



Universidade Estadual Paulista

“Júlio de Mesquita Filho”

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Instituto de Ciência e Tecnologia de Sorocaba

Trabalho de Graduação

Automação Residencial Com Controle Via Nuvem

Matheus do Nascimento Neri

Orientador: Prof. Dr. Galdenoro Botura Junior

Junho de 2022

Sorocaba

UNIVERSIDADE ESTADUAL PAULISTA
INSTITUTO DE CIÊNCIA E TECNOLOGIA DE SOROCABA

AUTOR: Matheus do Nascimento Neri

Automação Residencial Com Controle Via Nuvem

Trabalho de Graduação apresentado ao curso de Engenharia de Controle e Automação na Universidade Estadual Paulista – Instituto de Ciência e Tecnologia de Sorocaba, como requisito para a obtenção de grau de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Dr. Galdenoro Botura Junior

Sorocaba

2022

N445a	Neri, Matheus do Nascimento Automação residencial com controle via nuvem / Matheus do Nascimento Neri. -- Sorocaba, 2022 89 f. : il., tabs. Trabalho de conclusão de curso (Bacharelado - Engenharia de Controle e Automação) - Universidade Estadual Paulista (Unesp), Instituto de Ciência e Tecnologia, Sorocaba Orientador: Galdenoro Botura Junior 1. Automação residencial. 2. Raspberry Pi (Computador). 3. Python (Linguagem de programação de computador). 4. Redes de computadores. I. Título.
-------	--

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Ciência e Tecnologia, Sorocaba. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

AGRADECIMENTOS

Agradeço a Deus.

Aos meus pais por sempre terem me dado o apoio e feito a pressão necessária para que eu estivesse aqui hoje.

À minha namorada, Marina Bento, pelo apoio incondicional e por sempre acreditar em mim, até nos momentos de maior dificuldade.

Agradeço a todos meus familiares e amigos que também me apoiaram, em especial ao meu primo Eduardo por ter feito com que eu refletisse sobre o valor da vida.

Ao Prof. Dr. Galdenoro Botura Junior pela orientação e oportunidade de realizar esse trabalho, bem como pelos ensinamentos transmitidos durante a graduação.

Aos grupos extra-curriculares e colegas por toda a experiência que passei com eles e que contribuíram em minha formação pessoal e profissional.

RESUMO

O desenvolvimento da tecnologia vem disseminando cada vez mais a domótica e tornando-a mais acessível a todos, sendo capaz de automatizar cada atividade de nosso dia a dia e trazendo muito conforto ao possibilitar que tudo isto seja controlado na palma de nossas mãos. O presente Trabalho de Graduação utiliza dos conhecimentos adquiridos durante o curso e introduz as informações necessárias para o desenvolvimento do projeto na área da Automação Residencial utilizando recursos de baixo custo e fácil acesso. Para o funcionamento deste sistema, realizou-se a programação de uma placa Raspberry Pi utilizando a linguagem Python, a qual recebeu, em suas entradas digitais, os dados enviados por cada sensor instalado em uma simulação de cômodo de uma residência e, por meio da programação da placa e um serviço web, permitiu o usuário monitorar e controlar as condições do ambiente, como: iluminação, ventilação e acesso, enviando os devidos sinais digitais às saídas da placa Raspberry Pi. Através dos sinais de onda extraídos em laboratório foi possível medir o tempo de resposta dos relés de controle dos equipamentos e observar que o sistema criado possui uma boa performance e rápida resposta no controle de suas saídas. Mediante os métodos aplicados, foi possível observar como um projeto de domótica tornou-se mais acessível nos dias de hoje.

Palavras-chave: Raspberry Pi, Linguagem Python, Domótica, Automação residencial.

ABSTRACT

The development of technology is spreading even more the concept of domotic and making it more accessible for everyone who has interest in it and being able to automate every activity of our daily routine, bringing comfort and making possible to have it all controlled in our hands. This work destined to present the conclusion of the graduation course, uses concepts learned in the classes from graduation and introduces the necessary information to develop a project in the field of residence automation using easy access and low-cost resources. To make the system work, it was made a Python language program to run on Raspberry pi that received all the data collected from the sensor attached to the digital inputs of the Raspberry board and simulating a room from a house and with the board programming and using a web service, the user was able to monitor and control the conditions of the simulated room as illumination, ventilation and room access sending all the signals to the digital outputs of the Raspberry board. Using equipment available in laboratory it was possible to extract the wave signals and so measure the response time of the equipment control relays and observe that the created system has a good performance and fast response in the control of its outputs. Through the methods used in this work, it was possible to observe how a domotic project became more accessible in the current days.

Keywords: Raspberry Pi. Python language. Domotic, Home Automation.

ÍNDICE DE FIGURAS

Figura 1: Aplicações da domótica.....	12
Figura 2: Placa Raspberry Pi Zero W.....	15
Figura 3: Fotografia do sensor DHT11.....	16
Figura 4: Terminais do sensor DHT11.....	16
Figura 5: Fotografia de um sensor LDR.....	17
Figura 6: Fotografia do Sensor PIR DYP_ME003.....	17
Figura 7: Exemplificação do funcionamento da lente Fresnel.....	18
Figura 8: Módulo RFID com tags.....	18
Figura 9: Módulo de relés.....	19
Figura 10: Fotografia da Trava Elétrica.....	19
Figura 11: Fotografia do sensor Reed Switch.....	20
Figura 12: Fotografia do Display LCD.....	20
Figura 13: Exemplo de uma estrutura de flow NODE-RED.....	22
Figura 14: Exemplificação do funcionamento do protocolo MQTT.....	23
Figura 15: Ambiente residencial a ser simulado.....	24
Figura 16: Tela da Raspberry Pi através do VNC Viewer.....	26
Figura 17: Circuito para implementação do sensor PIR.....	27
Figura 18: Programação em Python para leitura do sensor de presença.....	27
Figura 19: Circuito físico para sensor PIR.....	28
Figura 20: Circuito em funcionamento do sensor PIR.....	28
Figura 21: Programa sendo executado na Raspberry Pi.....	29
Figura 22: Circuito para implementação do sensor LDR.....	30
Figura 23: Programação em Python para leitura do sensor de luminosidade.....	30
Figura 24: Circuito físico para sensor LDR.....	31
Figura 25: Circuito em funcionamento para componente LDR.....	31
Figura 26: Programa sendo executado na Raspberry Pi.....	32
Figura 27: Circuito para implementação do sensor DHT11.....	33
Figura 28: Programação em Python para leitura do sensor DHT11.....	34
Figura 29: Circuito físico para sensor DHT11.....	34
Figura 30: Circuito para sensor DHT11 em funcionamento.....	35
Figura 31: Programa em execução imprimindo a temperatura e umidade na tela.....	35

Figura 32: Circuito para implementação da trava elétrica.....	36
Figura 33: Programação em Python para o controle da trava elétrica.....	36
Figura 34: Circuito implementado para trava elétrica e módulo relé.....	37
Figura 35: Circuito para trava elétrica em funcionamento.....	37
Figura 36: Programa sendo executado e imprimindo estado da trava na tela.....	38
Figura 37: Janela de configuração da Raspberry.....	39
Figura 38: Circuito a ser implementado para o módulo RFID.....	41
Figura 39: Programação em Python para Leitura de cartões no módulo RFID.....	41
Figura 40: Circuito implementado na protoboard para módulo RFID.....	42
Figura 41: Circuito em funcionamento com a leitura RFID.....	42
Figura 42: Executando programa de leitura RFID e imprimindo a leitura dos cartões.....	43
Figura 43: Circuito a ser implementado para o sensor Reed Switch.....	43
Figura 44: Programação em Python para Leitura do sensor Reed Switch.....	44
Figura 45: Circuito implementado na protoboard para sensor reed switch.....	44
Figura 46: Circuito em funcionamento com a leitura sensor reed switch.....	45
Figura 47: Executando programa de leitura do sensor reed switch.....	45
Figura 48: Janela de configuração da Raspberry Pi.....	46
Figura 49: Endereço do display na placa Raspberry Pi.....	47
Figura 50: Circuito a ser implementado para o display 16x2 I2C.....	47
Figura 51: Programação em Python para escrita no display LCD.....	48
Figura 52: Circuito implementado na protoboard para display LCD 16x2 I2C.....	48
Figura 53: Circuito em funcionamento com o display LCD.....	49
Figura 54: Executando programa de escrita no display LCD.....	49
Figura 55: Circuito a ser implementado para o controle da trava elétrica.....	51
Figura 56: Programação em Python para controle da trava elétrica através do protocolo MQTT.....	52
Figura 57: Circuito implementado na protoboard para controle da trava elétrica através da web.....	52
Figura 58: Circuito implementado para o controle via comunicação MQTT.....	53
Figura 59: Executando programa de controle da trava elétrica imprimindo na tela o estado da trava e mensagem enviada via MQTT.....	53
Figura 60: Cliente MQTT executado no notebook em rede com a Raspberry Pi e enviando as mensagens “1” e “0” para acionamento da trava elétrica.....	54

Figura 61: Circuito elétrico implementado para Projeto Final.....	54
Figura 62: Pinagem da Placa Raspberry Pi.....	55
Figura 63: Circuito final implementado na simulação de cômodo residencial.....	56
Figura 64: Notificação de e-mail enviada pela aplicação web desenvolvida.....	57
Figura 65: Estrutura de nós utilizados para construção da aplicação web desktop de controle do sistema de automação.....	60
Figura 66: Estrutura de nós utilizados para construção da aplicação web mobile de controle do sistema de automação.....	61
Figura 67: Aplicação web desktop utilizada para monitoramento e controle do sistema automático.....	62
Figura 68: Aplicação web mobile utilizada para monitoramento e controle do sistema automático.....	62
Figura 69: Terminal Raspberry pi com monitoramento do sistema implementado no projeto.....	63
Figura 70: Sinais de onda de entrada e saída do módulo relé de controle da trava elétrica.....	64
Figura 71: Sinais de onda de entrada e saída do módulo relé de controle da ventilação.....	65
Figura 72: Sinais de onda de entrada e saída do módulo relé de controle da iluminação.....	66

ÍNDICE DE TABELAS

Tabela 1: Ligação dos componentes do projeto na placa Raspberry Pi.....	54
Tabela 2: Módulos utilizados para construção da aplicação web.....	57
Tabela 3: Valores gastos para implementação do projeto.....	67

SUMÁRIO

AGRADECIMENTOS.....	3
RESUMO	4
ABSTRACT	5
ÍNDICE DE FIGURAS.....	6
ÍNDICE DE TABELAS.....	9
1 INTRODUÇÃO	12
2 CONCEITOS FUNDAMENTAIS.....	14
2.1 PLATAFORMAS.....	14
2.1.1 RASPBERRY PI ZERO W.....	15
2.2 SENSOR DHT11.....	15
2.3 SENSOR LUMINOSIDADE	16
2.4 SENSOR DE MOVIMENTO	17
2.5 MÓDULO RFID	18
2.6 MÓDULO DE RELÉS	18
2.7 TRAVA ELÉTRICA.....	19
2.8 SENSOR REED SWITCH.....	19
2.9 DISPLAY LCD COM MÓDULO I2C	20
2.10 NODE-RED.....	20
2.11 PROTOCOLO MQTT.....	22
3 PROPOSIÇÃO	23
4 MATERIAIS E MÉTODOS	24
4.1 SENSOR DE MOVIMENTO PIR.....	26
4.2 SENSOR LDR	29
4.3 SENSOR TEMPERATURA.....	32

4.4 TRAVA ELÉTRICA.....	35
4.5 MÓDULO RFID	38
4.6 SENSOR REED SWITCH.....	43
4.7 DISPLAY LCD I2C	45
4.8 COMUNICAÇÃO MQTT	49
5. RESULTADOS E DISCUSSÃO	54
6. CONCLUSÃO	68
7. REFERÊNCIAS BIBLIOGRÁFICAS	69
8. ANEXOS.....	72
ANEXO A – PROGRAMAÇÃO DO SISTEMA DE AUTOMAÇÃO RESIDENCIAL EM LINGUAGEM PYTHON.....	72

1 INTRODUÇÃO

A automação residencial, também conhecida por domótica, atualmente é alvo de muita procura por parte da sociedade, uma vez que ela, através do uso das tecnologias disponíveis, facilita e torna automática algumas tarefas habituais, podendo assim, otimizar o tempo e proporcionar uma sensação de maior conforto e bem-estar ao usuário. [1]

A palavra domótica, derivada do termo francês “domotique” (“Domus” significa casa e “Imotique” significa automática), é uma tecnologia atribuída à segurança, comunicação, energia e, principalmente, ao conforto. [2]. Utilizando-se de diversos tipos de sensores, atuadores e redes de comunicação, a domótica torna prático e automático processos como, o controle de iluminação de ambientes e a segurança residencial ou/ e comercial, monitorando movimentos em determinados locais ou controlando o acesso de pessoas. Na Figura 1 é visto um exemplo da variedade de aplicações da domótica.

Figura 1: Aplicações da domótica



Fonte [1]

O início da implementação deste tipo de tecnologia se deu na França em torno dos anos de 1980, quando se sentiu a necessidade de controlar as funções de climatização, segurança e iluminação dos edifícios em construção. [2]

Um “boom” na utilização e implementação da domótica nos lares ocorreu dado o surgimento de microcontroladores como Arduino, Raspberry Pi, entre outros e, assim, este tipo de projeto se tornou mais popular e possível, principalmente, através de maneiras mais simples e com melhor custo-benefício. Neste sentido, a domótica procura ser central ao controlar todos

os pontos da casa por meio de microcontroladores e acessar todas as funções de controle por um painel ou tablet numa área comum da casa ou até mesmo pelo smartphone.

As funções de controle em uma casa automatizada variam de acordo com a infraestrutura implementada, o que torna cada projeto único pois, cada residência ou área comercial não é igual a outra, porém, é possível ter um panorama das principais funções a serem controladas com a aplicação da domótica, como por exemplo:

- Iluminação: quais luzes acendem, em qual momento e em qual intensidade.
- Climatização: no controle da climatização se pode controlar qual ambiente terá o ar-condicionado ligado e em qual temperatura será feito o setpoint, podendo este, ser inteligente, realizando o monitoramento da temperatura do ambiente e fazendo o acionamento automático do mesmo.
- Som: home theater e som ambiente controlados pela central.
- Comunicação: também é possível controlar telefone, acesso à internet Wi-Fi e televisores.
- Janelas e cortinas: alguns projetos podem incluir o controle de abertura e fechamento de janelas e cortinas, utilizando-se de motores e atuadores de forma mais inteligente ao reagir à umidade e à luminosidade do ambiente.
- Portas e fechaduras: trancar e destrancar portas é uma função que pode ser automatizada, assim como o controle de acesso por fechaduras biométricas, de senha ou cartão.
- Irrigação: casas com gramados e quintais podem automatizar a irrigação, configurando os horários ou mesmo instalando um sistema que permita detectar a umidade no solo.

Diante do grande crescimento da domótica e suas diversas aplicações, este trabalho de graduação tem como objetivo implementar os conceitos de automação residencial simulando um cômodo de uma residência com o controle de iluminação, ventilação e acesso, através da nuvem e utilizando recursos de baixo custo.

2 CONCEITOS FUNDAMENTAIS

A seguir são apresentados os tópicos que foram abordados para a implementação deste trabalho de conclusão de curso:

2.1 PLATAFORMAS

Neste projeto será utilizado apenas uma plataforma: Raspberry Pi.

O Raspberry Pi é um dos muitos hardwares encontrados atualmente, capazes de exercer a função de um minicomputador, possuindo as mesmas funções de um computador desktop ou notebook. Entretanto, utiliza microcomponentes, o que permite que o dispositivo seja compacto e, observando a versão Raspberry Pi Zero W/ 2 W é possível comparar o controlador à um cartão de crédito, por exemplo.

Sua criação se deu no Reino Unido pela Fundação Raspberry Pi, empresa criada por um grupo de cientistas do Laboratório de Computação da Universidade de Cambridge, o qual buscou investir na acessibilidade dos conhecimentos da ciência da computação ao compartilhar seus conhecimentos com escolas e universidades na região da Europa. Hoje, conhecida mundialmente, a Fundação Raspberry Pi possui um banco de recursos com acesso gratuito, sobre o desenvolvimento e aprendizado de suas placas, além de compartilhar diversos projetos com o passo a passo de como utilizar seus hardwares para projetos pessoais. O primeiro protótipo desse projeto foi lançado em 2012, nomeado de Raspberry Pi 1 Model B, o qual custava US\$ 25. Atualmente, é possível adquirir a placa Raspberry Pi, conectar periféricos e utilizá-la como um computador Desktop, apesar de possuir algumas limitações de performance. [12]

A utilização destes hardwares não é destinada ao suporte de programas, os quais requisitam muito do processador, mas são capazes de automatizar tarefas, controlar entradas e saídas digitais, permitindo assim, uma diversa gama de projetos. Hoje, o ramo que mais se beneficia com esta tecnologia é a automação residencial, pois um sistema de automação utilizando a placa Raspberry Pi permite o controle de várias rotinas e processos no domicílio.

Em 2019 foi possível instalar uma versão dedicada a dispositivos móveis, a qual permitiu utilizar o Windows 10 em um Raspberry Pi, sendo que, normalmente, estes

minicomputadores possuem sistemas personalizados, baseados em Linux. Esta ferramenta também permitiu a utilização de controladores de robôs, projetos de IoT(Internet of Things), entre outras utilidades. [12]

2.1.1 RASPBERRY PI ZERO W

A Raspberry Pi Zero W Anatel é uma placa de baixo custo, com tamanho reduzido que permite a criação de projetos com a mesma facilidade que os demais modelos mais potentes da Raspberry. Na Figura 2 temos uma imagem da placa cujo tamanho se assemelha ao de um cartão de crédito.

Figura 2: Placa Raspberry Pi Zero W



Fonte [3]

A placa possui Wi-Fi e Bluetooth integrados. O seu processador é o Broadcom BCM2835 single-core de 1GHz e uma memória de 512MB, o que permite a criação de projetos de IoT, automação residencial, monitoramento de processos e aplicações industriais, nos quais é possível controlar a GPIO de 40 pinos.

A Raspberry Pi Zero W Anatel tem slot para cartão micro SD, conector de vídeo mini HDMI, 2 portas USB (1 para dados e outra para alimentação 5V) e roda diversas distribuições Linux como o Raspbian e Ubuntu. [19]

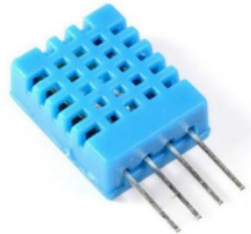
A certificação por parte da Anatel (Agência Nacional de Telecomunicações) garante a qualidade do hardware e ainda permite que seja utilizado em projetos comercializáveis.

2.2 SENSOR DHT11

O componente DHT11 trata-se de um sensor de temperatura e umidade muito utilizado em projetos que contam com o Arduino. A faixa de medição da temperatura é de 0 a 50 graus

Celsius, enquanto a de umidade varia entre 20 e 90%. Na Figura 3 é possível visualizar a fotografia do sensor.

Figura 3: Fotografia do sensor DHT11

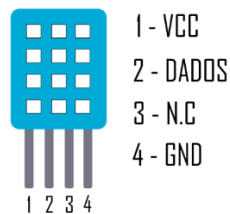


Fonte [5]

A aferição da temperatura é dada através do termistor tipo NTC, ou seja, tem o seu valor de resistência aumentado conforme o aumento de temperatura. Ressalta-se, ainda, que a aferição da umidade se faz através do sensor tipo HR202.

Este sensor pode ser alimentado por uma tensão de 3 a 5 V, apresentando corrente entre 200 μ A e 500 mA, e em stand-by de 100 μ A até 150 μ A. Na Figura 4 encontra-se o esquema dos terminais do sensor.

Figura 4: Terminais do sensor DHT11



Fonte [5]

2.3 SENSOR LUMINOSIDADE

O sensor LDR, “Light Dependence Resistor”, é um resistor cuja resistência varia de acordo com a luminosidade sobre ele. Na Figura 5 encontra-se a sua fotografia.

Figura 5: Fotografia de um sensor LDR



Fonte [6]

O sensor em questão apresenta tensão máxima de 150 Vdc, faixa de temperatura de operação entre -30°C e 70°C, potência máxima de 100 mW, resistência no escuro (0 lux) de 1 M Ω e resistência na luz (10 lux) entre 10 e 20 K Ω .

2.4 SENSOR DE MOVIMENTO

Os sensores de presença mais comuns usam os tipos PIR (Passive Infrared Sensor, ou Sensor Infravermelho Passivo) como, por exemplo, o detector de movimentos. O sensor PIR DYP-ME003 consegue detectar os movimentos de objetos que estejam em uma área de até 7 metros, com um tempo de estabilidade que pode variar entre 5 a 200 segundos. Na Figura 6 temos a demonstração do sensor PIR.

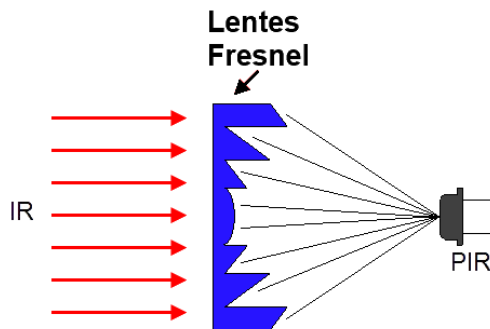
Figura 6: Fotografia do sensor PIR DYP_ME003



Fonte [7]

Este sensor possui duas faixas de infravermelho e é acionado quando há variação na detecção de infravermelho dessas faixas. Na parte externa do sensor, onde vemos uma tampa branca, como na Figura 6, é na realidade uma lente Fresnel, cujo objetivo é ampliar o campo de detecção do sensor. Na Figura 7, podemos ver a representação de como a lente Fresnel aumenta seu campo de detecção.

Figura 7: Exemplificação do funcionamento da lente Fresnel



Fonte [8]

2.5 MÓDULO RFID

O módulo leitor RFID baseado no chip MFRC522 é utilizado em comunicação sem contato a uma frequência de 13,56MHz. Ele permite, sem contato, ler e escrever em cartões os quais seguem o padrão Mifare. As tags RFID, podem conter vários dados sobre o proprietário do cartão conforme elas sejam projetadas, e, permitem, por exemplo, o controle de acesso a determinados cômodos ou locais. Na Figura 8 temos a fotografia do módulo RFID.

Figura 8: Fotografia do módulo RFID



Fonte [9]

2.6 MÓDULO DE RELÉS

A fim de controlar os processos de sensoriamento da temperatura, acesso e iluminação, foi utilizado um módulo com 4 relés. Este é uma ótima ferramenta para o acionamento de cargas diversas, utilizando sinais lógicos de 5V. Ele permite que uma alta gama de microcontroladores como Arduino, PIC, ARM, e microcomputadores como Raspberry Pi acionem dispositivos com tensões maiores, como motores e lâmpadas.

O relé trabalha com uma corrente típica de operação entre 15 a 20mA, possuindo um LED de indicador de status para cada relé, que tem conexões para normal aberto, normal fechado e comum. O tempo de resposta é de cerca de 5 ms, e os dispositivos de saída podem ser de até 10A, com 30V DC ou 220V AC. Na Figura 9 é visto uma imagem do módulo de relés.

Figura 9: Módulo de relés



Fonte [10]

2.7 TRAVA ELÉTRICA

Feita em aço inoxidável, esta trava elétrica funciona com tensão de 12V e tem consumo de corrente de aproximadamente 600mA. A trava elétrica solenoide funciona no conceito normal aberto, ou seja, quando não há tensão aplicada ele se mantém no estado normal e quando há tensão ao fechar o contato, ele é recolhido. Na Figura 10 a trava elétrica é apresentada em uma fotografia.

Figura 10: Fotografia da Trava elétrica



Fonte [11]

2.8 SENSOR REED SWITCH

O sensor *reed switch* é um sensor que atua como chave magnética com funcionamento por campo magnético, fechando os contatos internos quando um ímã é aproximado do sensor.

Funciona com tensão máxima de 200V e tem consumo de corrente de aproximadamente 500mA. Na Figura 11 é visto uma fotografia do sensor.

Figura 11: Fotografia do sensor Reed Switch



Fonte [22]

2.9 DISPLAY LCD COM MÓDULO I2C

O display LCD I2C é um display LCD comum de 16 colunas por 2 linhas com um módulo soldado que realiza a conexão entre o sinal I2C enviado pela placa Raspberry Pi e os pinos necessários para conexão do display. Este display opera em tensão de 5V e permite o ajuste de contraste por um potenciômetro. Na Figura 12 o display LCD é demonstrado em uma fotografia.

Figura 12: Fotografia do Display LCD



Fonte [23]

2.10 NODE-RED

O Node-RED é uma ferramenta de programação visual de código aberto, inicialmente desenvolvida para implementar, criar e/ou conectar dispositivos de IoT, expandindo seu campo de aplicação para hardwares, APIs e web services.

Este recurso por meio dos “nodes” ou nós, é capaz de ler arquivos CSV, escutar eventos http, tcp, websocket, twitter, mqtt, entre outros. O Node-RED possui um editor baseado em um

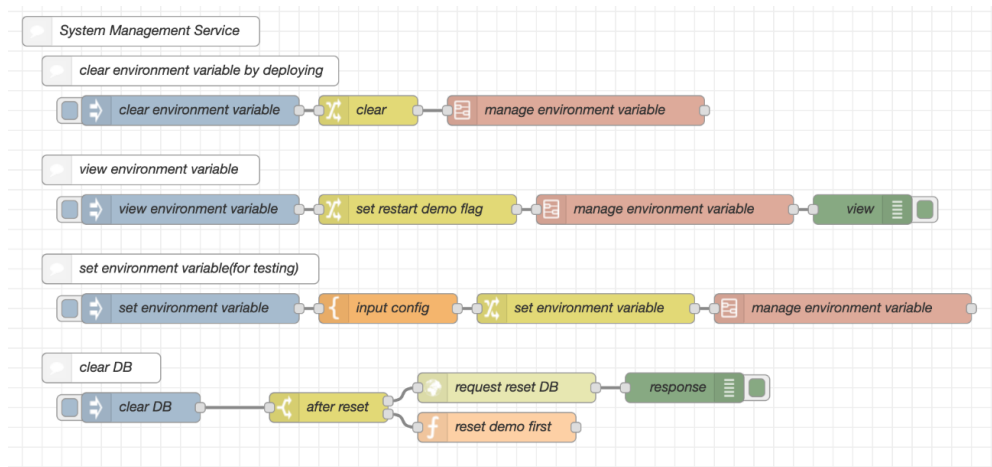
navegador simples e compatível com todos os browsers, o que facilita a conexão de fluxos usando os nós do palette. Assim como o protocolo MQTT (Message Queue Telemetry Transport), que será abordado no próximo tópico, o Node-RED também foi criado pela IBM Emerging Technology. [14]

Entre as vantagens desta ferramenta, destaca-se a possibilidade de poder ser desenvolvido em qualquer Sistema Operacional, contudo para que seja possível utilizá-lo, se faz necessário a instalação do Node.js.

O Node.js é um “node” pré compilado, o qual permite a programação em JavaScript, dentro do editor usando um rich text. Assim, é possível fazer com que a aplicação execute exatamente o que se deseja. O Node-RED possui cerca de 40 nodes, os quais podem simplesmente ser arrastados para o diagrama de fluxo da aplicação, sendo, então, configurados e conectados a outros nodes de maneira rápida e fácil. Vale ressaltar que eles podem ser criados ou adicionados a partir das bibliotecas existentes nas comunidades. [13] [17]

O Node-RED conta com bibliotecas de nós, os quais podem ser adicionados como, nodes para clouds, para sensores, dashboards e IoT. Muitos dos já existentes IoT foram baseados no IBM IoT Foundation, entre os quais destacam-se Input/Outputs: TCP, UDP, HTTP, serial, inject, debug, notify; funções: delays, switch, change, trigger, json, xml; acesso as mídias sociais: twitter, e-mail, irc, facebook; banco de dados: fileaccess, mongodb, mssql, mysql e os nodes de configuração: serialport, mqtt, twitter entre outros. Além dos nodes de comunicação com as entradas e saídas da plataforma Raspberry Pi, e daqueles que permitem a comunicação via protocolo MQTT, ferramenta esta, que será utilizada neste trabalho. Na Figura 13 temos um exemplo de construção de um fluxo de módulos desenvolvido no Node-RED.

Figura 13: Exemplo de uma estrutura de flow NODE-RED



Fonte [18]

2.11 PROTOCOLO MQTT

O protocolo de comunicação MQTT (Message Queuing Telemetry Transport) foi desenvolvido pela IBM no fim dos anos 1990. Seu propósito inicial foi vincular sensores, em pipelines de petróleo, a satélites.

Por ser um protocolo de mensagens com suporte para a comunicação assíncrona entre as partes, ele define dois tipos de sujeitos na rede: um servidor e um cliente. No caso do servidor, também denominado de Broker, ele receberá todos os envios de dados publicados pelos clientes na rede e, em seguida, trabalhará com essas mensagens a fim de destiná-las às suas funções ideais, seja o encaminhamento para outros clientes localizados na mesma rede, como também, para o controle e gestão dos dados recebidos entre outras aplicações.

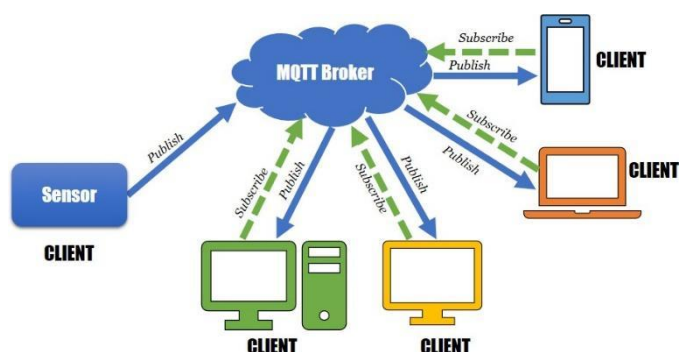
Já o cliente é qualquer dispositivo que possa interagir com o Broker e publicar ou receber mensagens, pode-se elencar como exemplo: um sensor de umidade instalado no ambiente ou uma aplicação que utiliza esses dados do sensor.

Este protocolo de comunicação, baseado na troca de mensagens entre o servidor (Broker) e o cliente, funciona através do modelo de publicação/assinatura (do inglês publish/subscribe). [20]

Broker é o dispositivo cuja função será receber todas as mensagens do publicador, as implementando no sistema e realizando toda a gestão da informação recebida. Porém ao mesmo tempo que o Broker recebe uma mensagem “assinando” um tópico de mensagem, quando ele

envia as informações para outro assinante ele se torna publicador (Publisher). Na Figura 14 é representada a exemplificação da forma como o protocolo MQTT funciona. A placa Raspberry Pi, configurada corretamente, pode atuar como um Broker, o qual recebe os sinais de leitura dos sensores do ambiente e, por meio do protocolo MQTT consegue publicar estas informações em um serviço Web através de mensagens. Uma aplicação web pode ser configurada como assinante destes dados. [16]

Figura 14: Exemplificação do funcionamento do protocolo MQTT



Fonte: Autoria Própria

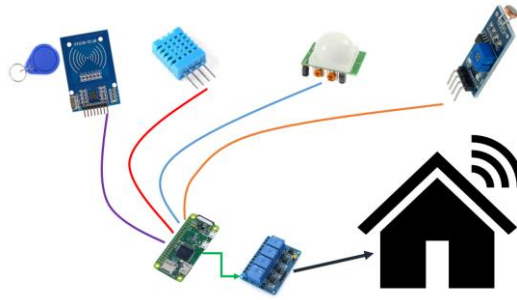
3 PROPOSIÇÃO

A automação residencial é o tópico mais popular, em consequência do crescente desenvolvimento das placas eletrônicas como Arduino, Raspberry Pi, Beaglebone, entre outras. Desta forma, o presente trabalho se propõe a simular um cômodo de uma residência com a finalidade de automatizar e controlar os processos de iluminação, ventilação e acesso.

Para a concretização do trabalho, é necessária a utilização de uma placa eletrônica com a capacidade de ser programada, por isso a escolhida é a Raspberry Pi Zero W. Em relação ao controle dos processos propostos, são utilizados os sensores de presença (sensor PIR), módulo sensor LDR, sensor de umidade DHT11 e sensor magnético *reed switch*. Aliado aos sensores, utiliza-se o módulo RFID para a leitura de “tags”, a fim de realizar o controle de acesso ao cômodo. O ambiente simulado é apresentado pelo esboço da Figura 15 abaixo.

Este controle e monitoramento é realizado por um serviço Web que, através da comunicação MQTT, realiza o controle do ambiente ao selecionar o modo de operação do programa permitindo o controle manual da iluminação, ventilação e acesso do cômodo.

Figura 15: Ambiente residencial a ser simulado



Fonte: Autoria Própria

4 MATERIAIS E MÉTODOS

Para que o projeto se desenvolvesse completamente e utilizando a linguagem Python de programação, se fez necessário o estudo da plataforma Raspberry Pi e a própria linguagem de programação.

Ao iniciar o projeto, o primeiro passo foi preparar o cartão SD que tem por responsabilidade suportar o sistema operacional da placa Raspberry Pi. Na seção de downloads do site oficial do Raspberry Pi foi baixado a ferramenta **Noobs – Offline and network install**. A ferramenta em questão é um pacote que contém diversas versões do sistema operacional com uma interface que auxilia no processo de instalação. Ao finalizar o download, foram extraídos os arquivos compactados e colocados no cartão SD. Desta forma, ao inicializar a placa Raspberry Pi com o cartão SD, na tela do monitor conectada a placa foram mostradas as versões do sistema operacional disponíveis para instalação. Para este projeto foi selecionada a versão **Raspberry Pi OS Full (32-bit)**, que é a versão recomendada pela ferramenta.

Com o sistema operacional em pleno funcionamento, foram estudados os principais comandos do sistema operacional Linux, ambiente utilizado pelo Raspberry Pi. O resultado destes estudos está descrito a seguir.

Toda operação que necessita de permissão do proprietário para ser executada é utilizado o comando **sudo** antes de qualquer outro comando, garantindo assim a permissão para que o usuário realize suas atividades no terminal de comandos.

Na criação de diretórios é utilizado o comando **mkdir**, e na transição entres os diretórios existentes na memória do controlador utiliza-se o comando **cd**. Exemplificando os comandos:

sudo mkdir tcc (Cria o diretório “tcc”)

cd tcc (Abre o diretório “tcc” e passa a trabalhar dentro dele)

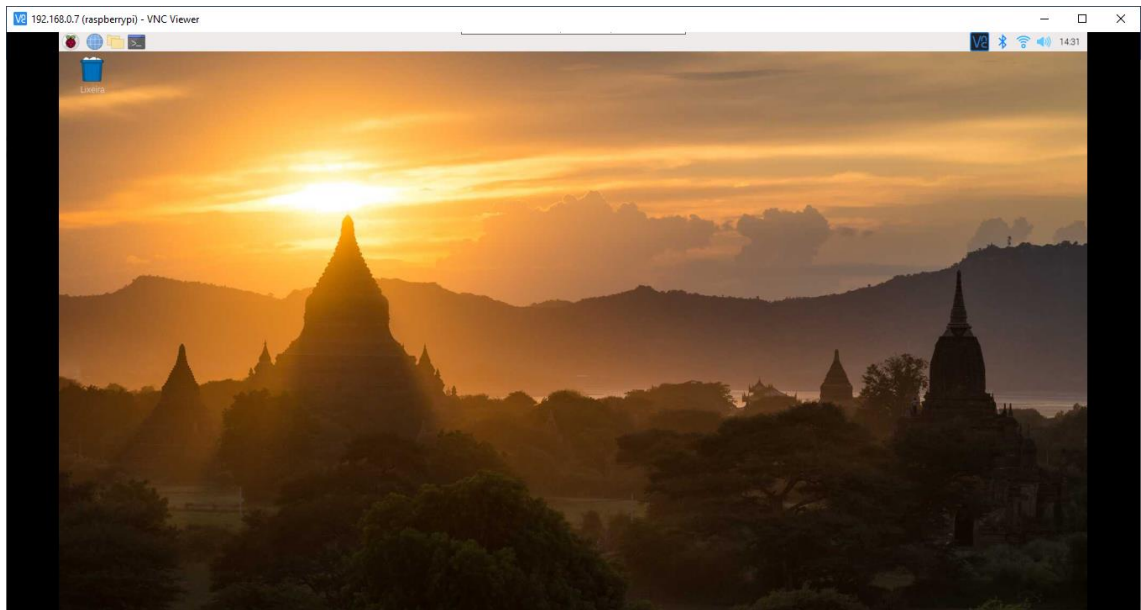
Para listar os arquivos existentes dentro de um diretório temos o comando ***ls***.

Muitos dos tópicos propostos neste trabalho precisam de pacotes extras para garantir a funcionalidade na plataforma Raspberry, por isso, foi muito utilizado o comando ***apt-get install***, o que garantiu que cem por cento das aplicações fossem implementadas e funcionassem da maneira correta.

Para a criação e edição dos arquivos em Python, e para a programação, foi utilizado o comando ***nano <nome arquivo>.py***, o qual abre o editor de texto na versão Python mais recente, e, caso não haja no diretório em questão um arquivo com o mesmo nome digitado no terminal de comando, será criado um arquivo em branco.

Concluída a primeira fase, iniciou-se o uso da plataforma Raspberry, a qual foi utilizada como um servidor VNC, ou seja, pode ser operada através de outro equipamento ligado na mesma rede, podendo assim, realizar a programação dentro de seu ambiente e usufruir das ferramentas do Windows para facilitar a obtenção de dados documentais úteis ao progresso deste projeto, além de não ter gastos com periféricos a parte para utilização do controlador, já que se utiliza apenas um notebook. Na figura 16 é representada a tela da Raspberry Pi através do VNC Viewer.

Figura 16: Tela da Raspberry Pi através do VNC Viewer



Fonte: Autoria própria

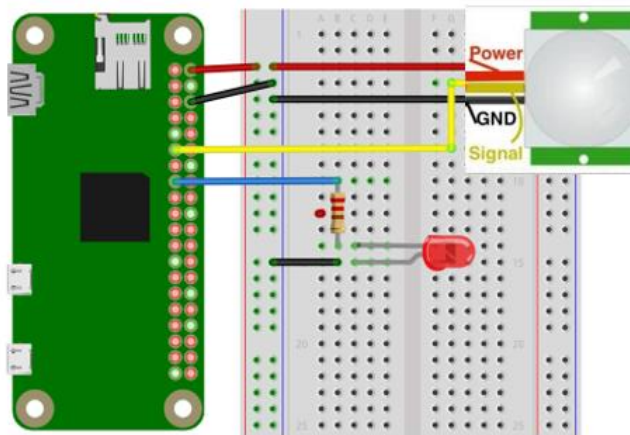
Criado o diretório “tcc” para organização de todos os arquivos a serem utilizados, iniciamos a programação e aplicação de cada componente com programas básicos para adquirir o conhecimento referente a linguagem Python e testar o funcionamento de cada sensor a ser implementado. Os resultados são apresentados a seguir.

4.1 SENSOR DE MOVIMENTO PIR

Pondo em prática os conhecimentos adquiridos com a realização do teste de funcionamento do sensor de presença PIR, foi implantado o circuito mostrado na Figura 17.

O pino 2, que tem como saída a 5V, foi ligado na alimentação do sensor PIR, enquanto o pino 6, cuja saída GROUND foi ligada ao protoboard, fornece a função ground tanto para o sensor PIR como para o LED, o qual indicará, visualmente, a leitura de movimento realizada pelo sensor. É importante salientar que o sinal do sensor PIR foi conectado no pino 11 da GPIO, a qual foi configurada pela programação como entrada, enquanto o LED foi ligado ao pino 15 da GPIO, configurado como saída.

Figura 17: Circuito para implementação do sensor PIR



Fonte: Autoria própria

O programa em Python para a leitura do sensor e visualização através do LED é visto na Figura 18.

Figura 18: Programação em Python para leitura do sensor de presença

```
pi@raspberrypi: ~/tcc
Arquivo Editar Abas Ajuda
GNU nano 3.2 pir_sensor.py
#!/usr/bin/env python

import RPi.GPIO as GPIO # Importa a biblioteca referente aos pinos da placa Raspberry
import time # Importa a biblioteca tempo para utilização da função sleep
import sys # Importa biblioteca que acessa funções usadas pelo próprio interpretador Python

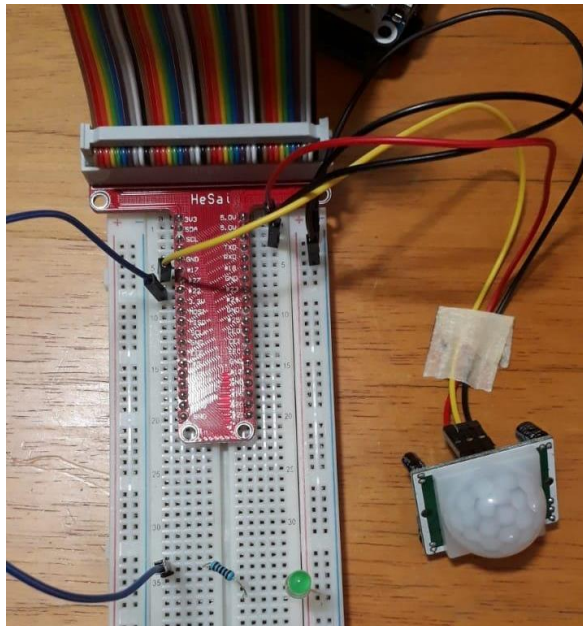
GPIO.setwarnings(False) #Set para ignorar qualquer aviso referente as portas GPIO da placa
GPIO.setmode(GPIO.BOARD) #Seta o modo de utilização para a pinagem da GPIO
GPIO.setup(11, GPIO.IN) #Define o pino 11 da GPIO como pino de entrada
GPIO.setup(15, GPIO.OUT) #Define o pino 15 da GPIO como pino de saída
GPIO.output(15, GPIO.LOW) #Seta nível lógico baixo no pino 15 para que o LED inicie o programa apagado

#Define programa principal
try:
    while True:
        if (GPIO.input(11) == 0): #Enquanto nao houver leitura no sensor
            print ("Sem presença") # imprime na tela que não há presença
            GPIO.output(15, GPIO.LOW) #Mantem LED em nível logico baixo
            time.sleep(1) #Aguarda um segundo para repetir programa
        else : #Quando detectada presença e pino 11 = 1
            print ("Presença detectada") #Imprime na tela presença detectada
            GPIO.output(15, GPIO.HIGH) #Seta nível lógico alto no pino 15 e acende LED
            time.sleep(1) #Aguarda um segundo para repetir programa
except KeyboardInterrupt: #Caso detecta o pressionamento das teclas "Ctrl+C" no teclado
    GPIO.cleanup() #Limpa qualquer configuração da GPIO
    print ("Programa finalizado") #Imprime na tela a finalização do programa
    sys.exit(0) #Função que faz finalizar o programa automaticamente
```

Fonte: Autoria própria

Enfim, com o programa pronto e o circuito esquematizado, o projeto foi implementado fisicamente e podemos vislumbrá-lo na Figura 19.

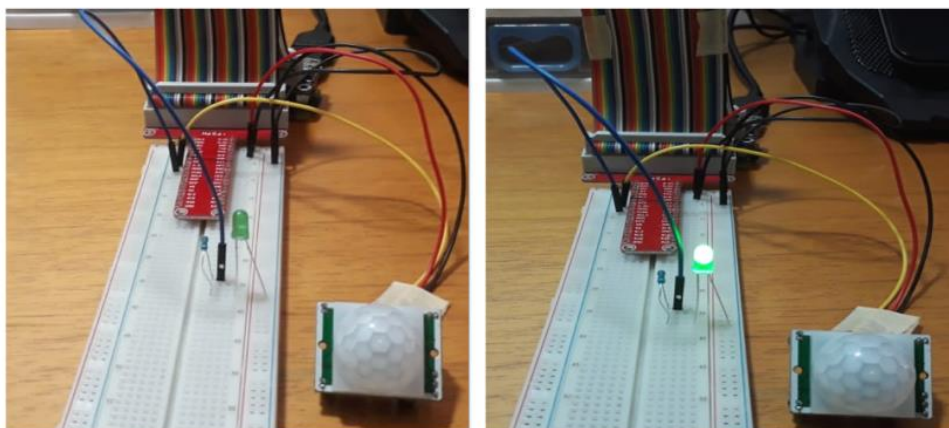
Figura 19: Circuito físico para sensor PIR



Fonte: Autoria própria

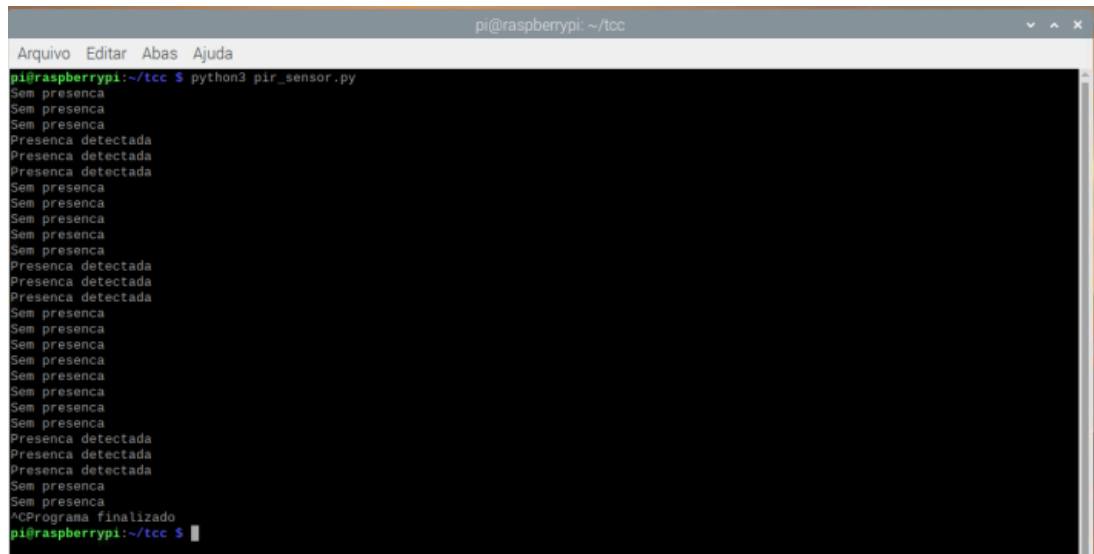
Nas Figuras 20 e 21, temos a sequência do circuito físico em funcionamento e a tela do programa executado, imprimindo caso haja presença ou não.

Figura 20: Circuito em funcionamento do sensor PIR



Fonte: Autoria própria

Figura 21: Programa sendo executado na Raspberry Pi



```
pi@raspberrypi: ~/tcc
Arquivo Editar Abas Ajuda
pi@raspberrypi:~/tcc $ python3 pir_sensor.py
Sem presença
Sem presença
Sem presença
Sem presença
Presença detectada
Presença detectada
Presença detectada
Sem presença
Sem presença
Sem presença
Sem presença
Sem presença
Presença detectada
Presença detectada
Presença detectada
Sem presença
Sem presença
Sem presença
Sem presença
Sem presença
Sem presença
Sem presença
Sem presença
Sem presença
Presença detectada
Presença detectada
Presença detectada
Sem presença
Sem presença
^CPrograma finalizado
pi@raspberrypi:~/tcc $
```

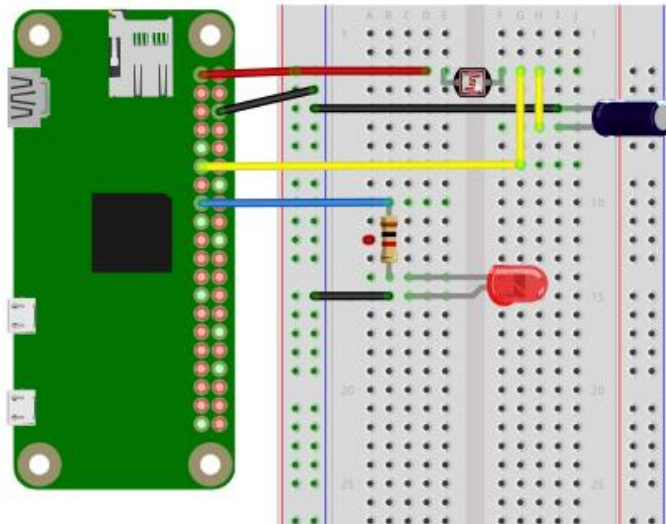
Fonte: Autoria própria

4.2 SENSOR LDR

Mantendo o método de teste para cada componente do projeto, foi implementado para o sensor de luminosidade LDR, o circuito mostrado na Figura 22. O pino 1 que possui como saída a 3.3V, foi ligado na alimentação do sensor LDR, enquanto o pino 6 cuja saída é GROUND, foi ligado ao protoboard, a fim de fornecer a função ground tanto para o sensor LDR, como ao LED que indicará, visualmente, que o sensor está realizando as leituras de luminosidade.

O sinal do sensor LDR foi conectado no pino 11 da GPIO, configurada pela programação como entrada e o LED foi ligado ao pino 15 da GPIO, configurado como saída. No sketch do circuito virtual, o módulo do sensor foi desmembrado em sensor LDR e capacitor de 1 μ F, porém no circuito físico, o capacitor já está integrado no sensor, como será visto na Figura 24.

Figura 22: Circuito para implementação do sensor LDR



Fonte: Autoria própria

O programa em Python para a leitura do sensor e visualização através do LED é visto na Figura 23.

Figura 23: Programação em Python para leitura do sensor de luminosidade

```
pi@raspberrypi: ~/tcc
Arquivo Editar Abas Ajuda
GNU nano 3.2          luz_sensor.py

#!/usr/bin/env python

import RPi.GPIO as GPIO #Importa biblioteca referente aos pinos da placa Raspberry
import time #Importa a biblioteca tempo para utilização da função sleep
import sys #Importa biblioteca que acessa funções usadas pelo proprio interpretador Python

GPIO.setwarnings(False) #Set para ignorar qualquer aviso referente as portas GPIO da placa
GPIO.setmode(GPIO.BOARD) #Seta o modo de utilização da pinagem da GPIO
GPIO.setup(15, GPIO.OUT) #Define o pino 15 da GPIO como pino de saída
GPIO.output(15, GPIO.LOW) #Seta nível lógico baixo para pino 15, iniciando o programa com o LED apagado
pino_sensor = 11 #Atribui a variável pino_sensor o pino a ser utilizado da GPIO como entrada

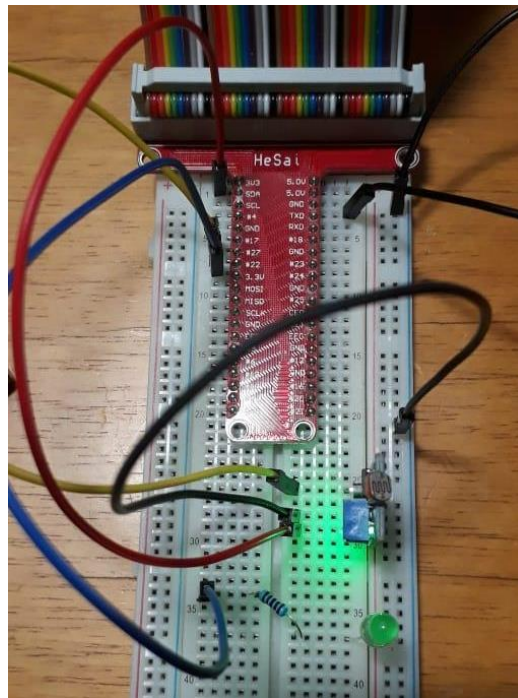
def tempo (pino_sensor): #Define função tempo que irá realizar a conta e fornecer o resultado da leitura do sensor
    conta = 0 #Define variável conta com valor inicial 0
    GPIO.setup(pino_sensor, GPIO.OUT) #Seta o pino do sensor como saída
    GPIO.output(pino_sensor, GPIO.LOW) #Aplica nível lógico baixo no pino do sensor
    time.sleep(0.1) #Programa aguarda 0.1 segundo para continuar
    GPIO.setup(pino_sensor, GPIO.IN) #Agora seta o pino do sensor como entrada
    while (GPIO.input(pino_sensor) == GPIO.LOW): #Enquanto o sensor estiver com nível lógico baixo, carregando o capacitor
        conta += 1 #Incrementa a variável conta em 1
    return conta #Quando o capacitor é carregado e tem-se nível lógico alto no pino do sensor, retorna o valor da variável conta

try:
    while True:
        print tempo(pino_sensor) #Imprime na tela o valor da luminosidade contado pela função tempo
        GPIO.output(15, GPIO.HIGH) #Seta nível lógico alto no LED, indicando que o programa está em andamento
except KeyboardInterrupt: #Define rotina de finalização do programa através das teclas "Ctrl+C"
    print ("Programa finalizado")
    GPIO.cleanup()
    sys.exit(0)
```

Fonte: Autoria própria

Com o programa pronto e o circuito esquematizado, implementamos o circuito fisicamente e o resultado é visto na Figura 24.

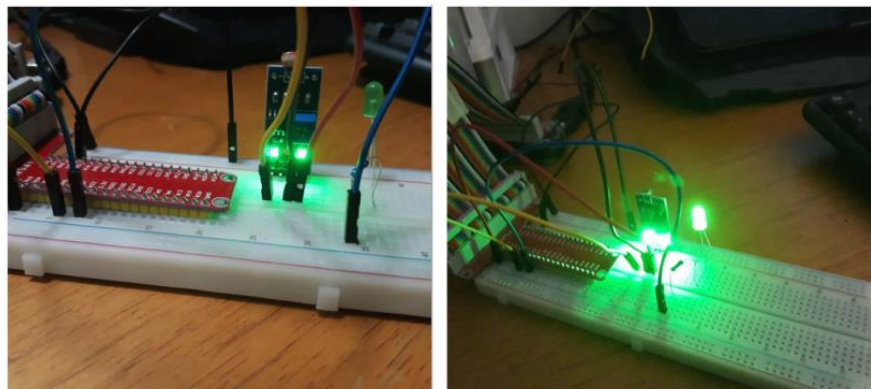
Figura 24: Circuito físico para sensor LDR



Fonte: Autoria própria

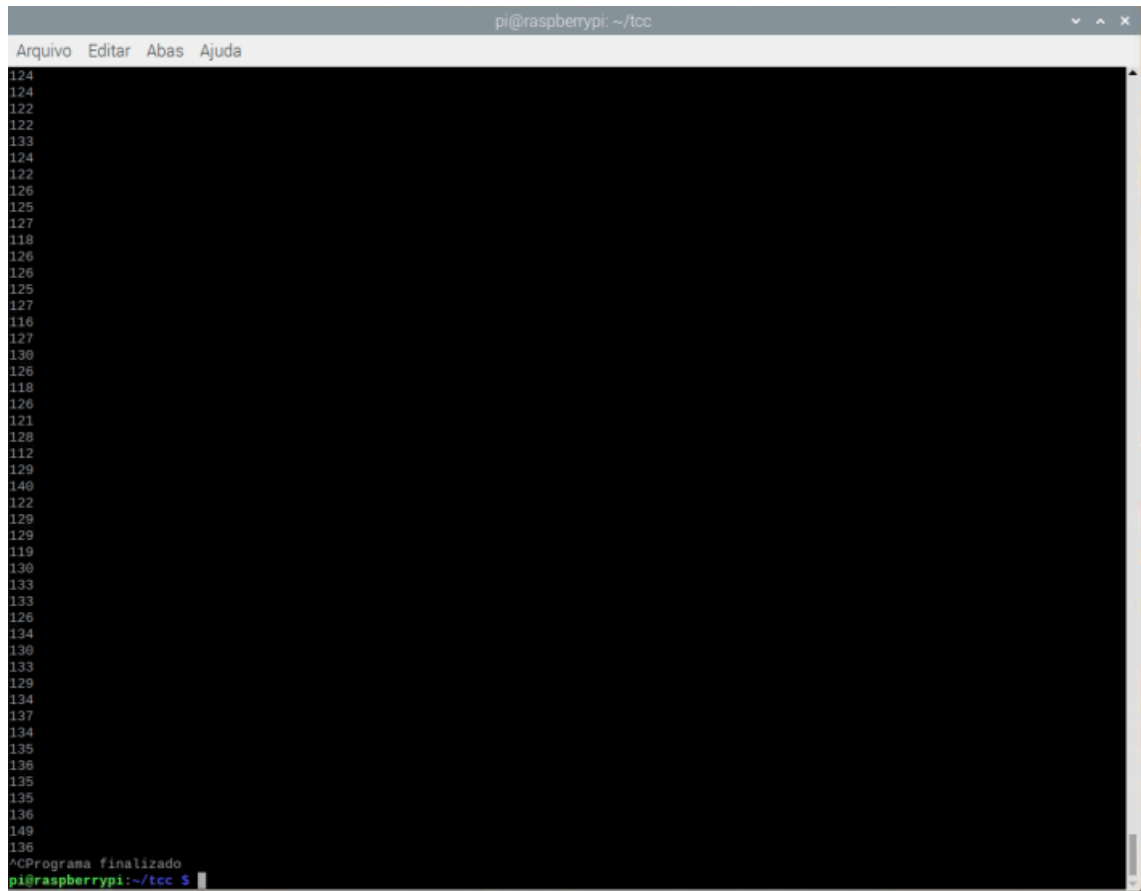
Nas Figuras 25 e 26, temos a sequência do circuito físico em funcionamento e a tela do programa executado ao imprimir as leituras de luminosidade.

Figura 25: Circuito em funcionamento para componente LDR



Fonte: Autoria própria

Figura 26: Programa sendo executado na Raspberry Pi



```
pi@raspberrypi: ~/tcc
Arquivo  Editar  Abas  Ajuda
124
124
122
122
133
124
122
126
125
127
118
126
126
125
127
116
127
130
126
118
126
121
128
112
129
140
122
129
129
119
130
133
133
126
134
130
133
129
134
137
134
135
136
135
135
136
149
136
Programa finalizado
pi@raspberrypi:~/tcc $
```

Fonte: Autoria própria

4.3 SENSOR TEMPERATURA

Para o sensor de temperatura DTH11, fez-se necessário a instalação da biblioteca Adafruit, a qual é responsável por permitir a implementação do sensor DHT11 na programação Python. Para esta instalação foram seguidos os passos descritos a seguir.

Com o Raspberry conectado à internet e com o LX Terminal foram utilizados os seguintes comandos:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git  
cd Adafruit_Python_DHT
```

Antes de continuar com a instalação da biblioteca, utilizamos os comandos abaixo para atualizar o Raspberry:

```
sudo apt-get update  
sudo apt-get install build-essential Python-dev
```

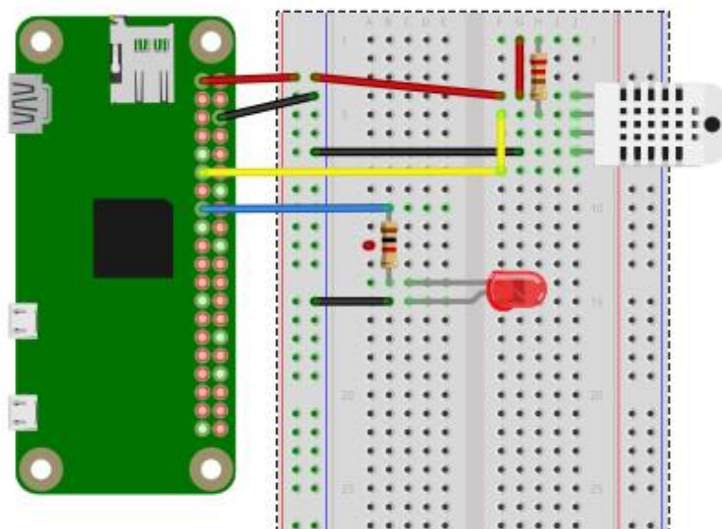
Agora sim, continuamos com a instalação da biblioteca:

sudo Python setup.py install

Com a instalação da biblioteca realizada, foi implantado o circuito mostrado na Figura 27. O pino 1, o qual tem saída 3,3V, foi ligado à alimentação do sensor DHT11, enquanto o pino 6 cuja saída é GROUND, foi ligado ao protoboard, a fim de fornecer a função ground tanto para o sensor, como para o LED, o qual indicará, visualmente, que o sensor realizou uma leitura de temperatura.

O sinal do sensor DHT foi conectado ao pino 11 da GPIO configurada pela programação como entrada e o LED foi ligado no pino 15 da GPIO, configurado como saída.

Figura 27: Circuito para implementação do sensor DHT11



Fone: Autoria própria

Para realizar o programa em Python e ler o sensor, foi necessária a instalação da biblioteca Adafruit tendo sido feito o download do diretório Adafruit_Python_DHT e, dentro dele, instalado o arquivo setup.py. Desta maneira, a leitura do sensor DHT11 é feita através da biblioteca importada no programa, como pode ser visto na Figura 28.

Figura 28: Programação em Python para leitura do sensor DHT11

```
pi@raspberrypi: ~/tcc
Arquivo Editar Abas Ajuda
GNU nano 3.2 temp_sensor.py Modificado

#!/usr/bin/env python

import Adafruit_DHT #Importa a biblioteca do sensor DHT11
import RPi.GPIO as GPIO #Importa biblioteca para utilização dos pinos da GPIO
import time #Importa biblioteca time para utilização de funções
import sys #Importa biblioteca que acessa funções utilizadas pelo proprio interpretador Python

sensor= Adafruit_DHT.DHT11 #Atribui a variavel sensor o tipo de sensor da biblioteca Adafruit, neste caso o sensor DHT11
GPIO.setmode(GPIO.BOARD) #Seta o modo de utilização da pinagem da GPIO
GPIO.setwarnings (False) #Set para ignorar avisos referentes as portas GPIO
pino_sensor = 17 #Define a GPIO conectada ao pino de dados do sensor
GPIO.setup(15, GPIO.OUT) #Seta o pino 15 da GPIO como saída
GPIO.output(15, GPIO.LOW) #Define nível logico baixo na saída do pino 15

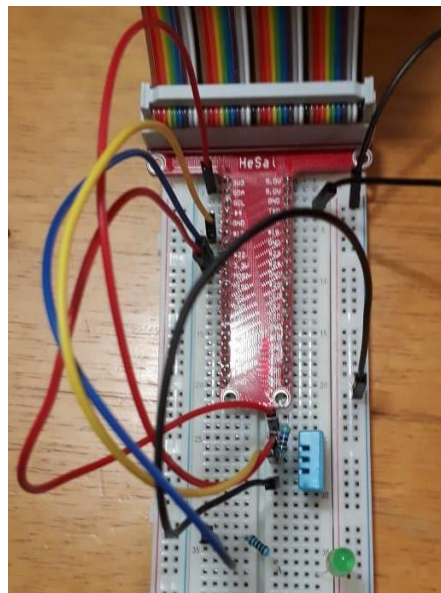
try: #Define programa principal
    print (**** Lendo os valores de temperatura e umidade****) #Imprime na tela mensagem de inicialização do programa

    while(True): #Define loop do programa
        umid, temp = Adafruit_DHT.read_retry(sensor, pino_sensor) #Realiza leitura do sensor
        if umid is not None and temp is not None: #Caso leitura do sensor esteja OK
            print ("Temperatura = {0:0.1f} Umidade = {1:0.1f}%".format(temp, umid)) #Imprime os valores na tela
            GPIO.output (15, GPIO.HIGH) #Seta nível lógico alto e acende LED para visualizar leitura realizada
            print ("Aguarda 5 segundos para efetuar nova leitura...n") #Imprime na tela mensagem de espera
            time.sleep(5) #Espera 5 segundo para reiniciar programa
            GPIO.output (15, GPIO.LOW) #Seta nível logico baixo e apaga LED até seja feita nova leitura
        else:
            print ("Falha ao ler dados do DHT11") #Imprime na tela caso haja erro de leitura do sensor
except KeyboardInterrupt: #Define rotina de finalização do programa
    GPIO.cleanup()
    print ("Programa finalizado")
    sys.exit(0)
```

Fonte: Autoria própria

Com o programa pronto e o circuito esquematizado, implementamos o circuito fisicamente e o resultado pode ser vislumbrado na Figura 29.

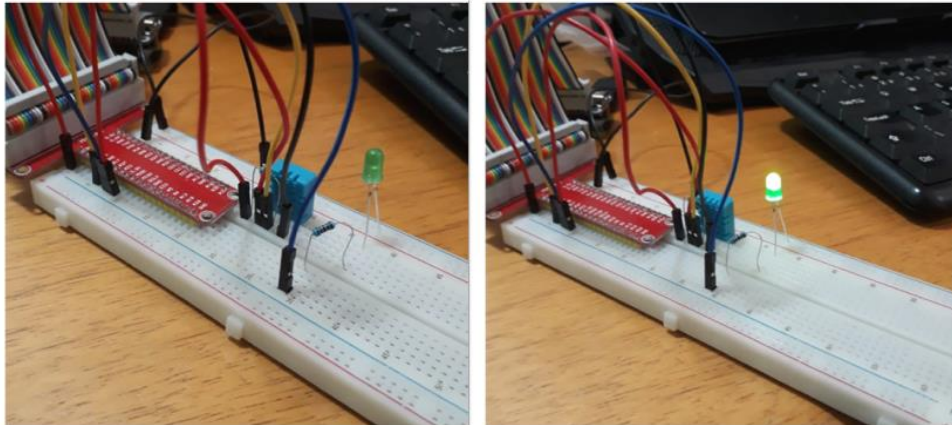
Figura 29: Circuito físico para sensor DHT11



Fonte: Autoria própria

Nas Figuras 30 e 31, temos a sequência do circuito físico em funcionamento e a tela do programa executado ao imprimir as leituras de temperatura e umidade.

Figura 30: Circuito para DHT11 em funcionamento



Fonte: Autoria própria

Figura 31: Programa em execução imprimindo a temperatura e umidade na tela

```
pi@raspberrypi: ~/tcc
Arquivo Editar Abas Ajuda
pi@raspberrypi:~/tcc $ python3 temp_sensor.py
*** Lendo os valores de temperatura e umidade***
Temperatura = 31.0 Umidade = 43.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 43.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 43.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 42.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 42.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 42.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 42.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 42.0n
Aguarda 5 segundos para efetuar nova leitura...n
Temperatura = 31.0 Umidade = 42.0n
Aguarda 5 segundos para efetuar nova leitura...n
Programa finalizado
pi@raspberrypi:~/tcc $
```

Fonte: Autoria própria

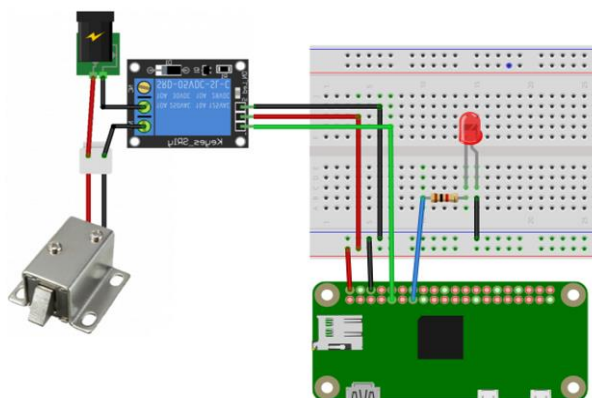
4.4 TRAVA ELÉTRICA

Para o acionamento da trava elétrica, foi implantado o circuito mostrado na Figura 32. O pino 2, cuja saída é 5V, foi ligado na alimentação do módulo relé, enquanto o pino 6, que tem a saída GROUND, foi ligado ao protoboard, a fim de fornecer a função ground, tanto para o relé de comando, como para o LED, o qual indicará, visualmente, que o comando foi enviado ao relé.

O sinal de entrada do relé foi conectado ao pino 11 da GPIO, a qual foi configurada pela programação como saída, enquanto o LED, ligado no pino 15 da GPIO, foi configurado como

saída. Para o funcionamento correto da trava elétrica, fez-se necessário o uso de uma fonte externa de 12V.

Figura 32: Circuito para implementação da trava elétrica



Fonte: Autoria própria

O programa em Python para o acionamento do relé da trava elétrica e visualização através do LED é observado na Figura 33.

Figura 33: Programação em Python para o controle da trava elétrica

```
pi@raspberrypi: ~/tcc
Arquivo Editar Abas Ajuda
GNU nano 3.2 rele.py
#!/usr/bin/env python

import RPi.GPIO as GPIO #Importa biblioteca de utilização da pinagem da GPIO
import time #Importa biblioteca tempo para utilização de funções no programa
import sys #Importa biblioteca que utiliza funções do proprio interpretador Python

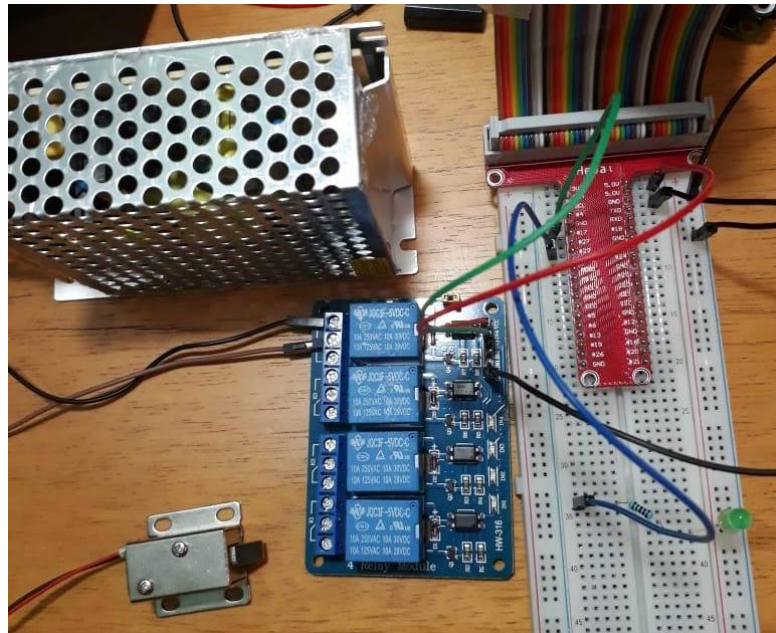
GPIO.setmode(GPIO.BOARD) #Define modo de utilização da pinagem da GPIO
GPIO.setwarnings(False) #Set que ignora avisos referentes as portas GPIO
GPIO.setup(11, GPIO.OUT) #Define pino 11 da GPIO como saída
GPIO.setup(15, GPIO.OUT) #Define pino 15 da GPIO como saída

try: #Inicia programa principal
    print ("Trava aberta") #Imprime na tela que a trava foi aberta
    GPIO.output(11, GPIO.LOW) #Define nivel lógico baixo para o pino ligado ao relé de contato normal fechado
    GPIO.output(15, GPIO.HIGH) #Define nivel logico alto para o pino, acendendo o LED
    time.sleep(5) #Aguarda 5s antes de prosseguir
    print ("Trava fechada") #Imprime na tela que a trava foi fechada
    GPIO.output(11, GPIO.HIGH) #Define nivel logico alto no pino ligado ao rele, alterando estado da trava eletrica
    GPIO.output(15, GPIO.LOW) #Define nivel logico baixo no pino, apagando LED
except KeyboardInterrupt: #Define rotina de finalização do programa
    GPIO.cleanup()
    print ("Programa finalizado")
    sys.exit(0)
```

Fonte: Autoria própria

Com o programa pronto e o circuito esquematizado, implementamos o circuito fisicamente e o resultado é visto na Figura 34.

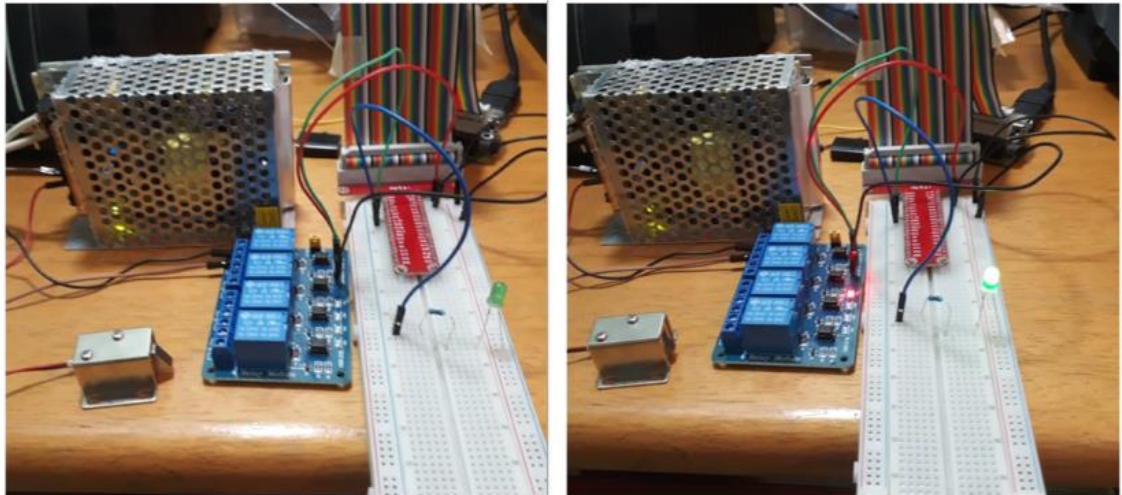
Figura 34: Circuito implementado para trava elétrica e módulo relé



Fonte: Autoria própria

Nas Figuras 35 e 36, temos a sequência do circuito físico em funcionamento e a tela do programa executado ao imprimir os estados da trava elétrica.

Figura 35: Circuito em funcionamento para trava elétrica



Fonte: Autoria própria

Figura 36: Programa sendo executado e imprimindo o estado da trava na tela

A screenshot of a terminal window titled 'pi@raspberrypi: ~/tcc'. The window has a menu bar with 'Arquivo', 'Editar', 'Abas', and 'Ajuda'. The terminal shows the command 'python3 rele.py' being executed, which outputs 'Trava aberta' and 'Trava fechada' on separate lines. The prompt 'pi@raspberrypi:~/tcc \$' is visible at the bottom.

```
pi@raspberrypi: ~/tcc
Arquivo  Editar  Abas  Ajuda
pi@raspberrypi:~/tcc $ python3 rele.py
Trava aberta
Trava fechada
pi@raspberrypi:~/tcc $
```

Fonte: Autoria própria

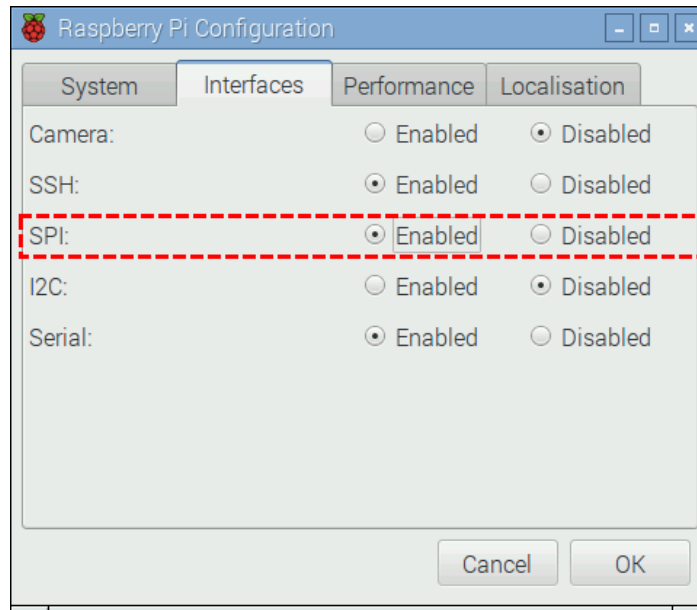
4.5 MÓDULO RFID

Para a correta utilização do módulo RFID, assim como o sensor DHT11, deve ser instalada uma biblioteca própria para que assim, seja configurado via programação. Para este módulo, o processo a ser seguido é um pouco mais complexo, pois também se faz necessária a configuração na placa Raspberry Pi, cuja utilização dos pinos específicos permita a conexão ao módulo RFID, pois na Raspberry a comunicação do leitor RFID RC522 é feita através da interface SPI (Serial Peripheral Interface).

O SPI utiliza comunicação síncrona, full duplex, servindo-se do modelo de comunicação mestre-escravo. Os passos para esta configuração e instalação da biblioteca foram os seguintes:

Para habilitar o SPI no Raspbian acessou-se o Menu > Preferences > Raspberry Pi Configuration e na aba Interface habilitou-se o SPI, como visto na Figura 37.

Figura 37: Janela de configuração da Raspberry



Fonte: Autoria própria

Em seguida, abriu-se o arquivo `/boot/config.txt` (como root) através do comando:

```
sudo nano /boot/config.txt
```

Logo, no mesmo arquivo foi alterado o seguinte parâmetro:

```
dtoverlay=spi-bcm2708
```

Após este processo, realizou-se o comando:

```
ls /dev/spi*
```

tendo como resultado obtido:

```
/dev/spidev0.0 /dev/spidev0.1
```

Para verificar se o módulo SPI foi carregado corretamente, utilizou-se o comando:

```
dmesg | grep spi
```

O resultado indicando a comunicação RFID com Raspberry Pi foi:

```
[ 6.240564] bcm2708_spi 3f204000.spi: master is unqueued, this is deprecated  
[ 6.241262] bcm2708_spi 3f204000.spi: SPI Controller at 0x3f204000 (irq 80)
```

Com a comunicação SPI habilitada, seguiu-se, então, para a instalação da biblioteca MFRC522, referente ao módulo RFID. O primeiro comando a ser utilizado foi:

```
sudo apt-get install Python-dev
```

Após, fez-se a instalação do pacote Python para comunicação SPI. Para isto, executou-se os seguintes comandos, em ordem:

```
git clone https://github.com/lthiery/SPI-Py.git  
cd SPI-Py  
sudo Python setup.py install
```

Para dar sequência e criar um programa para a leitura do leitor RFID, criou-se uma pasta identificada como “rfid”, com a intenção de armazenar o programa e, conseqüentemente, o projeto, executando o comando:

```
mkdir -p ~/tcc/rfid &&& cd ~/tcc/rfid
```

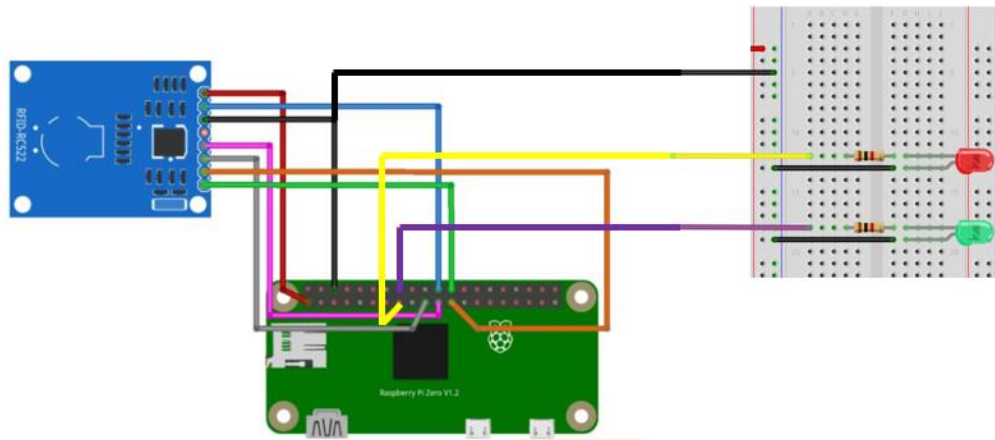
Após criar a pasta onde armazenamos nosso projeto, realizou-se o download do arquivo com a intenção de estabelecer comunicação com o módulo RC522 através do comando:

```
wget https://raw.githubusercontent.com/mxgxw/MFRC522-Python/master/MFRC522.py
```

Após este procedimento, foi utilizado o programa de exemplo da biblioteca para a confecção do programa que será apresentado na Figura 39. O acionamento da trava elétrica foi implantado com circuito como mostrado na Figura 38. O pino 1, cuja saída é 3,3V, foi ligado na alimentação do módulo RFID, enquanto o pino 6 que tem como saída GROUND, foi ligado ao protoboard a fim de fornecer a função ground, tanto para o módulo, como para os LEDs, os quais indicarão visualmente, que houve leitura do cartão RFID.

Conforme citado anteriormente, o módulo RFID se utiliza de um modo específico dos pinos GPIO, a SPI. Desta maneira, os pinos utilizados foram: MOSI, MISO, SCLK, CE0 e o pino 25. No módulo RFID, os sinais MOSI, MISO E SCLK foram ligados nos respectivos sinais da GPIO. O sinal SDA foi ligado no pino CE0 e o reset RST foi ligado no pino 25. Por fim, os LEDs foram ligados nos pinos 15 e 16 da GPIO, configurados como saídas.

Figura 38: Circuito a ser implementado para o módulo RFID



Fonte: Autoria própria

O programa em Python para o acionamento do relé da trava elétrica e visualização através do LED é visto na Figura 39.

Figura 39: Programação em Python para Leitura de cartões no módulo RFID

```
Arquivo Editar Abas Ajuda
pi@raspberrypi: ~/tcc/rfid/SPI-Py/MFRC522-Python
GNU nano 3.2      leitor.py      Modificado

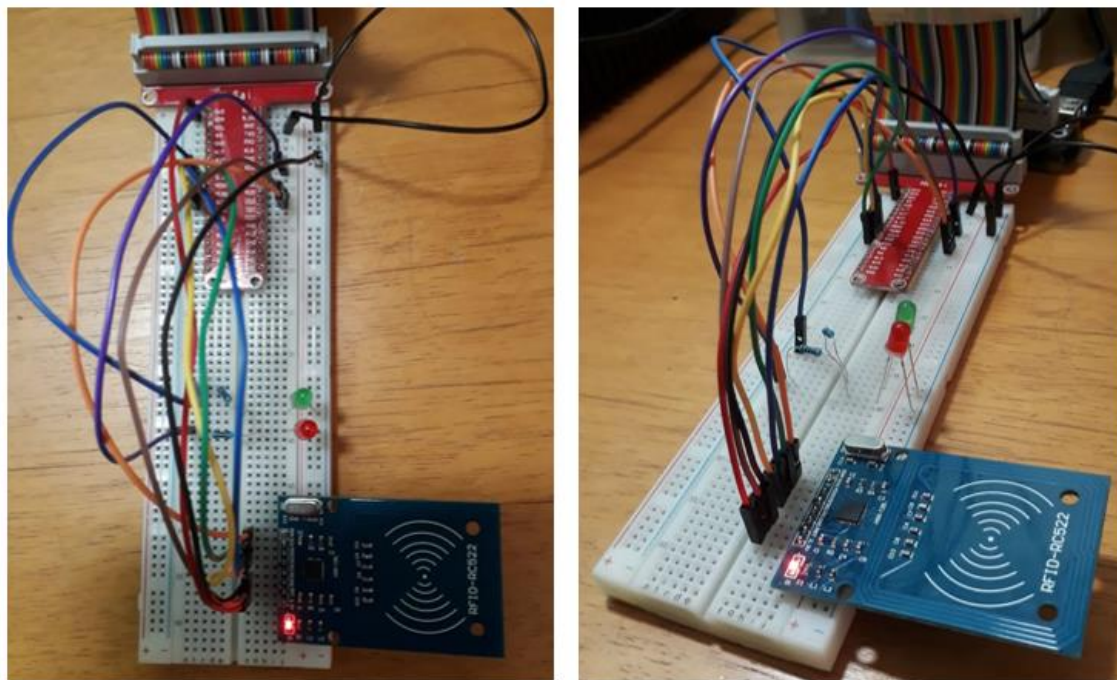
GPIO.setup(16, GPIO.OUT) #Seta o pino 16 da GPIO como saída
GPIO.output(15, GPIO.LOW) #Define nível lógico baixo para o pino 15
GPIO.output(16, GPIO.LOW) #Define nível lógico baixo para o pino 16

try: #Inicia programa principal
    leitor = MFRC522.MFRC522() #Atribui a variavel leitor a função do leitor RFID
    print ("Aproxime o cartao do leitor\nPressione Ctrl+C para encerrar") #Imprime na tela o inicio do programa
    while True: #Inicia laço loop que rodará o programa
        status, tag_type = leitor.MFRC522_Request(leitor.PICC_REQIDL) #Faz a leitura da tag a ser lida
        if status == leitor.MI_OK: #Checa se o status de leitura é OK
            print ("Cartao detectado") #Imprime que o cartão foi detectado
            status, uid = leitor.MFRC522_Anticoll() #Faz a leitura da UID do cartao
            if status == leitor.MI_OK: #Checa se a leitura é OK
                uid = ''.join('%X' % x for x in uid) #Compacta os componentes da UID do cartão em uma unica string
                print ("UID do cartao: %s" % uid) #Imprime na tela a UID do cartão
                if uid in cartao: #Verifica se a UID do cartão está na lista de liberação
                    print ("Acesso Liberado") #Em caso positivo, imprime na tela "Acesso Liberado"
                    GPIO.output(15, GPIO.HIGH) #Define nível lógico alto no pino 15 e acende LED verde
                    GPIO.output(16, GPIO.LOW) #Define nível lógico baixo no pino 16 para apagar LED vermelho
                    time.sleep(5) #Aguarda 5 segundo para nao gerar sobreposição de leitura
                else: #Em caso negativo
                    print ("Acesso negado") #Imprime na tela "Acesso Negado"
                    GPIO.output(16, GPIO.HIGH) #Define nível lógico alto no pino 16 e acende LED vermelho
                    GPIO.output(15, GPIO.LOW) #Define nível lógico baixo no pino 15 e apaga LED verde
                    time.sleep(5) #Aguarda 5 segundos para nao gerar sobreposição na leitura
            except KeyboardInterrupt: #Define rotina de finalização do programa
                GPIO.cleanup()
                print ("Encerrando programa")
                sys.exit(0)
```

Fonte: Autoria própria

Com o programa pronto e o circuito esquematizado, implementou-se o circuito fisicamente e o resultado é vislumbrado na Figura 40.

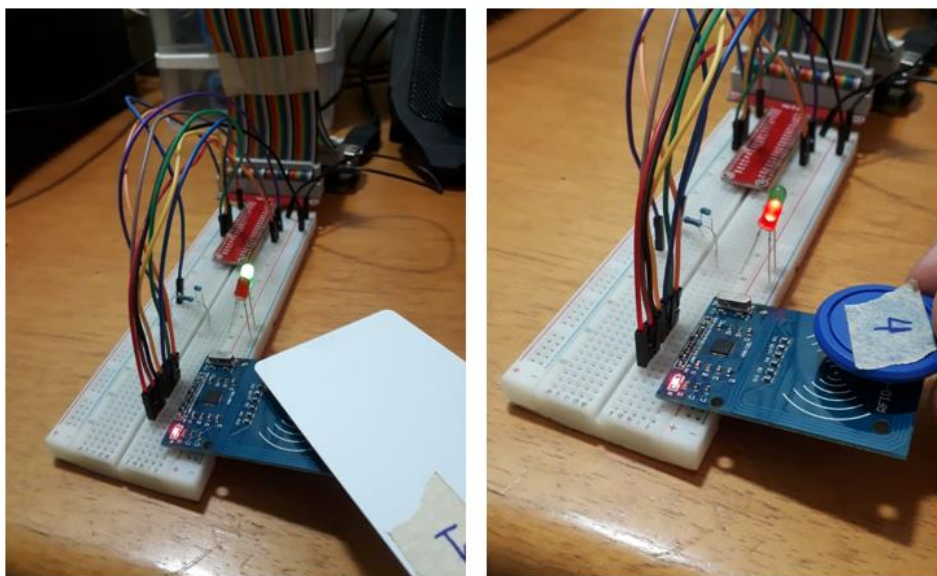
Figura 40: Circuito implementado na protoboard para módulo RFID



Fonte: Autoria própria

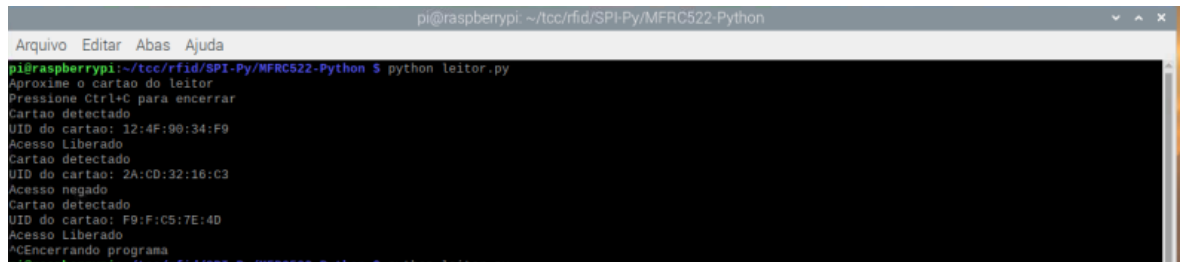
Nas Figuras 41 e 42, temos a sequência do circuito físico em funcionamento e a tela do programa executado imprimindo os estados da trava elétrica.

Figura 41: Circuito em funcionamento com a leitura RFID



Fonte: Autoria própria

Figura 42: Executando programa de leitura RFID e imprimindo a leitura dos cartões



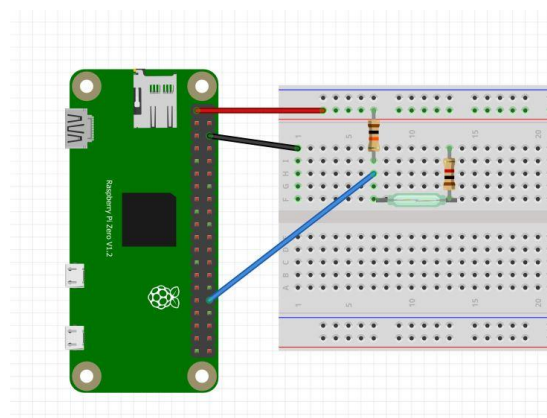
```
pi@raspberrypi: ~/tcc/rfid/SPI-Py/MFRC522-Python
Arquivo  Editar  Abas  Ajuda
pi@raspberrypi:~/tcc/rfid/SPI-Py/MFRC522-Python $ python leitor.py
Aproxime o cartao do leitor
Pressione Ctrl+C para encerrar
Cartao detectado
UID do cartao: 12:4F:90:34:F9
Acesso Liberado
Cartao detectado
UID do cartao: 2A:CD:32:16:C3
Acesso negado
Cartao detectado
UID do cartao: F9:F:C5:7E:4D
Acesso Liberado
^CEncerrando programa
```

Fonte: Autoria própria

4.6 SENSOR REED SWITCH

Mantendo o método de teste para cada componente do projeto, foi implementado, para o sensor *reed switch*, o circuito mostrado na Figura 43. O pino 1, o qual possui como saída a 3.3V, foi ligado no sensor *reed switch* com um resistor *pull-up* de 10k Ω e no pino de entrada 32 da placa, enquanto o pino 6 cuja saída é GROUND, foi ligado ao protoboard, a fim de fornecer a função ground tanto para o sensor, como aos LEDs que indicarão, visualmente, que o sensor está realizando as leituras do campo magnético.

Figura 43: Circuito a ser implementado para o sensor Reed Switch



Fonte: Autoria própria.

O programa em Python para a leitura do sensor *reed switch* é visto na Figura 44.

Figura 44: Programação em Python para Leitura do sensor Reed Switch

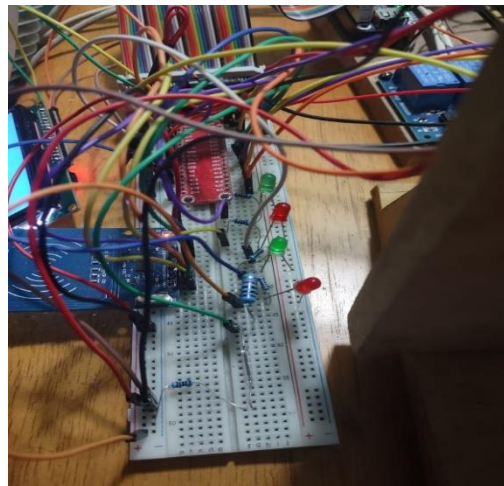
```
Arquivo  Editar  Abas  Ajuda
GNU nano 3.2  sensor_abertura.py  Modificado

import time #Importa biblioteca tempo para utilização de suas funções
import sys #Importa biblioteca sys para utilizar funções do sistema
import RPi.GPIO as GPIO #Importa biblioteca para utilização dos pinos da placa Raspber$
GPIO.setmode(GPIO.BOARD) #Define o modo de utilização dos pinos da GPIO
GPIO.setup(32, GPIO.IN, pull_up_down = GPIO.PUD_UP) #Define o modo de operação do sens$
GPIO.setup (12, GPIO.OUT) #Define o pino de saída para utilização de LED de indicação
GPIO.setup(16, GPIO.OUT) #Define o pino de saída para utilização de LED de indicação
try:
    while True: #Define loop de funcionamento
        input = GPIO.input(32) #Define a variável input com valor da entrada d$
        print (input) #Imprime valor de leitura do sensor
        if input == 1: #Define condição de leitura do sensor
            GPIO.output(12, GPIO.HIGH) #Caso valor de leitura = 1, acende $
            GPIO.output(16, GPIO.LOW) #Apaga LED vermelho
        else: #Caso valor de leitura do sensor = 0
            GPIO.output(12, GPIO.LOW) #Apaga LED verde
            GPIO.output(16, GPIO.HIGH) #Acende LED vermelho
```

Fonte: Autoria própria

Com o programa pronto e o circuito esquematizado, implementou-se o circuito fisicamente e o resultado é vislumbrado através da Figura 45.

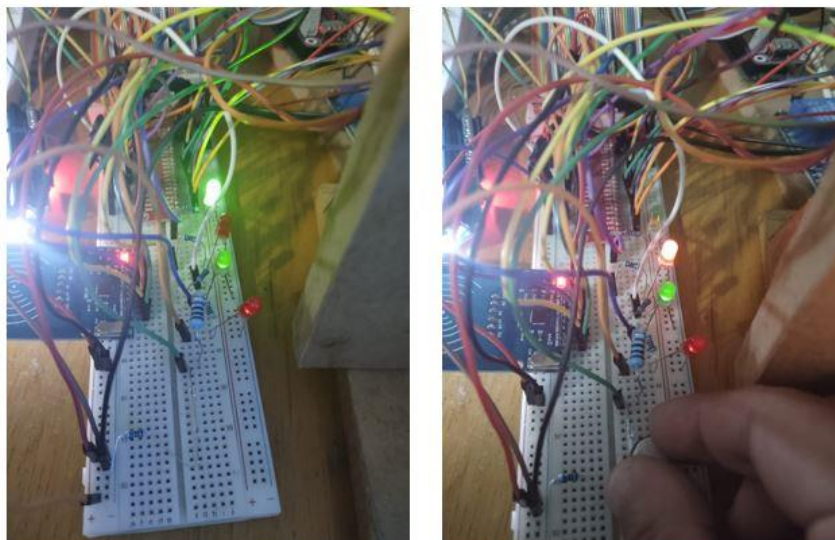
Figura 45: Circuito implementado na protoboard para sensor reed switch



Fonte: Autoria própria

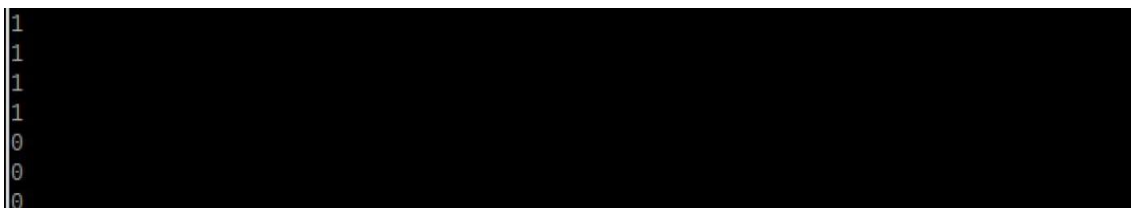
Nas Figuras 46 e 47, temos a sequência do circuito físico em funcionamento e a tela do programa executado imprimindo os estados do sensor.

Figura 46: Circuito em funcionamento com a leitura sensor reed switch



Fonte: Autoria própria

Figura 47: Executando programa de leitura do sensor reed switch



Fonte: Autoria própria

4.7 DISPLAY LCD I2C

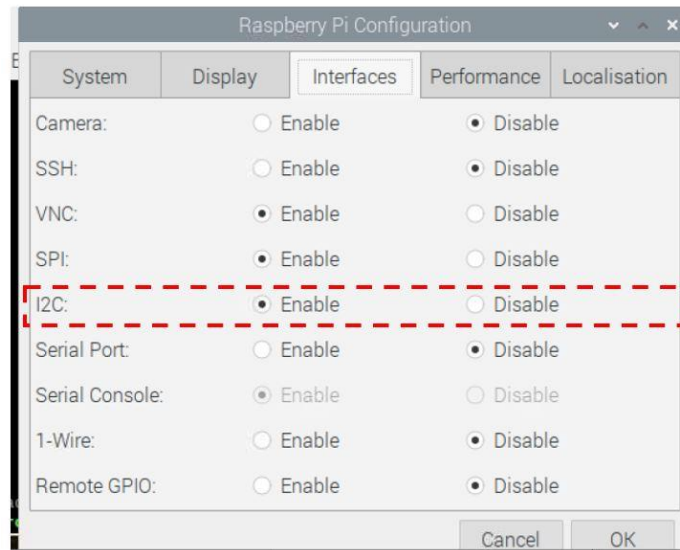
Para a correta utilização do display LCD I2C, assim como o sensor DHT11, deve ser instalada uma biblioteca própria para que, então, seja configurado via programação. Para este módulo, o processo a ser seguido é um pouco mais complexo, pois também se faz necessária a configuração na placa Raspberry Pi, cuja utilização dos pinos específicos permita a conexão ao módulo I2C, pois na Raspberry a comunicação com o módulo é realizada através da interface I2C (Inter-Integrated Circuit).

O I2C assim como na comunicação assíncrona, utiliza de dois canais para comunicação, porém, um canal é para transmissão e recepção de dados e outro para sincronização dos mesmos, tornando-se, dessa forma, uma comunicação síncrona, como a interface SPI utilizada

pelo módulo RFID já explicado em outra secção. Os passos para esta configuração e instalação da biblioteca estão descritos a seguir.

Para habilitar o SPI no Raspbian acessou-se o Menu > Preferences > Raspberry Pi Configuration e na aba Interface habilitou-se o SPI, como visto na Figura 48.

Figura 48: Janela de configuração da Raspberry Pi



Fonte: Autoria própria

Em seguida, foram instalados os pacotes *I2C-Tools* e o *smbus* através do comando:

```
sudo apt-get install i2c-tools
```

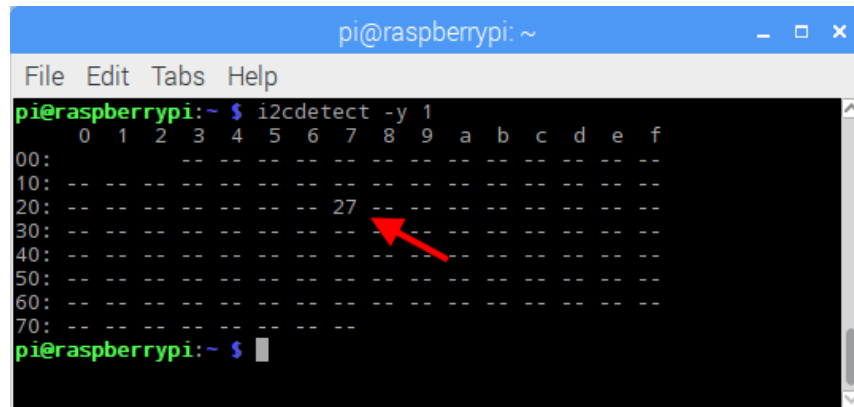
```
sudo apt-get install Python-smbus
```

Para mostrar o endereço I2C do display LCD executou-se o comando:

```
i2cdetect -y 1
```

O resultado obtido é visto na Figura 49.

Figura 49: Endereço do display na placa Raspberry Pi



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ i2cdetect -y 1  
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
20: -- -- -- -- -- -- 27 -- -- -- -- -- -- --  
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
pi@raspberrypi:~$
```

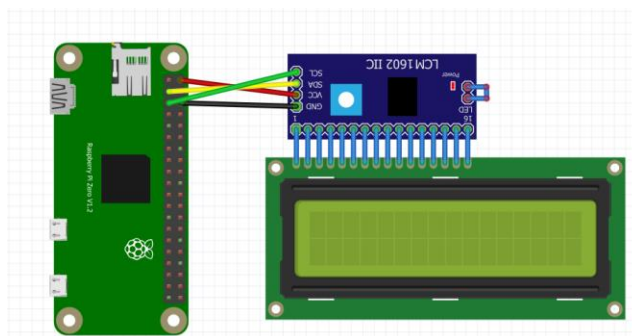
Fonte: Autoria própria

Após este processo, realizou-se o comando para instalar a biblioteca aberta para utilização do display LC:

sudo pip3 install rpi_lcd

Em seguida, foi utilizado o programa de exemplo da biblioteca para a confecção do programa, que será apresentado na Figura 51. A utilização do display LCD 16x2 I2C foi implantado com circuito como mostrado na Figura 50. O pino 2, cuja saída é 5V, foi ligado na alimentação do módulo I2C, enquanto o pino 6 que tem como saída GROUND, foi ligado ao ground do módulo. As saídas SDA e SDL do módulo I2C foram conectadas aos pinos 3 e 5 da placa Raspberry Pi, respectivamente.

Figura 50: Circuito a ser implementado para o display 16x2 I2C



Fonte: Autoria própria

O programa em Python para o acionamento do display é visto na Figura 51.

Figura 51: Programação em Python para escrita no display LCD

```
GNU nano 3.2                                LCD_1.py
from rpi_lcd import LCD
lcd = LCD()

try:
    while True:
        lcd.text('Hello World',1)
        lcd.text('Raspberry Pi!', 2)
except KeyboardInterrupt:
    lcd.clear()
```

Fonte: Autoria própria

Com o programa pronto e o circuito esquematizado, implementou-se o circuito fisicamente e o resultado é vislumbrado na Figura 52.

Figura 52: Circuito implementado na protoboard para display LCD 16x2 I2C



Fonte: Autoria própria

Nas Figuras 53 e 54, temos a sequência do circuito físico em funcionamento e a tela do programa executado.

Figura 53: Circuito em funcionamento com o display LCD



Fonte: Autoria própria

Figura 54: Executando programa de escrita no display LCD

```
pi@raspzerow: ~/tcc/rfid/SPI-Py/MFRC522-Python
Arquivo  Editar  Abas  Ajuda
pi@raspzerow:~/tcc/rfid/SPI-Py/MFRC522-Python $ python3 LCD_1.py
Escrevendo no display LCD
Escrevendo no display LCD
Escrevendo no display LCD
Escrevendo no display LCD
Escrevendo no display LCD
Escrevendo no display LCD
```

Fonte: Autoria própria

4.8 COMUNICAÇÃO MQTT

Assim como o sensor DHT11 e o módulo RFID, a utilização do protocolo de comunicação em Python precisa que algumas ferramentas e bibliotecas sejam adicionadas para seu pleno funcionamento. Neste projeto, foi utilizada a biblioteca Mosquitto, a qual permitiu configurar a placa Raspberry Pi como um servidor (Broker) para ser controlada via web através do protocolo MQTT. Isto, de acordo com a sequência descrita a seguir.

O primeiro passo foi realizar um update do sistema para garantir que todos os repositórios estivessem atualizados:

sudo apt-get update

sudo apt-get upgrade

Em seguida, instalou-se o Mosquitto (MQTT) no Raspberry Pi:

sudo apt install -y mosquitto mosquitto-clients

Com a instalação completa foi executado o seguinte comando:

sudo systemctl enable mosquitto.service

Para garantir que a comunicação fosse segura e não houvesse utilização indevida do sistema, foram seguidos os passos abaixo descritos, para criar credenciais de segurança na comunicação MQTT.

Desenvolveu-se um ficheiro responsável por armazenar as credenciais ***username*** e a ***password***, as quais, posteriormente, ficarão encriptadas, através do seguinte comando:

sudo mosquitto_passwd -c /etc/mosquitto/passwd <root>
</root>

Neste comando, o usuário ***root*** foi criado como credencial. Em seguida, o sistema solicitou duas vezes a senha que será utilizada. Após a criação das credenciais de segurança, foi indicado ao sistema onde estão guardadas as credenciais para bloqueio de acesso a utilizadores do protocolo MQTT, para que, aqueles que não as possuam não possam utilizar o sistema:

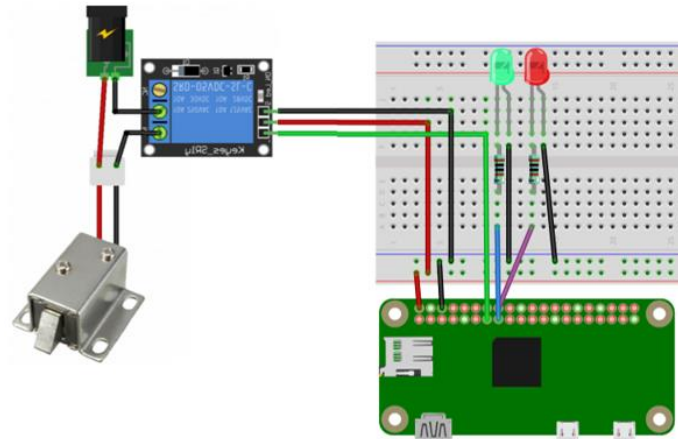
sudo nano /etc/mosquitto/mosquitto.conf
password_file /etc/mosquitto/passwd
allow_anonymous false

“Allow_anonymous false” como inserido no arquivo, deve estar definido como ***false*** para restringir acessos anônimos.

Feita toda a configuração de software, para o acionamento da trava elétrica através da comunicação MQTT, implantou-se o circuito mostrado na Figura 55. O pino 2, o qual tem como saída 5V, foi ligado à alimentação do módulo relé, enquanto o pino 6, cuja saída é GROUND, foi ligado ao protoboard, a fim de fornecer a função ground, tanto para o relé de comando, como para os LEDs, os quais indicarão visualmente que o comando foi enviado ao relé.

O sinal de entrada do relé foi conectado ao pino 13 da GPIO, configurada pela programação como saída, enquanto o LED verde foi ligado ao pino 15 da GPIO, configurado como saída e, o LED vermelho foi ligado no pino 16 da GPIO, configurado como saída. Para o funcionamento correto da trava elétrica se fez necessário a utilização de uma fonte externa de 12V.

Figura 55: Circuito a ser implementado para o controle da trava elétrica



Fonte: Autoria própria

O programa em Python para o acionamento do relé da trava elétrica e visualização através dos LEDs é visto na Figura 56.

Figura 56: Programação em Python para controle da trava elétrica através do protocolo MQTT

```
pi@raspzerow: ~/tcc
Arquivo Editar Abas Ajuda
GNU nano 3.2 rele_mqtt.py Modificado

#!/usr/bin/env python
import paho.mqtt.client as mqtt #Importa biblioteca de utilizacao do client mqtt
import RPi.GPIO as GPIO #Importa a biblioteca de utilizacao da pinagem GPIO
import sys #Importa biblioteca que utiliza funcoes do proprio interpretador Python

GPIO.setwarnings(False) #Desativa avisos referentes as portas GPIO
GPIO.setmode(GPIO.BOARD) #Seta o modo de utilizacao da pinagem da GPIO
GPIO.setup(13, GPIO.OUT) #Define o pino 13 como saida
GPIO.output(13, GPIO.HIGH) #Define nivel logico alto no pino 13, inicializando a trava no estado fechado
GPIO.setup(15, GPIO.OUT) #Define o pino 15 como saida
GPIO.output(15, GPIO.LOW) #Define nivel logico baixo no pino 15, inicializando o LED apagado
GPIO.setup(16, GPIO.OUT) #Define o pino 16 como saida
GPIO.output(16, GPIO.LOW) #Define nivel logico baixo no pino 16, inicializando o LED apagado

#Define função que recebe o callback para quando o cliente se conecta ao servidor MQTT e recebe uma resposta
def on_connect(client, userdata, flags, rc):
    print("Conectando ao MQTT Broker") #Imprime na tela que houve conexao
    client.subscribe("myTopic") #Subcreve o client MQTT no topico a ser utilizado

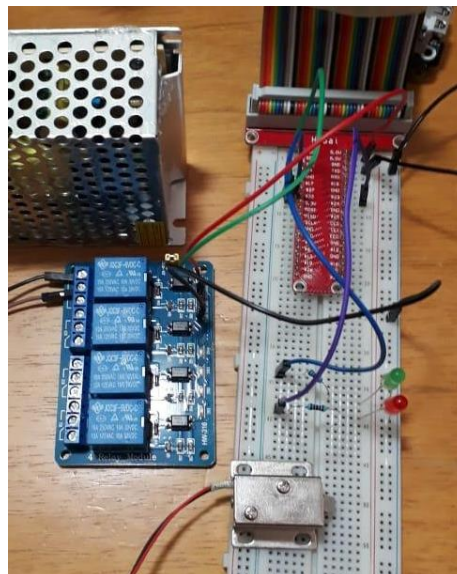
#Define função que recebe a mensagem do tópico MQTT e realiza as devidas acoes
def on_message(client, userdata, msg):
    payload = int(msg.payload) #Garante que a mensagem recebida pelo mqtt esta em formato inteiro
    print(msg.topic+ " " +str(payload)) #Imprime na tela o topico na qual foi enviada a mensagem e qual foi a mensagem enviada
    if payload == 1: #Verifica o valor da mensagem para dar sequencia
        print("Trava aberta") #Caso a mensagem seja o valor 1, imprime na tela que a trava foi aberta
        GPIO.output(13, GPIO.LOW) #Envia nivel logico baixo para o pino 13 que altera o estado da trava para aberta
        GPIO.output(15, GPIO.HIGH) #Envia nivel logico alto para o pino 15 e acende o LED verde
        GPIO.output(16, GPIO.LOW) #Envia nivel logico baixo para o pino 16 e mantem o LED vermelho apagado
    else: #Caso a mensagem seja diferente de 1
        print("Trava fechada") #Imprime na tela que a trava esta fechada
        GPIO.output(13, GPIO.HIGH) #Envia nivel logico alto para o pino 13 que altera o estado da trava para fechada
        GPIO.output(15, GPIO.LOW) #Envia nivel logico baixo para o pino 15 que apaga o LED verde
        GPIO.output(16, GPIO.HIGH) #Envia nivel logico alto para o pino 16 e acende o LED vermelho

try:
    client = mqtt.Client() #Cria uma constante chamada client
    client.on_connect = on_connect #Faz a conexao com o servidor
    client.on_message = on_message #Recebe a mensagem publicada no servido MQTT
    client.connect("raspzerow.local", port = 1883, keepalive =120) #Configuracao do servidor, porta e conectividade do servidor MQTT
    client.username_pw_set("root", "neri2143") #Configura usuario e senha para conexao
    client.loop_forever() #chamada de funcao que mantem o programa em looping, sempre lendo as mensagens a serem publicadas no topico MQTT
except KeyboardInterrupt: #Define rotina de finalizacao do programa
    print("Programa finalizado")
    GPIO.cleanup()
    sys.exit(0)
```

Fonte: Autoria própria

Com o programa pronto e o circuito esquematizado, foi implementado o circuito fisicamente e o resultado é visto na Figura 57.

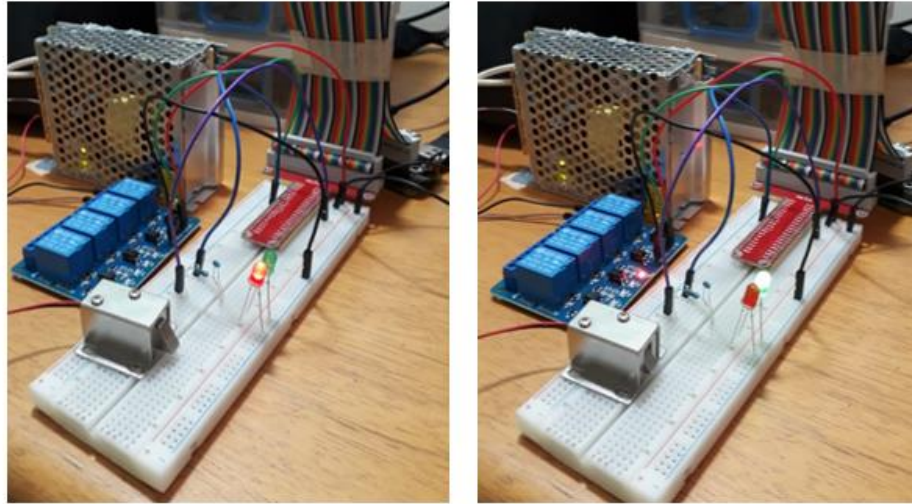
Figura 57: Circuito implementado na protoboard para controle da trava elétrica através da web



Fonte: Autoria própria

Nas Figuras 58, 59 e 60, tem-se a sequência do circuito físico em funcionamento e a tela do programa executado na Raspberry Pi, a qual imprime os estados da trava elétrica enquanto o cliente MQTT é utilizado no notebook para enviar os sinais de comando à Raspberry.

Figura 58: Circuito implementado para o controle via comunicação MQTT



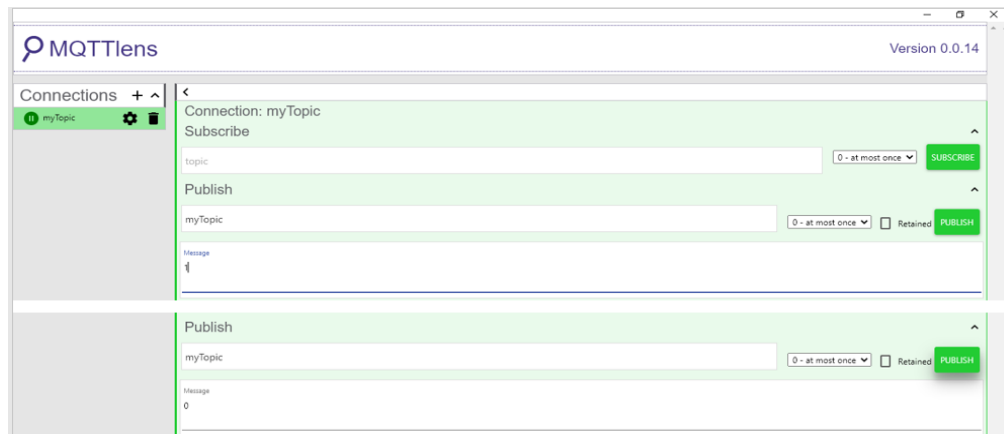
Fonte: Autoria própria

Figura 59: Executando programa de controle da trava elétrica imprimindo na tela o estado da trava e mensagem enviada via MQTT

```
pi@raspzerow:~/tcc $ python rele_mqtt.py
Conectando ao MQTT Broker5
Conectando ao MQTT Broker0
myTopic
0 payload e 1
Trava aberta
myTopic
0 payload e 0
Trava fechada
^CPrograma finalizado
```

Fonte: Autoria própria

Figura 60: Cliente MQTT executado no notebook em rede com a Raspberry Pi e enviando as mensagens “1” e “0” para acionamento da trava elétrica

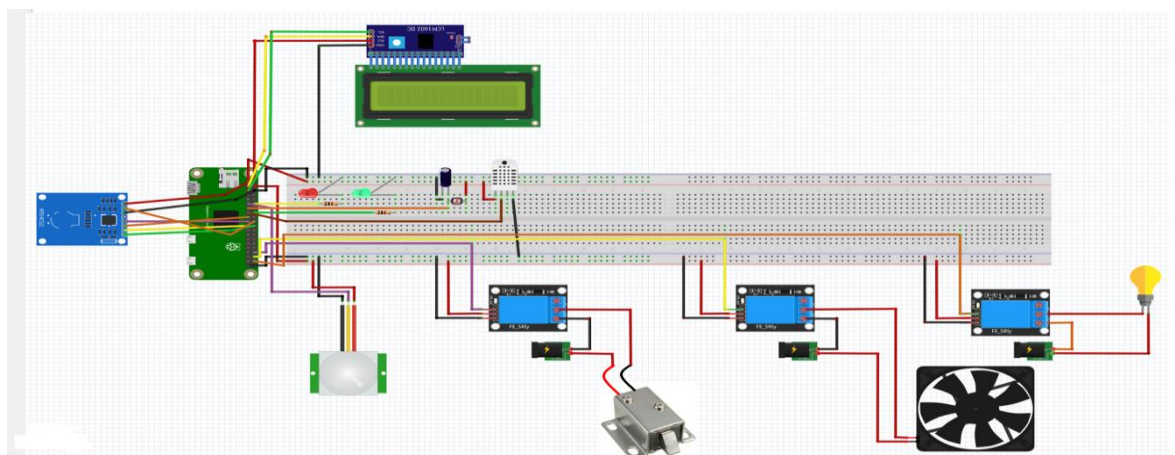


Fonte: Autoria própria

5. RESULTADOS E DISCUSSÃO

Realizados todos os testes apresentados na seção anterior e adquirido todo conhecimento prático em relação à programação e utilização dos componentes implementados neste projeto, realizou-se a programação que ditará o funcionamento do sistema de automação residencial. Para esta etapa do projeto, seguindo a ideia proposta na seção 3, o circuito implementado foi o visto na Figura 61.

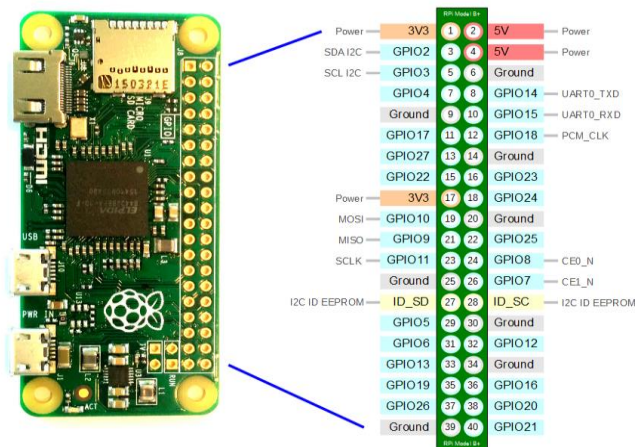
Figura 61: Circuito elétrico implementado para Projeto Final



Fonte: Autoria própria

Seguindo a Figura 62 e a Tabela 1 como referência para a leitura da Figura 61, aquelas demonstram a utilização e as conexões feitas dos pinos da Placa Raspberry Pi.

Figura 62: Pinagem da Placa Raspberry Pi



Fonte [19]

Tabela 1: Ligação dos componentes do projeto na placa Raspberry Pi

Componente	Pino GPIO	Descrição do Pino
Alimentação 3,3V Protoboard	Pino 1	3,3 V POWER
Alimentação 5V Protoboard	Pino 2	5V POWER
Terra	Pino 39	GND
Sensor de presença	Pino 11	GPIO 17
Sensor de luminosidade	Pino 13	GPIO 27
Sensor de temperatura	Pino 15	GPIO 22
Sensor de abertura	Pino 32	GPIO 12
Conexões RFID	Pino 19	MOSI
Conexões RFID	Pino 21	MISO
Conexões RFID	Pino 23	SCLK
Conexões RFID	Pino 22	GPIO 25
Conexões RFID	Pino 24	CE0
Relé de controle da Trava Elétrica	Pino 36	GPIO 16
Relé de controle do Ventilador	Pino 38	GPIO 20
Relé de controle da Lâmpada	Pino 40	GPIO 21
LED de indicação modo automático	Pino 12	GPIO 18
LED de indicação modo manual	Pino 16	GPIO 23
Entrada SDA	Pino 3	GPIO 2 - SDA
Entrada SCL	Pino 5	GPIO 3 - SCL
LED indicação porta-aberta	Pino 29	GPIO 5
LED indicação porta-fechada	Pino 31	GPIO 6

Fonte: Autoria própria

De acordo com a identificação da pinagem apresentada na Tabela 1 e o circuito proposto pelo esquema na Figura 61, podemos observar, na Figura 63, a simulação da implementação do circuito real em um cômodo residencial.

Figura 63: Circuito final implementado na simulação de cômodo residencial



Fonte: Autoria própria

Com o circuito instalado, programou-se, em Python, o projeto a ser executado na placa Raspberry Pi. Este, irá configurar a placa como servidor (Broker MQTT) responsável por realizar as leituras dos sensores do cômodo, e, sendo assinante do tópico “Modo de operação” irá receber do serviço web, desenvolvido com a ferramenta Node-RED, através de uma mensagem via protocolo MQTT, a seleção do modo de operação do programa. Para esta configuração do servidor, a fim de manter a segurança do ambiente, foram definidas credenciais para acesso, com usuário de login e senha.

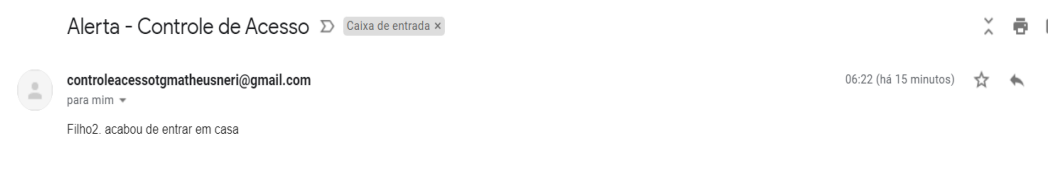
Sendo assim, ao dar permissão à placa Raspberry para controlar automaticamente o acionamento da iluminação e ventilação, ou caso o modo selecionado seja manual, ao assinar os tópicos “Iluminação”, “Ventilação” e “Acesso” via comunicação MQTT, permitirá ao usuário controlar o acionamento da iluminação e ventilação do cômodo, além de admitir o acionamento da trava que mantém a porta fechada através da aplicação web, paralelamente ao acionamento da porta pelo leitor RFID. O programa Python implementado é o apresentado no Anexo A.

Através da programação via linguagem Python, configurou-se a placa Raspberry Pi como servidor para comunicação MQTT, na qual ela irá receber as mensagens de controle do cliente

(Aplicação Web) e ao mesmo tempo publicar as informações, como o valor de leitura do sensor de temperatura.

Pensando em uma aplicação doméstica e segura, incluiu-se neste projeto a ação de controle de acesso para menores de idade, portanto, no programa definiu-se uma lista de *tags* de leitura RFID, as quais, ao serem lidas pelo leitor, enviam uma mensagem à aplicação web de que o menor adentrou ao cômodo. Este controle pode simular o controle parental para saber quando o filho está chegando em casa, por exemplo. Além disso, utilizando os módulos de comunicação através do e-mail, é possível enviar essa notificação de controle de acesso parental ao e-mail previamente definido, como pode ser visto na Figura 64.

Figura 64: Notificação de e-mail enviada pela aplicação web desenvolvida



Fonte: Autoria própria

A aplicação web desenvolvida através da ferramenta Node-RED é apresentada na Figura 65 e 66. A primeira é referente à aplicação desenvolvida para utilização via web browser em computadores e notebooks conectados na mesma rede que o servidor (placa Raspberry Pi), e a segunda demonstra a aplicação desenvolvida para ser utilizada em *mobile*. A Tabela 2 apresenta os módulos utilizados para o desenvolvimento da aplicação web.

Tabela 2: Módulos utilizados para construção da aplicação web

(continua)

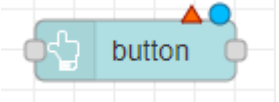
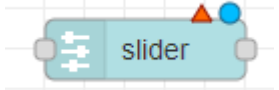
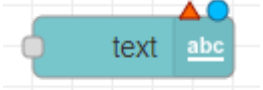
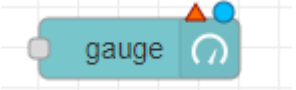
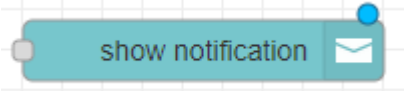
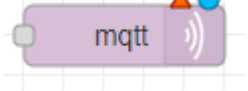
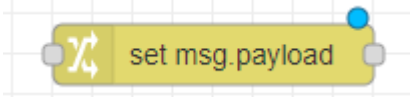
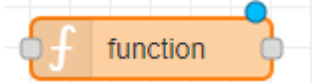
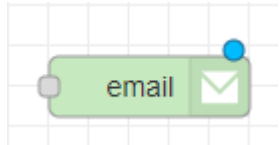
Módulo utilizado	Nome	Função
	<i>Switch Node</i>	Botão utilizado para alternar os estados da mensagem enviada ao controlador
	<i>Button</i>	Botão utilizado para acionamento direto e envio de mensagem única ao controlador
	<i>Slider</i>	Botão funcional que permite o envio de uma mensagem pelo controle de um slider linear
	<i>Text</i>	Caixa de texto que irá disponibilizar em forma de texto a mensagem recebida pelo módulo
	<i>Gauge</i>	Módulo ilustrativo que irá disponibilizar em forma visual a mensagem recebida pelo módulo
	<i>Show notification</i>	Módulo que fará aparecer uma notificação na aplicação
	<i>MQTT IN</i>	Módulo que recebe uma mensagem no protocolo MQTT do controlador
	<i>MQTT OUT</i>	Módulo que envia uma mensagem MQTT para o controlador

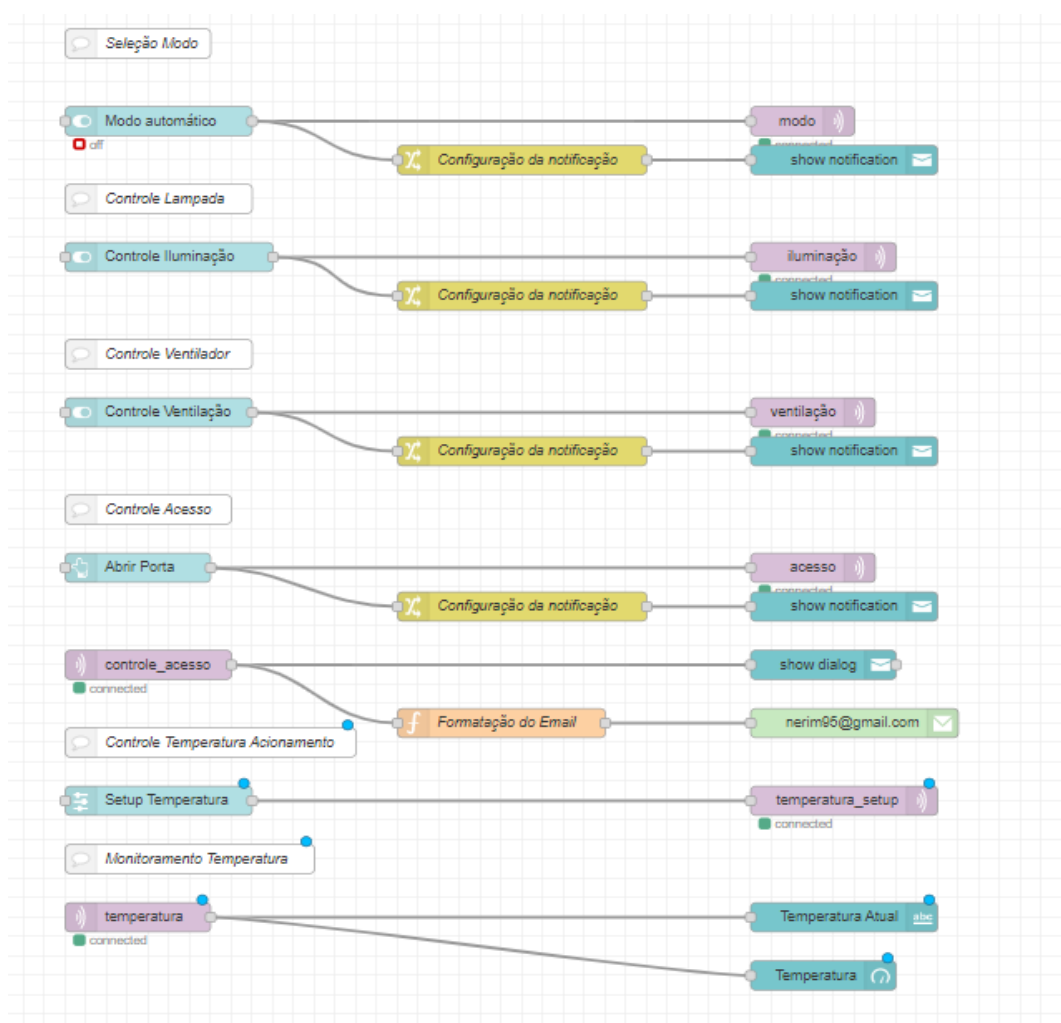
Tabela 2: Módulos utilizados para construção da aplicação web

(conclusão)

Módulo utilizado	Nome	Função
	<i>Set Message</i>	Módulo utilizado para configuração da mensagem a ser disponibilizada na aplicação
	<i>Function</i>	Módulo utilizado para diversas funções, nesta aplicação utilizado para configuração do e-mail a ser enviado
	<i>Email out</i>	Módulo que irá enviar um e-mail com as configurações definidas

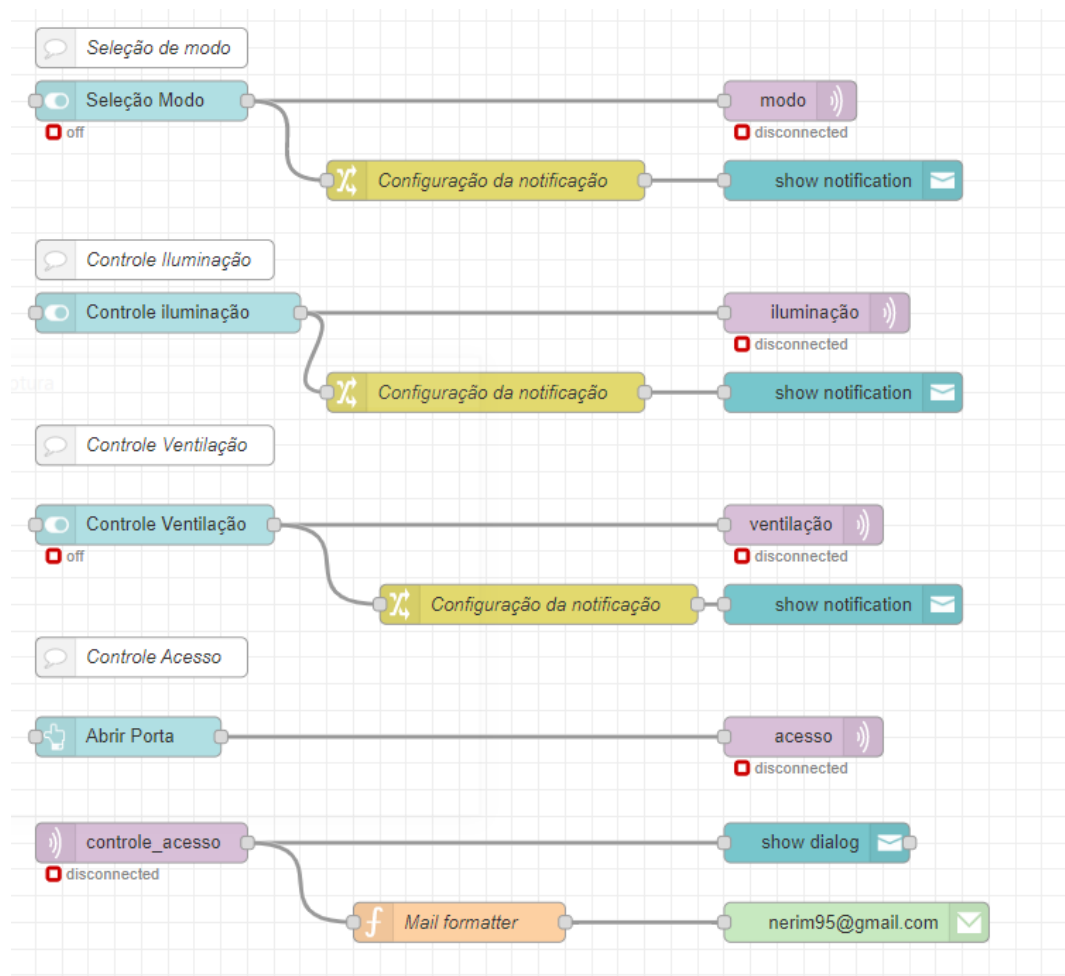
Fonte: Autoria Própria

Figura 65: Estrutura de nós utilizados para construção da aplicação web desktop de controle do sistema de automação



Fonte: Autoria própria

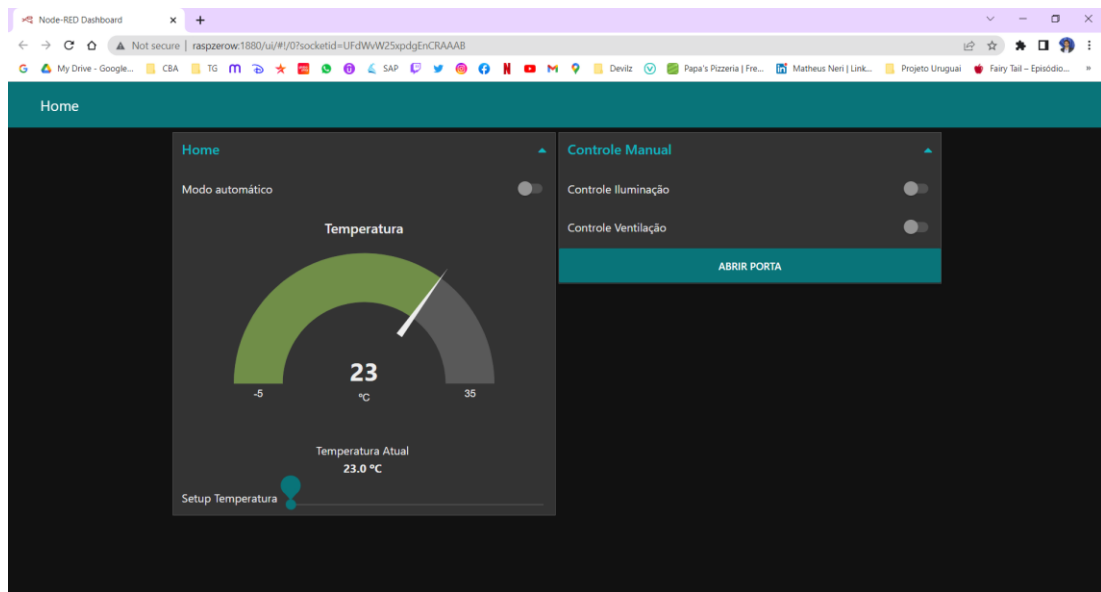
Figura 66: Estrutura de nós utilizados para construção da aplicação web mobile de controle do sistema de automação



Fonte: Autoria própria

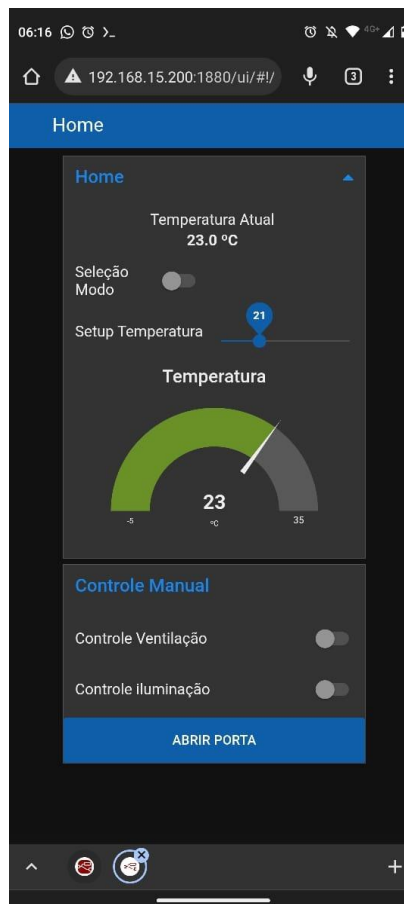
Para cada estrutura de nós da ferramenta Node-RED foi gerado um *dashboard* que é a aplicação web funcional, demonstrada nas Figuras 67 e 68 abaixo, representando as aplicações *web desktop* e *mobile*, respectivamente.

Figura 67: Aplicação web desktop utilizada para monitoramento e controle do sistema automático



Fonte: Autoria própria

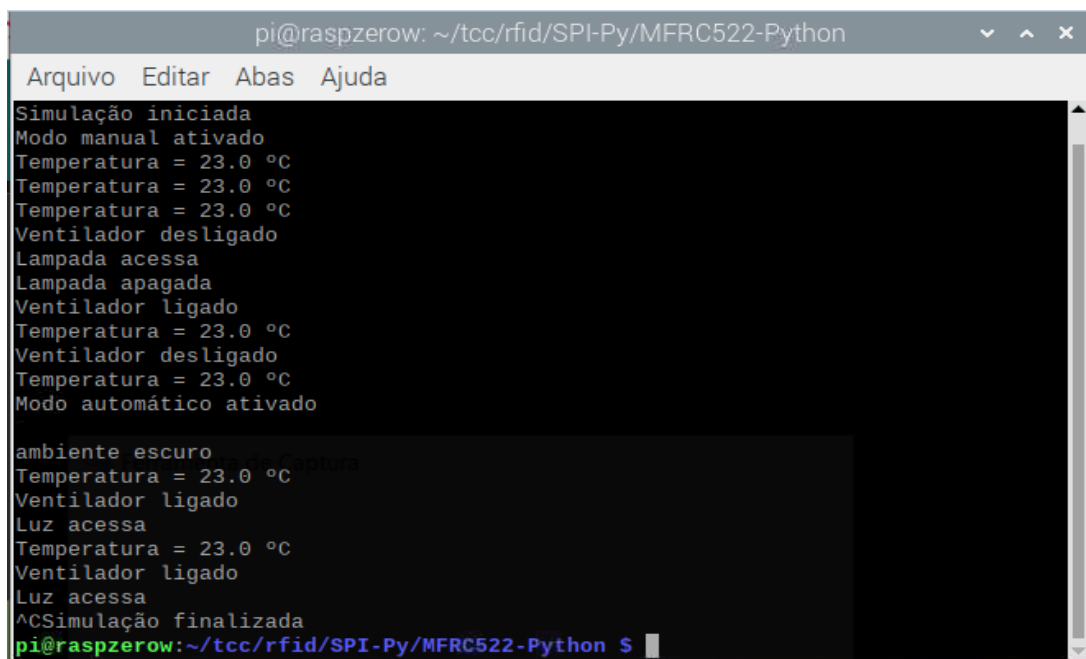
Figura 68: Aplicação web mobile utilizada para monitoramento e controle do sistema automático



Fonte: Autoria Própria

A tela de impressão do terminal Raspberry Pi, utilizada para o monitoramento do programa em execução e que imprime os estados dos processos no display LCD, é apresentada na Figura 69.

Figura 69: Terminal Raspberry Pi com monitoramento do sistema implementado no projeto



```
pi@raspzerow: ~/tcc/rfid/SPI-Py/MFRC522-Python
Arquivo  Editar  Abas  Ajuda
Simulação iniciada
Modo manual ativado
Temperatura = 23.0 °C
Temperatura = 23.0 °C
Temperatura = 23.0 °C
Ventilador desligado
Lampada acessa
Lampada apagada
Ventilador ligado
Temperatura = 23.0 °C
Ventilador desligado
Temperatura = 23.0 °C
Modo automático ativado

ambiente escuro
Temperatura = 23.0 °C
Ventilador ligado
Luz acessa
Temperatura = 23.0 °C
Ventilador ligado
Luz acessa
^CSimulação finalizada
pi@raspzerow:~/tcc/rfid/SPI-Py/MFRC522-Python $
```

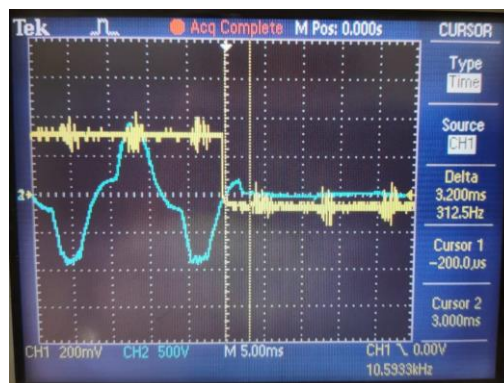
Fonte: Autoria própria

Para observar o acionamento dos equipamentos utilizados, foram obtidos em laboratório os sinais de onda de entrada e saída dos relés. As Figuras 70, 71 e 72 demonstram os sinais de onda dos relés de comando da trava elétrica, ventilação e iluminação, respectivamente. Na Figura 70, a referência do sinal terra do osciloscópio estava conectada no contato fechado da saída do relé que por sua vez não estava conectada na carga utilizada, a ponteira do canal 1 estava conectada no sinal de entrada do relé e a ponteira do canal 2 do osciloscópio estava conectada no contato de alimentação da trava elétrica ligada a uma fonte de tensão 12 Vac - 60 Hz.

Com esta configuração é possível observar que ao ser aplicada uma mudança de nível lógico no sinal de entrada do relé (sinal de onda amarelo) enviado pela placa Raspberry Pi, ocorre a alteração do seu estado de Normal Aberto para Normal Fechado, o que pode ser visto pela mudança de nível de tensão observado na saída do relé (sinal de onda em azul), ou seja, o sinal de onda de saída lido pelas ponteiras, resultando em uma onda senoidal, é interrompido e a alimentação da carga é realizada. Sendo assim, ao receber o sinal da placa, observa-se que a

mudança de estado ocorre após 3,2 milissegundos permitindo assim que a trava elétrica receba a alimentação da fonte e altere seu estado de fechada para aberta.

Figura 70: Sinais de onda de entrada e saída do módulo relé de controle da trava elétrica

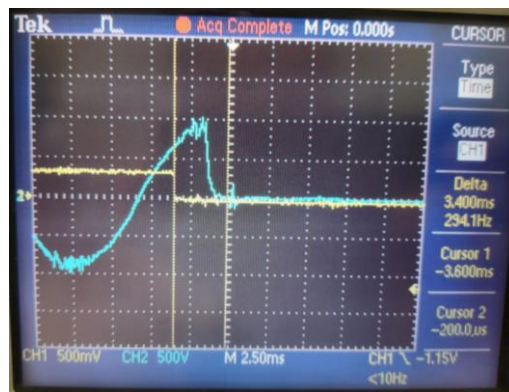


Fonte: Autoria Própria

Na Figura 71, a referência do sinal terra do osciloscópio estava conectada no contato fechado da saída do relé que por sua vez não estava conectada no equipamento utilizado, a ponteira do canal 1 estava conectada no sinal de entrada do relé e a ponteira do canal 2 do osciloscópio estava conectada no contato de alimentação do ventilador ligado a uma tomada de tensão 127 Vac - 60 Hz.

Desta maneira é possível observar que ao ser aplicada uma mudança de nível lógico no sinal de entrada do relé (sinal de onda amarelo) enviado pela placa Raspberry Pi, ocorre a alteração do seu estado de Normal Aberto para Normal Fechado, o que podemos observar pela mudança de nível de tensão visto na saída do relé (sinal de onda em azul), ou seja, o sinal de onda de saída lido pelas ponteiros, resultando em uma onda senoidal, é interrompido e a alimentação da carga é realizada. Logo, ao receber o sinal da placa, observa-se que a mudança de estado ocorre após 3,4 milissegundos permitindo assim que o ventilador receba a alimentação da tomada e altere seu estado de desligado para ligado.

Figura 71: Sinais de onda de entrada e saída do módulo relé de controle da ventilação



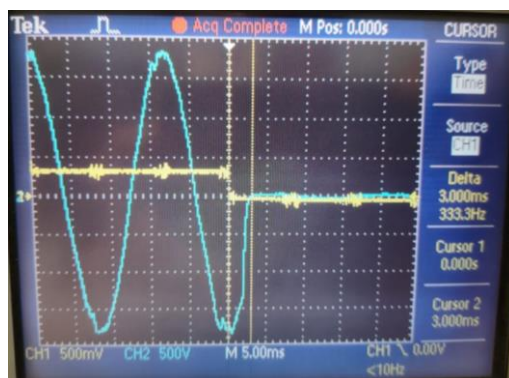
Fonte: Autoria Própria

Na Figura 72, a referência do sinal terra do osciloscópio estava conectada no contato fechado da saída do relé que por sua vez não estava conectada no equipamento utilizado, a ponteira do canal 1 estava conectada no sinal de entrada do relé e a ponteira do canal 2 do osciloscópio estava conectada no contato de alimentação do ventilador ligado a uma tomada de tensão 127 Vac - 60 Hz.

Assim, é possível observar que ao ser aplicada uma mudança de nível lógico no sinal de entrada do relé (sinal de onda amarelo) enviado pela placa Raspberry Pi, ocorre a alteração do seu estado de Normal Aberto para Normal Fechado, o que pode ser observado pela mudança de nível de tensão visto na saída do relé (sinal de onda em azul), ou seja, o sinal de onda de saída lido pelas ponteiros, resultando em uma onda senoidal, é interrompido e a alimentação da carga é realizada. Portanto, ao receber o sinal da placa, observa-se que a mudança de estado ocorre após 3 milissegundos permitindo assim que a lâmpada receba a alimentação da tomada e altere seu estado de apagada para acesa.

Na Figura 72, onde a referência do terra do osciloscópio estava no contato fechado da saída do relé e a ponteira no contato de alimentação da lâmpada ligada a tomada de tensão 127 V, observa-se que, quando aplicada uma mudança de nível lógico no sinal de entrada do relé (sinal de onda amarelo) enviado pela placa Raspberry Pi, ocorre a alteração do seu estado de normal aberto para normal fechado, o que pode ser visto pela mudança de nível de tensão observado na saída do relé (sinal de onda em azul). Assim ao receber o sinal da placa, observa-se que a mudança de estado do relé ocorre após 3 milissegundos permitindo assim que a lâmpada receba a alimentação da tomada e altere seu estado de apagada para acesa.

Figura 72: Sinais de onda de entrada e saída do módulo relé de controle da iluminação



Fonte: Autoria Própria

Através destas análises, foi possível observar que ao ser enviado o comando através da aplicação web e devido a placa Raspberry Pi estar conectada a aplicação via rede local, o tempo de envio da informação e processamento da placa é mínimo, entretanto, observou-se que o tempo de resposta dos relés, em média, é de 3,2 milissegundos e desta forma é possível concluir que o sistema de controle dos módulos relé através da nuvem possui uma boa performance e rápida resposta para o comando dos equipamentos.

A expansão e popularização de projetos de automação residencial utilizando as diversas plataformas disponíveis como, a placa Raspberry Pi ou a placa Arduino, é verificada através deste projeto, o qual conseguiu demonstrar a real acessibilidade desta tecnologia através do monitoramento da presença em ambientes e o controle automático de seu acesso; monitoramento da iluminação e ventilação e, permissão do controle destas via aplicação web, tudo com mais conforto para o usuário e de baixo custo, o que possibilita a implementação do projeto mesmo com pouco investimento.

Podemos observar este resultado analisando a Tabela 3, a qual apresenta os custos gerados para execução deste projeto.

Tabela 3: Valores gastos para implementação do projeto

Componente	Valor
Placa Raspberry Pi W Zero	R\$ 79,90
Fonte alimentação	R\$ 29,90
Adaptador mini HDMI	R\$ 12,90
Módulo RFID	R\$ 24,90
Trava elétrica	R\$ 39,90
Módulo Relé 4 canais	R\$ 23,90
LEDs	R\$ 5,80
Protoboard	R\$ 16,90
Sensor PIR	R\$ 12,90
Sensor Luminosidade	R\$ 9,90
Conectores	R\$ 21,80
Sensor umidade	R\$ 13,90
Conector cabo flat	R\$ 12,90
Fonte Chaveada	R\$ 39,90
Display LCD 16x2 I2C	R\$ 42,00
Sensor magnético <i>Reed Switch</i>	R\$ 4,00
Total	R\$ 391,40

Fonte: Autoria Própria

6. CONCLUSÃO

O presente trabalho teve como objetivo simular a automação de um ambiente, seja ele um cômodo de uma residência ou de um ambiente comercial, e, através desta simulação, com a documentação dos programas desenvolvidos, promover o conhecimento em linguagem Python, a automação residencial e o desenvolvimento web.

No desenvolvimento foram utilizadas tecnologias de código aberto, como a linguagem de programação Python e bibliotecas livres para leitura dos sensores e módulos, além da implementação do protocolo de comunicação MQTT. Também foi utilizado o hardware Raspberry Pi Zero W, de baixo custo e fácil acesso no mercado, o qual permite a execução de novas funcionalidades por meio da instalação de diferentes tipos de sensores e saídas. Por fim, o projeto cria o conceito de desenvolvimento de serviços e aplicações do Node-RED, demonstrando a acessibilidade da ferramenta e a forma como sua utilização é intuitiva.

Para a realização do trabalho foram essenciais as disciplinas de Introdução às Ciências da Computação, Sistemas Microprocessados I e II, Sistemas para Automação e Controle I e II, bem como os conhecimentos adquiridos no período da elaboração do projeto com a utilização do hardware Raspberry Pi e linguagem de programação Python.

Por meio das aplicações realizadas, foi possível verificar a funcionalidade da placa Raspberry Pi como servidor MQTT, capaz de receber mensagens dos clientes de aplicação web e traduzi-las em acionamento e controle dos processos no ambiente residencial simulado, demonstrando que é possível desenvolver um sistema automático capaz de trazer conforto em um ambiente automatizado, seja domiciliar ou comercial, de baixo custo e com desempenho, uma vez que o tempo de resposta dos relés de controle dos processos está na média de 3,2 milissegundos.

7. REFERÊNCIAS BIBLIOGRÁFICAS

- [1] **O que é a domótica?**. Sislite – Integração de Sistemas. Disponível em: <<https://www.sislite.pt/domus.htm>>. Acesso em: 15/03/2021.
- [2] **Domótica**. InfoEscola. Disponível em: <<https://www.infoescola.com/tecnologia/domotica/>>. Acesso em: 15/03/2021
- [3] **SparkFun Raspberry Pi Zero W Basic Kit**. Mouser. Disponível em: <<https://br.mouser.com/new/sparkfun/sparkfun-pi-zero-w-basic-kit/>>. Acesso em 15/03/2021.
- [4] **Sensor de Umidade e Temperatura DHT11**. FilipeFlop. Disponível em <<https://www.filipeflop.com/produto/sensor-de-umidade-e-temperatura-dht11/>>. Acesso em: 07/07/2021.
- [5] Adilson Thomsen. **Monitorando Temperatura e Umidade com sensor DHT11**. FilipeFlop. Disponível em <<https://www.filipeflop.com/blog/monitorando-temperatura-e-umidade-com-o-sensor-dht11/>>. Acesso em: 16/07/2021
- [6] **Sensor de Luminosidade (LDR)**. FilipeFlop. Disponível em: <<https://www.filipeflop.com/produto/sensor-de-luz-ldr/>> Acesso em: 07/07/2021.
- [7] **Sensor de movimento presença (PIR)**. FilipeFlop. Disponível em: <<https://www.filipeflop.com/produto/sensor-de-movimento-presenca-pir/>>. Acesso em: 07/07/2021.
- [8] Arduino e Cia. **Como usar um sensor de presença PIR com Arduino**. Arduino e Cia. Disponível em: <<https://www.arduinoecia.com.br/sensor-presenca-arduino-modulo-pir-dyp-me003/>>. Acesso em 16/07/2021.
- [9] **Kit Módulo Leitor RFID – MFRC522**. FilipeFlop. Disponível em: <<https://www.filipeflop.com/produto/kit-modulo-leitor-rfid-mfrc522-mifare/>>. Acesso em: 07/07/2021.
- [10] **Módulo Relé 5V 4 Canais**. FilipeFlop. Disponível em: <<https://www.filipeflop.com/produto/modulo-rele-5v-4-canais/>>. Acesso em: 07/07/2021.
- [11] **Minitrava Elétrica Solenoide 12V**. FilipeFlop. Disponível em: <<https://www.filipeflop.com/produto/mini-trava-eletrica-solenoide-12v/>>. Acesso em: 07/07/2021.

- [12] Lucas Carvalho. **Raspberry Pi: o que é, para que serve e como comprar**. Olhar Digital; Disponível em: < <https://olhardigital.com.br/noticia/Raspberry-pi-o-que-e-para-que-serve-e-como-comprar/82921>>. Acesso em: 25/08/2021.
- [13] Shirley Basilio, **O que é Node-RED? Conhecendo e instalando**. MasterWalker Blog. Disponível em: <<http://blogmasterwalkershop.com.br/outros/o-que-e-node-red-conhecendo-e-instalando/>>. Acesso em: 25/08/2021.
- [14] Neto Oliveira, **Por que Node-RED para IOT?**. Medium. Disponível em: <<https://medium.com/@netoolii/por-que-node-red-para-iot-41a4ab170c56>>. Acesso em: 30/10/2021.
- [15] Michael Yuan. **Conhecendo o MQTT**. IBM. Disponível em: <<https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>>. Acesso em: 25/08/2021.
- [16] Martins, V. F. **Automação residencial usando protocolo mqtt, nodered e mosquitto broker com esp32 e esp8266**. 2019. 53 f. Trabalho de Graduação (Bacharelado em Engenharia de Controle e Automação). Universidade Federal de Uberlândia, Uberlândia, 2019.
- [17] OpenJs Foundation & Contributors. **Node-RED**. Node-Red. Disponível em: <<https://nodered.org>>. Acesso em 16/07/2021.
- [18] Shahbaz Ahmed, **Controlling an External LED using a Raspberry Pi and GPIO pins**. freeCodeCamp. Disponível em: <<https://www.freecodecamp.org/news/hello-gpio-blinking-led-using-Raspberry-pi-zero-wh-65af81718c14/>>. Acesso em 25/08/2021
- [19] **Placa Raspberry Pi Zero W C/ Wifi Bluetooth Mini Pc - semi novo**. Balão da Informática. Disponível em: < <https://www.balaodainformatica.com.br/placa-Raspberry-pi-zero-w-c-wifi-bluetooth-mini-pc-semi-novo>>. Acesso em 10/12/2021.
- [20] Andrade, G.P.C.. Rinaldi, L.C.. **O uso da domótica na construção de um sistema mobile embarcado para conforto nas rotinas diárias**. Simpósio de Iniciação Científica, Didática e de Ações Sociais da FEI, VIII, 2018, São Bernardo do Campo.
- [21] **Aprenda a instalar o MQTT (Mosquitto) no seu Raspberry PI**. PPLWARE. Disponível em: < <https://pplware.sapo.pt/tutoriais/mqtt-no-seu-Raspberry-pi/>>. Acesso em 10/12/2021

- [22] **Reed Switch Chave Magnética.** FilipeFlop. Disponível em: <<https://www.filipeflop.com/produto/reed-switch-chave-magnetica/>> Acesso em: 07/07/2021.
- [23] **Display LCD 16x2 I2C Backlight Azul.** FilipeFlop. Disponível em: <<https://www.filipeflop.com/produto/display-lcd-16x2-i2c-backlight-azul/>> Acesso em: 07/07/2021.
- [24] Raspberry Pi Foundation. **Raspberry Pi.** Raspberry Pi Foundation. Disponível em: <<https://www.raspberrypi.com/>>. Acesso em 16/07/2021.
- [25] Sartori, R. V. L. **Domótica – acionamento de dispositivos residenciais via smartphone utilizando arduino e comunicação bluetooth.** 2016. 57 f. Trabalho de Graduação (Bacharelado em Engenharia de Controle e Automação). Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Ciência e Tecnologia, Sorocaba, 2016.
- [26] Dias, G. D. **Automação das etapas de mosturação e clarificação do processo de brassagem na produção de cerveja artesanal.** 2020. 83 f. Trabalho de Graduação (Bacharelado em Engenharia de Controle e Automação). Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Ciência e Tecnologia, Sorocaba, 2020.
- [27] Quinaglia, A. C. **Monitoramento, com armazenamento de dados em nuvem, das variáveis de ambientes de Estoque de peças aeronáuticas.** 2019. 55 f. Trabalho de Graduação (Bacharelado em Engenharia de Controle e Automação). Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Ciência e Tecnologia, Sorocaba, 2019.

8. ANEXOS

ANEXO A – PROGRAMAÇÃO DO SISTEMA DE AUTOMAÇÃO RESIDENCIAL EM LINGUAGEM PYTHON

```
1 #!/usr/bin/ env python
2
3 #Import de bibliotecas#
4 import paho.mqtt.client as mqtt #Importa biblioteca
   para utilização do protocolo de comunicação MQTT
5 import RPi.GPIO as GPIO #Importa a bibliote de
   utilização da pinagem GPIO
6 import sys #Importa biblioteca que utiliza funções do
   próprio interpretador Python
7 import time #Importa biblioteca time para utilização
   de funções baseadas em tempo
8 import Adafruit_DHT #importa a biblioteca do sensor
   DHT11
9 import MFRC522 #importa a biblioteca utilizada para
   realizar as leituras do módulo RFID
10 from rpi_lcd import LCD
11 #Configuração das entradas e saídas do circuito#
12
13 GPIO.setwarnings(False) #Desativa avisos referentes
   as portas GPIO
14 GPIO.setmode(GPIO.BOARD) #Define o modo de utilização
   da pinagem da GPIO
15 pino_presenca = 11 #Define o pino da GPIO que recebe
   o sinal do sensor de presença
16 GPIO.setup(pino_presenca, GPIO.IN) #Define o pino 11
   da GPIO como entrada digital
17 pino_luminosidade = 15 #Define o pino da GPIO que
   recebe o sinal do sensor de luminosidade
18 GPIO.setup(pino_luminosidade, GPIO.IN) #Define o pino
   15 da GPIO como entrada digital
19 pino_temperatura = 27 #Define a entrada GPIO que
   recebe o sinal do sensor de temperatura
20 pino_sensorabertura = 32 #Define a entrada GPIO que
   recebe o sinal do sensor de abertura magnética
21
22 #Define os LEDs de indicação de estado de leitura dos
   sensores#
23 GPIO.setup(12, GPIO.OUT) #Define o pino de saída do
   LED para indicar modo automático
24 GPIO.output(12, GPIO.LOW) #Define o estado inicial do
   LED
```

```
25 GPIO.setup(16, GPIO.OUT) #Define o pino de saída do
    LED para indicar modo manual
26 GPIO.output(16, GPIO.LOW) #Define o estado inicial do
    LED
27 GPIO.setup(29, GPIO.OUT) #Define o pino de saída do
    LED para indicar porta aberta
28 GPIO.output(29, GPIO.LOW) #Define o estado inicial do
    LED
29 GPIO.setup(31, GPIO.OUT) #Define o pino de saída do
    LED para indicar porta fechada
30 GPIO.output(29, GPIO.LOW) #Define o estado inicial do
    LED
31
32 #Define as saídas de controle dos reles
33 GPIO.setup(36, GPIO.OUT) #Define o pino de saída do
    sinal enviado ao rele de controle da trava eletrica
34 GPIO.output(36, GPIO.HIGH) #Define o estado inicial
    da saída
35 GPIO.setup(38, GPIO.OUT) #Define o pino de saída do
    sinal enviado ao rele de controle da ventilação
36 GPIO.output(38, GPIO.HIGH) #Define o estado inicial
    da saída
37 GPIO.setup(40, GPIO.OUT) #Define o pino de saída
    enviado ao rele de controle da iluminação
38 GPIO.output(40, GPIO.HIGH) #Define o estado inicial
    da saída
39
40 #Define as variáveis globais do programa
41 estado_lampada = 0 #Define e atribui a variavel de
    estado da lampada o pino 40 da placa
42 estado_ventilador = 0 #Define e atribui a variavel de
    estado do ventilador
43 modo_auto = False #Inicializa a variavel modo_auto
    com valor 0
44 temperatura_setup = 22 #Inicializa a variavel de
    setup da temperatura para controle automático
45 cartao = {"12:4F:90:34:F9":"Matheus Neri", "2A:CD:32:
    16:C3":"Matheus Neri"} #Define as tags com acesso
    liberado
46 cartao_controle = {"47:22:9D:60:98":"Filho 1", "F9:F:
    C5:7E:4D":"Filho2"} #Define as tags de identificação
```

```
46 para controle de acesso
47 leitor = MFRC522.MFRC522() #Define a variavel leitor
    como leitor do módulo RFID
48 lcd = LCD() #Define a variavel lcd para controle do
    display LCD através da biblioteca rpi_lcd
49
50 #Define função que executará o programa em modo
    automático
51 def automatico():
52     global leitor #Define a variavel leitor como
        global para utilização na subrotina
53     global estado_ventilador #Define a variavel de
        estado do ventilador como global para utilização na
        subrotina
54     global estado_lampada #Define a variavel de
        estado do ventilador como global para utilização na
        subrotina
55     global modo_auto #Define a variavel modo_auto
        como global para utilização na subrotina
56     global estado_luminosidade #Declara a variavel
        estado_de luminosidade como variavel global
57     global temperatura_setup #Define a variavel de
        temperatura_setup como global para utilização na
        subrotina
58     global pino_luminosidade #Define a variavel do
        pino do sensor de luminosidade como global para
        utilização na subrotina
59     GPIO.setup(pino_luminosidade, GPIO.IN) #Define o
        pino 15 como entrada digital dentro da subrotina
60     estado_luminosidade = GPIO.input(
        pino_luminosidade) #Atribui a variavel de
        estado_luminosidade o valor de leitura do sensor LDR
61     print ("Modo automático ativado") #Imprime na
        tela do terminal que o modo automatico foi acionado
62     lcd.clear() #Limpa o display LCD
63     lcd.text("Modo auto ativ", 1) #Escreve na
        primeira linha do display que o modo automatico foi
        ativado
64     GPIO.output(12, GPIO.HIGH) #Define nivel logico
        alto no pino de saída que aciona o LED de indicação
        de modo automatico selecionado
```

```

65     GPIO.output(16,GPIO.LOW) #Define nível logico
    baixo no pino de saída que aciona o LED de indicação
    de modo manual selecionado
66     while modo_auto == True: #Define laço de
    repetição do programa
67         estado_presenca = GPIO.input(pino_presenca)
    #Atribui a variavel estado_presenca o valor de
    leitura do sensor PIR
68         if estado_presenca == 1: #Cria condicional
    que verifica a leitura do sensor PIR
69             modo_auto = True #Caso valor no sensor
    = 1, atribui valor True a variavel modo_auto
70             while modo_auto == 1: #Define laço de
    repetição do programa caso modo_auto = 1
71                 status,tag_type = leitor.
    MFRC522_Request(leitor.PICC_REQIDL) #Faz a leitura
    da tag
72                 if status == leitor.MI_OK: #Checa se
    o status de leitura é OK
73                     print ("Cartao detectado") #
    Imprime no terminal que a tag foi detectado
74                     lcd.text("Cartao detectado",2) #
    Imprime na segunda linha do display LCD que a tag
    foi detectada
75                     status, uid = leitor.
    MFRC522_Anticoll() #Faz a leitura da uid da tag
76                     if status == leitor.MI_OK: #
    Checa se a leitura é OK
77                         uid = ':'.join(['%X' % x for
    x in uid]) #Compacta as informações da tag para
    impressão
78                         print ("UID do cartao: %s"
    % uid) #Imprime no terminal o valor da uid da tag
79                         if uid in cartao: #Verifica
    se a UID da tag está contida na lista de tags
    liberadas
80                             print ("Acesso Liberado"
    ) #Em caso positivo imprime no terminal que o acesso
    foi liberado
81                             print ("Olá %s." %
    cartao[uid]) #Imprime nome do usuário da tag

```

```

82         lcd.clear() #Limpa o
display LCD
83         lcd.text("Olá %s." %
cartao[uid],1) #Imprime na primeira linha o nome do
usuario
84         lcd.text("Acesso
Liberado",2) #Imprime na segunda linha "Acesso
liberado"
85         time.sleep(3) # Aguarda
3 segundos
86         acesso() #Chama a função
que libera o acesso da porta
87         if uid in cartao_controle: #
Verifica se a tag está na lista de tags com controle
de acesso
88         print ("Acesso Liberado"
) #Em caso positivo imprime no terminal que o acesso
está liberado
89         print ("Olá %s." %
cartao_controle[uid]) #Imprime o nome do usuário da
tag
90         nome = ("%s. acabou de
entrar em casa" %cartao_controle[uid]) #Define
variavel com mensagem a ser enviada ao client MQTT
91         lcd.clear() # Limpa o
display LCD
92         lcd.text("Olá %s." %
cartao[uid], 1) # Imprime na primeira linha o nome
do usuario
93         lcd.text("Acesso
Liberado", 2) # Imprime na segunda linha "Acesso
liberado"
94         time.sleep(3) # Aguarda
3 segundos
95         acesso() #Chama a função
que libera o acesso da porta
96         client.publish('
controle_acesso', payload=nome) #Publica no client
MQTT (serviço web) a mensagem de que o usuário com
controle de acesso adentrou o cômodo
97         else: #Em caso negativo da

```

```

97 verificação das tags
98                                     print ("Acesso negado")
    #Imprime no terminal Acesso negado
99                                     print ("UID do cartao: %
    s" % uid) #Imprime no terminal a uid da tag
100                                     lcd.clear() # Limpa o
    display LCD
101                                     lcd.text("Acesso Negado"
    , 1) # Imprime na segunda linha "Acesso negado"
102                                     GPIO.output(36, GPIO.
    HIGH) #Envia nivel logico alto para rele de controle
    da trava eletrica, mantendo trava fechada
103                                     time.sleep(1) #Aguarda 1
    segundo
104                                     estado_luminosidade = GPIO.input(
    pino_luminosidade) #Atribui a variavel
    estado_luminosidade o valor de leitura do sensor LDR
105                                     estado_presenca = GPIO.input(
    pino_presenca) #Atribui a variavel estado_presença o
    valor de leitura do sensor PIR
106                                     if estado_luminosidade == 1: #
    Verifica se valor de leitura do sensor LDR = 1
107                                     print ("ambiente escuro") #Caso
    valor da leitura = 1, imprime no terminal que o
    ambiente está escuro
108                                     lcd.text("Ambiente escuro", 2) #
    Imprime na segunda linha do display LCD o estado do
    ambiente
109                                     GPIO.output(40, GPIO.LOW) #Envia
    sinal baixo para rele de controle da iluminação que
    irá acender a lâmpada
110                                     if estado_luminosidade == 0: #
    Verifica se valor de leitura do sensor LDR = 0
111                                     print ("Luz acesa") #Em caso
    positivo imprime no terminal que a luz está acesa
112                                     lcd.text ("Luz acesa")
113                                     temp_atual = temperatura() #Chama
    função que realiza leitura do sensor DHT11
114                                     if temp_atual > temperatura_setup:#
    Verifica se temperatura atual do ambiente é maior
    que temperatura de setup

```

```

115         estado_ventilador = 1 #Em caso
        positivo, atribui valor 1 a variavel
        estado_ventilador
116         ventilador_automatico() #Chama
        função de acionamento do ventilador no modo
        automatico
117         if temp_atual < temperatura_setup
        and estado_ventilador == 1: #Verifica se temperatura
        atual é menor que temperatura de setup e se
        ventilador está ligado
118         GPIO.output(38, GPIO.HIGH) #Em
        caso positivo das condições acima, desliga
        ventilador
119         if estado_presenca == 0: #Verifica
        se leitura do sensor de presença é igual 0 (sem
        presença)
120         modo_auto = False #Atribui
        estado false para variavel modo_auto
121         if estado_luminosidade == 0 and
        estado_presenca == 0: #Verifica se luz está acessa e
        se não há presença no ambiente
122         GPIO.output(40, GPIO.HIGH) #Em caso
        positivo, apaga a luz
123         if estado_presenca == 0 and estado_ventilador
        == 1: #Verifica se não há presença e se ventilador
        está ligado
124         GPIO.output(38, GPIO.HIGH) #Desliga
        ventilador
125         print ("Sem presença") #Imprime no terminal
        que leitura do sensor não detectou presença
126         lcd.clear() #Limpa o display LCD
127         lcd.text("Sem presença", 2) #Imprime na
        segunda linha do display que não há presença
        detectada
128         time.sleep(3) #Aguarda 3 segundo para
        continuar execução do programa
129
130 #Define rotina de execução do programa em modo
        manual
131 def manual():
132     global modo_auto #Define variavel modo_auto como

```

```

132  global para utilização na subrotina
133      try: #Define rotina de execução
134          print ("Modo manual ativado") #Imprime que
            modo manual foi ativado
135          lcd.text("Modo manual ativ", 1) #Imprime na
            primeira linha do display que o modo manual foi
            ativado
136          GPIO.output(16,GPIO.HIGH) #Acende LED que
            indica modo manual selecionado
137          GPIO.output(12, GPIO.LOW) #Apaga LED que
            indica modo automatico selecionado
138          leitor = MFRC522.MFRC522() #Define a
            variavel como leitor RFID
139          estado_luminosidade = GPIO.input(
            pino_luminosidade)
140          while modo_auto == False: #Enquanto
            modo_auto é igual a Falso
141              status, tag_type = leitor.
            MFRC522_Request(leitor.PICC_REQIDL) #Realiza leitura
            da tag
142              if status == leitor.MI_OK: #Verifica se
            tag foi detectada
143                  print ("Cartao detectado") #Imprime
            no terminal que tag foi detectada
144                  lcd.text("Cartao detectado", 2) #
            Imprime na segunda linha do display LCD que a tag
            foi detectada
145                  status, uid = leitor.
            MFRC522_Anticoll() #Faz leitura da uid da tag
146                  if status == leitor.MI_OK: #Verifica
            se leitura foi realizada
147                      uid = ":".join(["%X" % x for x
            in uid]) #Compacta as informações da tag
148                      print ("UID do cartao: %s" % uid
            ) #Imprime no terminal a uid da tag
149                      if uid in cartao: #Verifica se
            uid da tag está na lista de tags liberadas
150                          print ("Acesso liberado") #
            imprime no terminal que o acesso foi liberado
151                          print ("Olá %s." % cartao[
            uid]) #Imprime nome do usuário da tag

```

```

152                                lcd.clear()  # Limpa o
display LCD
153                                lcd.text("Olá %s." % cartao[
uid], 1)  # Imprime na primeira linha o nome do
usuario
154                                lcd.text("Acesso Liberado",
2)  # Imprime na segunda linha "Acesso liberado"
155                                time.sleep(3) #Aguarda 3
segundos
156                                acesso() #Chama a função que
libera o acesso da porta
157                                if uid in cartao_controle: #
Verifica se uid da tag está na lista de tags no
controle de acesso
158                                print ("Acesso liberado") #
Imprime acesso liberado
159                                print ("Olá %s." %
cartao_controle[uid]) #Imprime nome do usuário da
tag
160                                nome = ("%s. acabou de
entrar em casa" % cartao_controle[uid]) #Define
variavel que recebe mensagem a ser enviado via MQTT
161                                lcd.clear()  # Limpa o
display LCD
162                                lcd.text("Olá %s." % cartao[
uid], 1)  # Imprime na primeira linha o nome do
usuario
163                                lcd.text("Acesso Liberado",
2)  # Imprime na segunda linha "Acesso liberado"
164                                time.sleep(3) #Aguarda 3
segundos
165                                acesso() #Chama a função que
libera o acesso da porta
166                                client.publish ("
controle_acesso", payload=nome) #Publica via
protocolo MQTT a mensagem de que usuario adentrou o
cômodo no tópico "controle_acesso"
167                                else: #Caso nao tenha acesso
liberado
168                                print ("Acesso negado") #
Imprime no terminal que acesso nao foi liberado

```

```

169             lcd.clear() # Limpa o
display LCD
170             lcd.text("Acesso Negado", 1
) # Imprime na segunda linha "Acesso negado"
171             GPIO.output(36, GPIO.HIGH) #
Define estado logico alto para manter trava fechada
172             time.sleep(1) #Aguarda 1 segundo
173             temperatura () #Chama função para
realizar leitura da temperatura
174     except KeyboardInterrupt: #Define método de
interrupção da execução do programa
175         GPIO.cleanup() #Limpa a configuração das
entradas e saídas da placa Raspberry Pi
176         print ("Simulação finalizada") #Imprime no
terminal que o programa finalizou sua execução
177         lcd.clear() #Limpa o display LCD
178         lcd.backlight(False) #Apaga a luz de fundo
do display LCD
179         sys.exit(0) #Realiza ação de finalizar
programa
180
181 #Define função de conexão ao Broker MQTT e
assinatura aos tópicos de mensagem MQTT
182 def on_connect (client, userdata, flags, rc):
183     client.subscribe("acesso") #Assina broker ao
tópico "acesso"
184     client.subscribe("iluminação") #Assina broker ao
tópico de iluminação
185     client.subscribe("ventilação") #Assina broker ao
tópico de ventilação
186     client.subscribe("modo") #Assina broker ao
tópico de seleção de modo de operação
187     client.subscribe("temperatura_setup") #Assina
broker ao tópico de definição da temperatura setup
188
189 #Define função que recebe as mensagens do cliente
que publica no tópicos assinados
190 def on_message(client, userdata, msg):
191     msg.payload = msg.payload.decode("utf-8") #
Formata a mensagem recebida
192     mensagem = msg.payload #atribui mensagem

```

```
192 recebida via MQTT a variavel mensagem
193     global estado_lampada #Define estado da lampada
        como variavel global para utilização na subrotina
194     global estado_ventilador #Define estado do
        ventilador como variavel global para utilização na
        subrotina
195     global temperatura_setup #Define temperatura
        setup como variavel global para utilização na
        subrotina
196     global modo_auto #Define variavel modo_auto como
        variavel global para utilização na subrotina
197     if (msg.topic == "temperatura_setup"): #Verifica
        se o topico da mensagem enviada é o topico de setup
        da temperatura
198         if modo_auto == True: #Verifica o estado da
            variavel modo_auto
199             modo_auto = True #Mantem o valor da
                variavel modo_auto
200         if modo_auto == False: #Verifica o estado da
            variavel modo_auto
201             modo_auto = False #Mantem o valor da
                variavel modo_auto
202         temperatura_setup = mensagem #Atribui a
            variavel de setup da temperatura o valor enviado
            pelo client via MQTT
203         temperatura_setup = int (temperatura_setup)
            #Torna o valor da mensagem em tipo inteiro
204         temp_lcd = ("Temperatura setup definida como
            %s. °C" % temperatura_setup) #Configura a mensagem
            a ser exibida no display LCD
205         print ("Temperatura setup definida como %s.
            °C" % temperatura_setup) #Imprime no terminal o
            valor da temperatura definida
206         lcd.text(temp_lcd, 2) #Imprime na segunda
            linha do display LCD a temperatura de setup definida
207         if mensagem == "ON": #Verifica se mensagem
            recebida é igual a ON
208             modo_auto = True #Atribui valor True a
                variavel de seleção de modo
209         if mensagem == "OFF": #Verifica se mensagem
            recebida é igual a OFF
```

```
210         modo_auto= False #Atribui valor False a
        variavel de seleção de modo
211         if mensagem == "ONL": #Verifica se mensagem
        recebida é igual a ONL
212             if estado_lampada == 1: #Em caso positivo,
        verifica o estado da lampada
213                 estado_lampada = 0 #Caso lampada esteja
        acessa, atribui valor 0 a variavel de estado da
        lampada
214                 lampada() #Chama função que controla o
        acionamento do rele de controle da lampada
215         if mensagem == "OFFL": #Verifica se mensagem
        recebida é igual a OFFL
216             if estado_lampada == 0: #Em caso positivo,
        verifica estado da lampada
217                 estado_lampada = 1 #Caso lampada esteja
        apagada, atribui valor 1 a variavel de estado da
        lampada
218                 lampada() #Chama função que controla o
        acionamento do rele de controle da lampada
219         if mensagem == "ONV": #Verifica se mensagem
        recebida é igual a ONV
220             if estado_ventilador == 1: #Em caso positivo
        , verifica o estado do ventilador
221                 estado_ventilador = 0 #Caso ventilador
        esteja ligado, atribui valor 0 a variavel de estado
        do ventilador
222                 ventilador() #Chama função que controla o
        acionamento do rele de controle do ventilador
223         if mensagem == "OFFV": #Verifica se mensagem
        recebida é igual a OFFV
224             if estado_ventilador == 0: #Em caso positivo
        , verifica o estado do ventilador
225                 estado_ventilador = 1 #Caso ventilador
        esteja desligado, atribui valor 1 a variavel de
        estado do ventilador
226                 ventilador() #Chama função que controla o
        acionamento do rele de controle do ventilador
227         if mensagem == "ONP": #Verifica se mensagem
        recebida é igual a ONP
228         acesso() #Em caso positivo, chama função que
```

```

228  controla o acionamento do rele de controle da trava
    eletrica
229
230 #Define função de controle manual da trava eletrica
231 def acesso ():
232     global modo_auto
233     if modo_auto == True: #Verifica o estado da
        variavel modo_auto
234         modo_auto = True #Mantem o valor da
            variavel modo_auto
235     if modo_auto == False: #Verifica o estado da
        variavel modo_auto
236         modo_auto = False #Mantem o valor da
            variavel modo_auto
237     global pino_sensorabertura
238     GPIO.setup(32, GPIO.IN, pull_up_down = GPIO.
        PUD_UP) #Configura o pino 32 como entrada digital
239     sensor_abertura = GPIO.input(pino_sensorabertura
        ) #Define a variavel sensor de abertura com a
        entrada GPIO 32
240     while sensor_abertura == 1: #Define laço de
        repetição
241         GPIO.output(29, GPIO.HIGH) #Define nivel
            logico alto no LED que indica porta aberta
242         GPIO.output(31, GPIO.LOW) #Define nivel
            logico baixo no LED que indica porta fechada
243         GPIO.output(36, GPIO.LOW) #Define nivel
            logico baixo na trava elétrica e abre a porta
244         print ("Porta aberta") #Imprime no terminal
            que a porta foi aberta
245         lcd.clear() #Limpa o display LCD
246         lcd.text("Porta aberta",1) #Imprime na
            primeira linha do displav LCD que a porta foi aberta
247         time.sleep(2) #Aguarda 2 segundos
248         sensor_abertura = GPIO.input(
            pino_sensorabertura) #Define a variavel sensor de
            abertura com a entrada GPIO 32
249     else:
250         print("Porta fechada")
251         lcd.clear() #Limpa o display LCD
252         lcd.text("Porta fechada", 1) # Imprime na

```

```

252 primeira linha do displau LCD que a porta foi aberta
253     GPIO.output(29, GPIO.LOW) #Define nivel
    logico baixo no LED que indica porta aberta
254     GPIO.output(31, GPIO.HIGH) #Define nivel
    logico alto no LED que indica porta fechada
255     GPIO.output(36, GPIO.HIGH) #Define nivel
    logico balto na trava elétrica e fecha a porta
256
257 #Define função de acionamento automatico do
    ventilador
258 def ventilador_automatico():
259     print("Ventilador ligado") #Imprime no
    terminal que ventilador foi ligado
260     lcd.text("Ventilador ligado", 1) # Imprime
    na primeira linha do display LCD o estado do
    ventilador
261     GPIO.output(38, GPIO.LOW) #Define nivel
    logico baixo e liga ventilador
262     estado_ventilador = 1 #Atribui valor 1 a
    variavel de estado do ventilador
263
264 #Define função de controle manual do ventilador
265 def ventilador():
266     GPIO.setup (38, GPIO.OUT) #Define pino 38 como
    saída digital
267     global estado_ventilador #define variavel do
    estado do ventilador como global para utilização na
    subrotina
268     if estado_ventilador == 0: #Verifica valor do
    estado do ventilador
269         print ("Ventilador ligado") #Caso valor = 0
    imprime no terminal que ventilador está ligado
270         lcd.text("Ventilador ligado",1) #Imprime na
    primeira linha do display LCD o estado do ventilador
271         GPIO.output(38,GPIO.LOW) #Define nivel
    logico baixo e liga ventilador
272         estado_ventilador = 1 #Atribui valor 1 a
    variavel do estado ventilador
273     else: #Em caso de valor ser diferente de 0
274         print ("Ventilador desligado") #Imprime no
    terminal que o ventilador está desligado

```

```

275         lcd.text("Ventilador desligado", 1) #
        Imprime na primeira linha do display LCD o estado do
        ventilador
276         GPIO.output(38, GPIO.HIGH) #Define nivel
        logico alto e desliga ventilador
277         estado_ventilador = 0 #Atribui valor 0 a
        variavel do estado ventilador
278
279 #Define função de controle manual da iluminação
280 def lampada():
281     GPIO.setup(40, GPIO.OUT) #Define pino 40 como
        saída digital
282     global estado_lampada #define variavel do estado
        da lampada como global para utilização na subrotina
283     if estado_lampada == 0: #Verifica valor do
        estado da lampada
284         GPIO.output(40, GPIO.LOW) #Define nivel
        logico baixo e acende lampada
285         print ("Lampada acesa") #Imprime no terminal
        que a lampada esta acesa
286         lcd.text("Lampada acesa", 1) # Imprime na
        primeira linha do display LCD o estado da lampada
287         estado_lampada = 1 #Atribui valor 1 a
        variavel do estado lampada
288     else: #Em caso de valor ser diferente de 0
289         GPIO.output(40,GPIO.HIGH) #Define nivel
        logico baixo e apaga lampada
290         print ("Lampada apagada") #Imprime no
        terminal que lampada esta apagada
291         lcd.text("Lampada apagada", 1) # Imprime na
        primeira linha do display LCD o estado da lampada
292         estado_lampada = 0 #Atribui valor 0 a
        variavel do estado lampada
293
294 #Define função que realiza leitura do sensor de
        temperatura DHT11
295 def temperatura():
296     sensor = Adafruit_DHT.DHT11 #Atribui a variavel
        sensor o tipo de sensor da biblioteca DHT11
297     pino_sensor = 27 #Atribui o valor do pino ao
        qual o sensor está ligado para utilização na função

```

```

297 de leitura
298     umid, temp = Adafruit_DHT.read_retry(sensor,
    pino_sensor) #Define as variaveis umid e temp como
    entradas do sensor DHT11
299     if umid is not None and temp is not None: #
    Verifica se as variaveis nao estão vazias
300         print ("Temperatura = {0:0.1f} °C".format(
    temp)) #Em caso das variaveis possuirem valor,
    imprime o valor de temperatura
301         temp_lcd = "Temp: {0:0.1f} C".format(temp)
302         temperatura_atual = ("{0:0.1f} °C".format(
    temp)) #Atribui e formata a variavel
    temperatura_atual o valor de temperatura lido pelo
    sensor
303         client.publish ("temperatura", payload =
    temperatura_atual) #Publica no topico "temperatura
    " o valor da leitura do sensor para monitoramento da
    temperatura via serviço web
304         lcd.text(temp_lcd,1) #Imprime na primeira
    linha do display LCD a temperatura do ambiente
305         if umid is None and temp is None: #Em caso das
    variaveis estarem vazias
306             print ("Falha ao ler dados do DHT11") #
    Imprime no terminal que houve falha na leitura do
    sensor
307             lcd.clear() #Limpa o display LCD
308             lcd.text("Falha sensor",1) #Imprime na
    primeira linha do display LCD que houve erro de
    leitura do sensor
309         return temp #Retorna o valor da temperatura
    medida dentro da função
310
311 #Define programa principal
312 try:
313     client = mqtt.Client() #Declara a variavel
    client e define como objeto da biblioteca mqtt
314     client.on_connect = on_connect #Realiza a
    conexão com o broker
315     client.on_message = on_message #Realiza o
    monitoramento das mensagens enviadas aos tópicos de
    assinatura

```

```
316     client.connect("raspzerow.local", port = 1883,
    keepalive =120) #Realiza configuração do servidor,
    porta e conectividade do broker MQTT
317     client.username_pw_set("root", "neri2143") #
    Define as credenciais de segurança do serviço de
    comunicação MQTT
318     client.loop_start() #Inicia loop de conexão do
    broker MQTT
319     print ("Simulação iniciada") #imprime no
    terminal que o programa iniciou sua execução
320     lcd.text("Iniciando...",1) #Imprime na primeira
    linha do display LCD que o programa está
    inicializando
321     while True: #Cria laço de repetição que mantém
    programa em execução
322         if modo_auto: #Verifica se estado da
    variavel modo_auto é igual a True
323             automatico() #Em caso positivo chama
    função que dá inicio ao modo automático
324         else: #Em caso negativo
325             manual() #chama função que dá inicio ao
    modo manual
326 except KeyboardInterrupt:#Define método de
    finalização do programa
327     GPIO.cleanup() #Limpa as configurações de
    entradas e saídas da placa Raspberry Pi
328     print ("Simulação finalizada") #Imprime na tela
    do terminal que o programa finalizou sua execução
329     lcd.clear() #Limpa o display LCD
330     lcd.text("Finalizando...", 1) #Imprime no
    display LCD que o programa foi finalizado
331     lcd.clear() #Limpa o display LCD
332     lcd.backlight(False) #Apaga a luz de fundo do
    display LCD
333     sys.exit(0) #Finaliza execução do programa
334
```