor/#", Qos)

def on_message(mqttc, obj, msg):
    if msg.topic == "Sensor/Desliga":
        Inicializa("PIR",False)
        Inicializa("GAS",False)
        global Continua
        Continua = False
        global Desliga
        Desliga = True
    if msg.topic == "Sensor/PIRReset":
        Zera("PIR")
    if msg.topic == "Sensor/PIRAtiva" and ContinuaPir == False:
        Inicializa("PIR",True)
    if msg.topic == "Sensor/PIRDesativa" and ContinuaPir:
        Inicializa("PIR",False)
    if msg.topic == "Sensor/GASReset":
```

```
            Zera("GAS")
        if msg.topic == "Sensor/GASAtiva" and ContinuaGas == False:
            Inicializa("GAS",True)
        if msg.topic == "Sensor/GASDesativa" and ContinuaGas:
            Inicializa("GAS",False)
        if msg.topic == "Sensor/PIRStatus":
            Mostra("PIR")
        if msg.topic == "Sensor/GASStatus":
            Mostra("GAS")

    mqttc = mosquitto.Client()

    # Assign event callbacks
    mqttc.on_message = on_message
    mqttc.on_connect = on_connect

    # Parse CLOUDMQTT_URL (or fallback to localhost)
    url_str = os.environ.get('MQTT_URL', MqttUrl )
    url = urlparse(url_str)

    # Connect
    #mqttc.username_pw_set(url.username, url.password)
    mqttc.connect(url.hostname, url.port, KeepAlive)

    rc = 0
    while rc == 0 and self._running:
        rc = mqttc.loop()

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO_GAS = 6
GPIO_LED_B = 16
GPIO_BUZ = 20
```

```
GPIO_PIR = 21
GPIO_LED_V = 26
GPIO.setup(GPIO_GAS, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(GPIO_LED_B, GPIO.OUT)
GPIO.setup(GPIO_PIR, GPIO.IN)
GPIO.setup(GPIO_BUZ, GPIO.OUT)
GPIO.setup(GPIO_LED_V, GPIO.OUT)

MqttUrl = "mqtt://localhost:1883"
KeepAlive = 60
Qos = 1

print("....")
global Continua
Continua = True

for x in range(0, 4):
    if GPIO.input(GPIO_LED_V)==1:
        Continua = False
    time.sleep(0.2)

if Continua:
    print("Ligado")
    #Criar Classe
    Operante = Inicio()
    #Criar thread
    OperaThread = Thread(target=Operante.run)
    #Start tread
    OperaThread.start()
    time.sleep(0.5)

    #Sensor de movimento PIR
    ExecPIR = SensorPIR()
```

```
    PIR_Thread = Thread(target=ExecPIR.run)
    PIR_Thread.start()
    time.sleep(0.5)

    #Sensor MQ5 GAS
    ExecGAS = SensorGAS()
    GAS_Thread = Thread(target=ExecGAS.run)
    GAS_Thread.start()
    time.sleep(0.5)

    #MQTT
    ExecMQTT = Mosquito()
    MQTT_Thread = Thread(target=ExecMQTT.run)
    MQTT_Thread.start()
    time.sleep(0.3)

else:
    print("Já iniciado")

try:

    while Continua:
        time.sleep(1)
    if Desliga:
        while ContinuaPir or ContinuaGas:
            time.sleep(1)
        ExecMQTT.terminate()
        ExecPIR.terminate()
        ExecGAS.terminate()
        Operante.terminate()
        GPIO.output(GPIO_LED_V,GPIO.LOW)
        GPIO.cleanup()
        print("Desligado...")
```

```
    else:
        print("Ligado...")
        GPIO.output(GPIO_LED_V,GPIO.LOW)
        GPIO.cleanup()

except KeyboardInterrupt:
    ExecPIR.terminate()
    ExecGAS.terminate()
    ExecMQTT.terminate()
    Operante.terminate()
    GPIO.output(GPIO_LED_V,GPIO.LOW)
    print("Saindo...")
    GPIO.cleanup()
except:
    GPIO.output(GPIO_LED_V,GPIO.LOW)
    GPIO.cleanup()
```