Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Projeto de um controle remoto universal infravermelho com suporte a comandos de voz

Lucas Nascimento Santos Souza

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. José Edil Guimarães de Medeiros

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Projeto de um controle remoto universal infravermelho com suporte a comandos de voz

Lucas Nascimento Santos Souza

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. José Edil Guimarães de Medeiros (Orientador)
ENE/UnB

Prof. Ricardo Zelenovsky
ENE/UnB

Prof. Daniel Chaves Café
ENE/UnB

Prof. João José Costa Gondim
Coordenador do Curso de Engenharia da Computação

Brasília, 24 de Maio de 2021

Dedicatória

Eu dedicaria este trabalho primeiramente aos meus pais e meu irmão, que são as pessoas mais importantes da minha vida. Também dedicaria aos meus amigos de infância e amigos que conheci durante os estudos. Não deixaria de dedicar aos professores que me guiaram no decorrer do curso. Porém infelizmente, essa dedicatória não será da forma como eu tinha idealizado. Dedico este trabalho para meu querido avô materno Hélio e para minha amada avó materna Aparecida, carinhosamente chamada de Cida, que foi vítima de COVID-19 neste ano de 2021.

"...Her grandmother was gone. Kainé blinked, trying to feel the hands which had been in hers just a moment before. For a fleeting instant, she could remember the warmth of that embrace, the trembling of the fingers, but then the sensation drifted away on the breeze and was gone.

Memories flashed through Kainé's mind, one after the other, faster and faster, until they became a meaningless jangle of noise. Kainé screamed then, a thunderous sound that echoed off the cliffs and seemed to roll away forever." - Kainé's Dreams: Separation, NIER (2010).

Agradecimentos

Meu principal agradecimento é para o meu orientador, Prof. José Edil, que me incentivou a realizar este trabalho e concluir o curso de graduação.

Agradeço também ao meu pai e à minha mãe por me proporcionarem todo o suporte possível para realização do trabalho e do curso.

Agradeço às fontes de conhecimento citadas no trabalho.

Agradeço aos meus amigos que me ajudaram durante a graduação e realização do projeto, em ordem alfabética: Davi Freitas, Eduardo Carvalho, Filipe Teixeira, Henrique Brant, Hugo Pfeffer, Igor Coutinho, Luiz Fernando, Marcos Tonin, Matheus Henrique e Matheus Santana.

E finalmente, agradeço às seguintes pessoas que me inspiraram de alguma forma durante o processo de desenvolvimento do trabalho: Yoko Taro, Keiichi Okabe, Emi Evans, Hironobu Sakaguchi, Nobuo Uematsu, Hidetaka Miyazaki, Motoi Sakuraba, Akai Haato, Amatsuka Uto, Gawr Gura, Gokigen Naname, Jean Dias, Leandro Bife e Victor Schiavon.

Resumo

A popularidade da indústria de IoT proporcionou uma ascensão no número de tecnologias disponíveis para implementações de automação residencial. Este trabalho busca tirar proveito dessas ferramentas para a prototipação de um controle remoto universal infravermelho com suporte a comandos de voz. O assistente virtual utilizado para recepção e tratamento dos comandos de voz foi o *Google Assistant*, enquanto a placa de desenvolvimento *NodeMCU* foi adotada para construção do controle universal. Para implementar a comunicação entre o assistente de voz e o protótipo, um servidor utilizando o sistema operacional de automação residencial *Home Assistant* foi estabelecido. Com o protótipo finalizado, se fez possível mensurar a temperatura ambiente, assim como controlar dois aparelhos de televisão de diferentes fabricantes e um aparelho de TV a cabo. Os testes realizados evidenciam o potencial de centralização das funções de controles remotos infravermelhos de quaisquer dispositivo no protótipo construído, adicionando suporte a realização dessas funções via comandos de voz.

Palavras-chave: controle remoto, infravermelho, comandos de voz, assistente virtual, automação residencial, internet das coisas

Abstract

The rising popularity of the IoT industry has provided the increase of available technologies for home automation implementations. This project seeks to take advantage of these tools to build a prototype of a universal infrared remote control with voice command support. The virtual assistant used to receive and handle the voice commands was *Google Assistant*, while the *NodeMCU* development board was adopted to build the universal control. To enable the communication between the voice assistant and the prototype, a server using the *Home Assistant* home automation operating system was established. With the prototype built, it was possible to measure the room temperature, as well as to control two TVs from different manufacturers and a cable TV box. The tests performed show the potential of centralizing the functions performed by infrared remote controls of any device in the built prototype, adding support for performing these functions via voice commands.

Keywords: remote control, infrared, voice commands, virtual assistant, home automation, internet of things

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

ADC *Analog-to-Digital Converter.*

API *Application Programming Interface.*

CGNAT *Carrier-grade Network Address Translation.*

CHUNK *Chunking.*

COP *Common Operational Picture.*

CPU *Central Processing Unit.*

DC *Direct current.*

DHCP *Dynamic Host Configuration Protocol.*

DIY *Do It Yourself.*

DNS *Domain Name System.*

EPoSS *European Technology Platform on Smart Systems Integration.*

ETP *European Technology Platforms.*

GB Gigabytes.

GPIO *General Purpose Input/Output Interface.*

HTTP *Hypertext Transfer Protocol.*

HTTPS *Hyper Text Transfer Protocol Secure.*

HVAC *Heating, Ventilating and Air Conditioning.*

IDE *Integrated Development Environment.*

IoT *Internet of Things.*

IP *Internet Protocol.*

IR *Infrared.*

ISP *Internet Service Provider.*

JSON *JavaScript Object Notation.*

LED *Light Emitting Diode.*

LWIR *Long Wave Infrared.*

M2M *Machine to Machine.*

MB Megabytes.

MCU *Microcontroller Unit.*

MQTT *MQ Telemetry Transport.*

NER *Named Entity Recognition.*

NFC *Near Field Communications.*

NIR *Near Infrared.*

NLP *Natural Language Processing.*

OS *Operational System.*

OTA *Over The Air.*

PCB *Printed Circuit Board.*

PEM *Privacy-Enhanced Mail.*

POS *Part-Of-Speech Tagging.*

PWM *Pulse Width Modulation.*

QoS *Quality of Service.*

RAM *Random Access Memory.*

REST *Representational State Transfer.*

RF *Radio frequency.*

RFID *Radio Frequency Identification.*

ROM *Read-Only Memory.*

SMB/CIFS *Server Message Block/Common Internet File System.*

SMTP *Simple Mail Transfer Protocol.*

SOAP *Simple Object Access Protocol.*

SoC *System-On-a-Chip.*

SPI *Serial Peripheral Interface.*

SRAM *Static Random Access Memory.*

SRL *Semantic Role Labeling.*

TCP *Transmission Control Protocol.*

TLS *Transport Layer Security.*

UART *Universal Asynchrounous Receiver/Transmiter.*

URL *Uniform Resource Locator.*

USB *Universal Serial Bus.*

VM *Virtual Machine.*

Wi-Fi *Wireless Fidelity.*

XML *Extensible Markup Language.*

YAML *YAML Ain't Markup Language.*

Capítulo 1

Introdução

Na contemporaneidade, quartos e salas estão equipados com vários dispositivos eletrônicos que possuem seus próprios controles remotos, como por exemplo ar condicionado, televisão, reprodutor de *Blu-Ray*, *home theater*, entre outros. Cada dispositivo possui seu próprio controle, isso torna dispendioso e até complicada a realização de tarefas simples, sempre existem questões como "Onde está o controle do ar condicionado?", "A pilha de um dos controles acabou", "Como eu faço para mudar para a exibição da TV para o *home theater*?". Aparelhos de televisão mais modernos possuem controles mais avançados que podem controlar vários dispositivos, mas mesmo neste caso é necessário um nível de conhecimento tecnológico para gerenciar os aparelhos eletrônicos por meio deste controle.

Para pessoas debilitadas, como por exemplo em situação de internação hospitalar, se torna ainda mais incômoda a gerência dos dispositivos por meio de diversos controles remotos. Em casos de função motora comprometida, essa gerência por meio de controle remoto em si chega a se tornar impraticável.

A implementação de automação residencial vem se tornando cada vez mais comum, podendo ser definido como uma residência ou ambiente que possui aparelhos que se comunicam entre si e que podem ser operados remotamente por meio de um sistema de controle [1]. Ao aplicar esse conceito na unificação do controle de dispositivos em um ambiente utilizando comandos por voz, verifica-se uma simplificação na gerência destes aparelhos.

1.1 Motivação e Justificativa

A centralização e simplificação do controle de dispositivos eletrônicos que utilizam controle remoto com tecnologia infravermelha, se enquadra no conceito de casa inteligente, mas também possui uma importância que vai além da praticidade, podendo ajudar ido-

sos, pessoas com dificuldade de locomoção, pessoas com deficiência visual, pacientes em hospitais, entre outros casos.

1.2 Objetivos

O objetivo principal do trabalho é projetar um controle remoto universal que utiliza comandos de voz para gerenciar aparelhos eletrônicos que possuem controle remoto de tecnologia infravermelha. Os objetivos específicos são:

1. configuração das ferramentas disponibilizadas pelo *Google Assistant*, empregando o *Google Home Mini* e um aparelho celular como receptores dos comandos de voz do controle universal;
2. implementação de um servidor para o estabelecimento da comunicação entre o *Google Assistant* e o controle universal utilizando o protocolo MQTT ou semelhante para troca de mensagens entre o servidor e o controle;
3. construção de um protótipo de controle universal por comando de voz a partir do *NodeMCU* com capacidade de receber e transmitir sinais infravermelhos, explorando os diferentes protocolos de comunicação infravermelha das diversas marcas de aparelhos eletrônicos que possuem controles remotos com essa tecnologia.

1.3 Estrutura do documento

O Capítulo 2 apresenta os conceitos básicos que fizeram parte do desenvolvimento do trabalho como *Internet of Things* (IoT), *Smart Home*, *Natural Language Processing* (NLP), *Microcontroller Unit* (MCU), Protocolo MQTT, *Application Programming Interface* (API) e comunicação por infravermelho. O Capítulo 3 conceitualiza o projeto em si, detalhando as etapas de desenvolvimento do projeto, assim como as técnicas utilizadas para a prototipação do controle remoto universal controlado por voz. O Capítulo 4 apresenta os testes realizados e resultados obtidos com o protótipo desenvolvido. O Capítulo 5 se trata da conclusão do trabalho e possíveis trabalhos futuros.

Capítulo 2

Revisão Teórica

Este capítulo apresenta os principais conceitos e ferramentas utilizados no desenvolvimento do trabalho com o objetivo de detalhar o projeto e evidenciar o planejamento da sua implementação.

2.1 *Internet of Things* (IoT)

IoT é um termo muito utilizado, sendo um um assunto bastante debatido dentro e fora da comunidade acadêmica. Entender o verdadeiro significado de IoT pode ser complicado, assim como conceber uma definição para o termo, se faz necessário estudar a etimologia da expressão para esclarecer o sentido de IoT e realizar uma conceitualização própria.

Segundo Atzori *et al.* [2], o termo internet das coisas é composto por duas palavras: "Internet", que idealiza uma visão orientada a redes; e "coisas", que foca em generalizar os objetos que podem ser integrados em uma rede. Além de analisar sintaticamente, os autores também realizaram uma análise semântica do termo, utilizando a definição de internet das coisas dada pelo *Working group RFID of the ETP EPoSS* [3]: "Uma rede mundial de objetos interconectados endereçados unicamente, baseada em protocolos de comunicação padronizados", para dividir o paradigma de internet das coisas em três visões:

- **Visão orientada a coisas:** sensores, *Radio Frequency Identification* (RFID), *Near Field Communications* (NFC), objetos do cotiano, dispositivos inteligentes;
- **Visão orientada a redes:** *middleware*, conectividade e comunicação;
- **Visão orientada a semântica**: tecnologia semântica, análise e entendimento dos dados, *data science*.

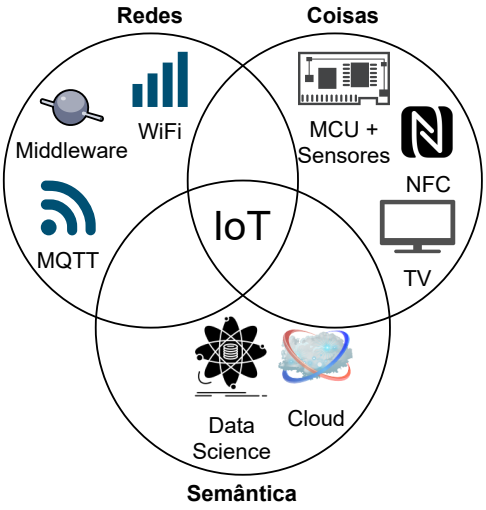

Figura 2.1: As três visões (redes, coisas e semântica) do paradigma de internet das coisas.

Essas diferentes visões, ilustradas na Figura 2.1, evidenciam a interdisciplinaridade na acepção de IoT, dessa forma, se faz importante verificar a convergência dos três paradigmas para notar a utilidade de uma aplicação IoT [4]. Um exemplo de aplicação que possui essa interseção entre as três visões: o próprio projeto de controle remoto universal controlado por voz, onde a parte orientada a redes é a implementação da comunicação entre os dispositivos e o controle, a parte orientada a coisas se relaciona aos objetos que podem ser controlados remotamente e a parte orientada a semântica, a implementação da interpretação dos comandos de voz realizados pelo usuário.

A definição de internet das coisas mais utilizada na comunidade acadêmica foi idealizada por Gubbi *et al.* [4]: "Interconexão entre dispositivos sensores e atuantes que provém a habilidade de compartilhar informação entre plataformas por meio de um *framework* unificado, desenvolvendo uma *Common Operational Picture* (COP) para permitir aplicações inovadoras. Essa interconexão pode ser atingida por meio de sensoriamento fluido em larga escala, análise de dados e representação de informações utilizando sensores onipresentes de ponta e computação em nuvem.". A caracterização de IoT feita por Gubbi *et al.* remove a limitação da utilização de um protocolo de comunicação padrão, presente na definição dada pelo *Working group RFID of the ETP EPoSS* e utilizada por Atzori *et al.*, permitindo a utilização de protocolos estado-da-arte para aplicações duradouras [4].

A partir das definições apresentadas, é possível elucidar uma conceituação intuitiva para internet das coisas: conjunto de dispositivos com identificação única interconectados por meio de uma rede com o objetivo de compartilhar informações e realizar ações com base nessas informações.

2.2 *Smart Home*

Existem várias conceituações para o significado de casa inteligente (*smart home*, em inglês), uma definição intuitiva seria: uma casa ou um ambiente que possui aparelhos que automatizam ou facilitam tarefas antes realizadas por pessoas, como por exemplo, automatização do funcionamento das luzes de acordo com a presença em um ambiente, facilitando a tarefa de ligar ou desligar uma lâmpada por meio de um interruptor. A automatização em uma casa inteligente pode ser benéfica em outros aspectos além da própria facilitação de uma tarefa. Ainda no exemplo anterior, situações como luzes ligadas desnecessariamente podem ser evitadas, economizando energia e diminuindo a conta de luz, beneficiando o usuário e também o meio-ambiente. Alguns exemplos de implementação de serviços em uma *smart home* são ilustrados na Figura 2.2.

Figura 2.2: Exemplos de implementação de serviços em uma *smart home*.

Uma das primeiras definições de casa inteligente foi dada por Lutolf [5] em 1992. De acordo com Lutolf, "O conceito de casa inteligente é a integração de diferentes serviços em uma casa por meio de um sistema de comunicação em comum, assegurando uma operação econômica, segura e confortável da residência, incluindo um alto grau de inteligência funcional e flexibilidade". A definição de Lutolf se preocupa em esclarecer a forma de operação do sistema, considerando a economia, conforto e segurança que devem ser proporcionados pelos diferentes serviços integrados. Também há uma generalização na utilização de serviços ao invés de dispositivos ou aparelhos como se é pensado atualmente devido à associação da ideia de IoT com a implementação de casas inteligentes.

Alam *et al.* [6] abstraem o entendimento de casa inteligente considerando tendências mais recentes nas pesquisas sobre *smart home*: "Uma casa inteligente é a aplicação de computação onipresente capaz de prover serviços automatizados ou assistivos com reconhecimento de contexto do usuário em forma de inteligência ambiental, controle residencial remoto ou automação residencial." . A abstração de casa inteligente nessa definição pode ser considerada uma evolução natural da conceituação de Lutolf, devido a utilização dos conceitos de automação e inteligência residencial, acesso remoto e inteligência ambiental. Além de diversas definições, *smart home* também possui múltiplos sinônimos: domótica (*domotique*, em inglês); automação residencial (*home automation*, em inglês); casa adaptativa (*adaptive house*, em inglês); e casa consciente (*aware house*, em inglês) [6].

Diversos tipos de usuários podem se beneficiar das tecnologias de casa inteligente, alguns exemplos de usuários são: usuários que desejam mais praticidade na realização de ações repetitivas que podem ser automatizadas; pessoas em situação de internação hospitalar, onde a locomoção é limitada e existem vários sistemas que podem ser controlados remotamente; idosos ou pessoas com deficiência que possuem impedimentos cognitivos, como Alzheimer, e/ou impedimentos físicos, como deficiência visual, deficiência auditiva, dificuldade de locomoção e distúrbio da fala; cuidadores de idosos e pessoas com deficiência, como família, amigos, vizinhos e cuidadores contratados; pessoas que vivem sozinhas e são incapazes de buscar ajuda em situações de emergência, como derrame, perda de consciência, quedas e infarto [7].

Winkler [8] exemplifica o conceito de casa inteligente para facilitar a vida de idosos, essa aplicação exige um sistema com um computador central que controla variados sistemas na casa inteligente para prover uma maior independência a idosos, entre esses sistemas são citados controle de temperatura e de iluminação. No caso de idosos e pessoas debilitadas, a ideia da casa inteligente se torna muito mais crítica, pois podem acontecer situações de vida ou morte que podem ser evitadas pelas decisões tomadas na implementação dessa automação residencial.

2.3 *Natural Language Processing* (NLP)

O processamento de linguagem natural (*Natural Language Processing*, em inglês) pode ser entendido com uma tecnologia que tem como objetivo automatizar a interpretação, assim como a geração de texto e fala em linguagens humanas para proporcionar uma interação mais natural entre pessoas e máquinas. No contexto de IoT, NLP se relaciona com a parte de análise de dados (língua falada ou escrita), sendo o principal componente da interface responsável pelo gerenciamento da interação entre usuários e aplicação.

Cambria *et al.* [9] definem NLP como: "variedade de técnicas computacionais motivadas teoricamente utilizadas para análise automática e representação de linguagem humana". Com o advento da Internet e a evolução da mesma com a criação de redes sociais (como *Facebook*, *LinkedIn*, *Instagram*, etc.), o compartilhamento de conteúdo nessas plataformas produz uma quantidade exorbitante de informações que não podem ser processadas diretamente por computadores devido à sua natureza não-estruturada. Existem diversos algoritmos que são eficientes no processamento sintático de textos (obtenção, sumarização, tradução, particionamento, análise gramatical e contagem de palavras), porém são limitados no processamento semântico (interpretação de sentenças, extração de informação relevante), isso se deve ao fato de que a comunidade pesquisadora em NLP adotou como estratégia focar no estudo de processamento sintático ao invés do semântico, devido a menor complexidade e aplicabilidade de técnicas de aprendizado de máquina [9].

Para compreender algumas das tarefas de processamento de linguagem natural, assim como a importância da relação entre NLP e aprendizado de máquina, temos como exemplo a utilização de um *framework* de *deep learning* feita por Collobert *et al.* [10] para realização de tarefas de NLP. A implementação dos autores supera a maioria das abordagens estado-da-arte de cada tarefa, as tarefas de *benchmark* realizadas nessa implementação são:

- ***Chunking* (CHUNK)**: também chamado de *shallow parsing* (análise superficial, em português), rotula segmentos de uma sentença a partir dos constituintes sintáticos da mesma, como por exemplo, frase de pronome (*noun phrase*, em inglês), frase de verbo (*verb phrase*, em inglês);
- ***Named Entity Recognition* (NER)**: rotulação de elementos atômicos da sentença em categorias, como "PESSOA", "TEMPO" ou "LOCALIZAÇÃO";
- ***Part-Of-Speech Tagging* (POS)**: rotulação de cada palavra com uma etiqueta única que indica a classificação sintática da palavra, como por exemplo, substantivo, advérbio, plural e etc.;
- ***Semantic Role Labeling* (SRL)**: provê um papel semântico para um componente sintático de uma sentença. É composto por várias etapas: Produção de uma árvore de análise sintática (*parse tree*, em inglês); Identificação dos nós que representam argumentos de um verbo; Classificação dos nós para computar a etiqueta SRL correspondente.

Assistentes virtuais que possuem suporte a comandos de voz, como o *Google Assistant*[11], *Microsoft Cortana*[12] e *Amazon Alexa*[13] são alguns dos programas que usufruem de implementações de NLP para facilitar a interação entre os usuários e seus respectivos produtos. Ferramentas de pesquisa como *Google* e *Bing* também utilizam *Natural*

Language Processing para realizar a filtragem de informações relacionadas com os termos presentes na busca realizada pelos usuários. No contexto de casa inteligente, pode-se notar cada vez mais a presença de assistentes de voz em diversos produtos presentes em residências, principalmente em televisores, evidenciando a importância de NLP para implementações de *smart home*.

2.4 *Microcontroller Unit* (MCU)

Um microcontrolador (*Microcontroller Unit*, em inglês) é um processador construído em um único circuito integrado, sendo muito utilizado na implementação de sistemas embarcados devido ao seu baixo custo comercial, baixo consumo de energia, tamanho reduzido, velocidade e entre outros aspectos vantajosos para aplicações de propósito específico. Se comparado ao microprocessador utilizado em computadores pessoais, o microcontrolador tem como diferencial justamente a sua arquitetura integrada, tomando como exemplo a memória *Read-Only Memory* (ROM), o microcontrolador já possui memória ROM em seu circuito integrado, assim como outros componentes essenciais para o funcionamento de um computador, enquanto o microprocessador é somente uma *Central Processing Unit* (CPU), necessitando de memória *Random Access Memory* (RAM) e componentes conectados de forma externa para realizar a função de um computador. Sendo assim, o microcontrolador é ideal para aplicações embarcadas, enquanto o microprocessador é ideal para aplicações de propósito geral, por esse motivo o microprocessador é amplamente utilizado em computadores pessoais. A Figura 2.3 compara visualmente as arquiteturas do microprocessador e do microcontrolador.

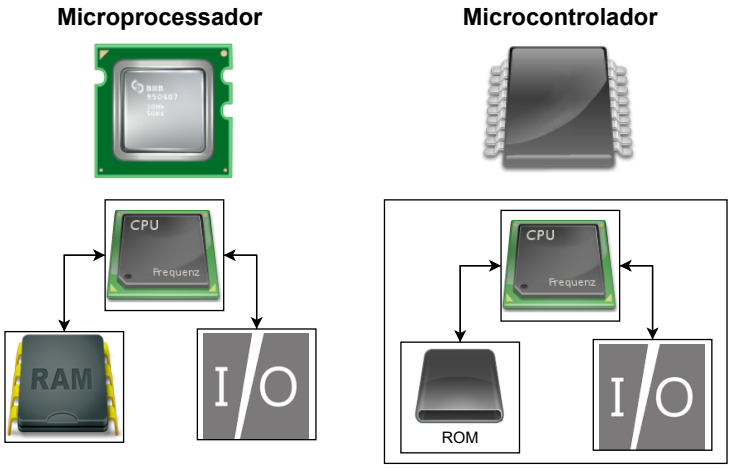

Figura 2.3: Comparação entre a arquitetura do microprocessador (vários *chips*) e do microcontrolador (único *chip* integrado).

Ibrahim [14] evidencia os componentes mínimos de um microcontrolador, sendo eles a CPU, memória RAM e um módulo de entrada e saída. O autor considera como microcontrolador desde um controlador embarcado em um único *chip* até um computador com teclado, monitor, impressora e outros componentes, evidenciando a diferença entre o microprocessador e o microcontrolador por meio da necessidade de *chips* de suporte externos por parte do microprocessador para ser utilizado como um controlador digital da mesma maneira que um microcontrolador.

No contexto de IoT, os microcontroladores são vastamente utilizados como o cérebro das "coisas" da internet das coisas, devido ao baixo custo, capacidade de obtenção de dados por meio de sensores e atuação. Tomando como exemplo o projeto de controle universal, o microcontrolador recebe um comando e realiza uma ação específica, a ação neste caso é de enviar sinais infravermelhos para o gerenciamento de dispositivos eletrônicos. O microcontrolador pode funcionar por si só baseando-se no contexto apresentado por sensores, como por exemplo um sensor de luminosidade pode definir a ação de acender ou apagar as luzes de um ambiente. A partir dos exemplos pode-se evidenciar a importância dos microcontroladores em projetos de IoT.

Existem diversas placas de desenvolvimento que utilizam microcontroladores com suporte à conexão Wi-Fi para a implementação de projetos IoT, como por exemplo a *WEMOS LOLIN D1 mini*[1] que utiliza o MCU *ESP8266*, e a *ESP32-S2-DevKitM-1*[2] que utiliza o MCU *ESP32-S2-MINI-1*. A placa de desenvolvimento adotada no projeto foi a *NodeMCU V2* que possui o MCU *ESP8266* com o *firmware NodeMCU* instalado.

2.5 Protocolo MQTT

O *MQ Telemetry Transport* (MQTT) [15] é um protocolo de transporte de mensagens que utiliza o padrão de troca de mensagens publicar-assinar (*publish-subscribe pattern*, em inglês) no modelo cliente-servidor sobre a pilha TCP/IP, tendo como objetivo ser um protocolo leve, simples, aberto e de fácil implementação, tornando-o ideal para aplicações onde o uso de banda de rede e consumo de energia devem ser minimizados, como por exemplo aplicações no contexto IoT e comunicação *Machine to Machine* (M2M).

A parte "MQ" do nome do protocolo, criado em 1999 e utilizado internamente pela IBM até seu lançamento de forma gratuita como MQTT 3.1 em 2010, se refere ao produto *MQ Series*, desenvolvido pela IBM para suportar o protocolo em si, e não à *message queue/queuing* como se é apresentado em diversas fontes. O protocolo MQTT não é uma solução de *message queuing* [16].

[1] `https://www.wemos.cc/en/latest/d1/d1_mini.html`

[2] `https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/hw-reference/esp32s2/user-guide-devkitm-1-v1.html`

O padrão publicar-assinar (também conhecido como *pub-sub*, em inglês) é uma alternativa à arquitetura cliente-servidor tradicional. No modelo cliente-servidor, a comunicação entre um cliente que produz uma mensagem e o recipiente alvo dessa mensagem é realizada de forma direta. Já no modelo publicar-assinar, essa comunicação é desacoplada por meio da definição de três tipos de componentes em sua arquitetura [16]:

- ***Publisher***: cliente que envia uma mensagem ao *broker*, não se comunica diretamente com o *subscriber*, pois nem considera a existência do mesmo;
- ***Subscriber***: cliente que recebe uma mensagem do *broker*, não se comunicando de forma direta com o *publisher*, pois também não considera a existência desse tipo de cliente;
- ***Broker***: servidor que conecta as duas categorias de clientes, sendo o responsável pela filtragem e distribuição das mensagens enviadas pelos *publishers* para seus respectivos *subscribers*.

O desacoplamento provido nesse padrão de comunicação possui três dimensões [17]: espacial, pois os dois tipos de clientes não precisam se conhecer; temporal, os clientes não precisam executar ao mesmo tempo, como por exemplo, um *publisher* envia uma mensagem que os *subscribers* podem receber posteriormente e em tempos diferentes entre si; de sincronização, um *publisher* pode enviar uma mensagem tanto de forma síncrona quanto de forma assíncrona, ou seja, interrompendo ou não interrompendo suas operações até que o *subscriber* receba a mensagem.

Com a utilização de um *broker*, a melhoria da escalabilidade no padrão *pub-sub* é outro fator relevante na comparação com o modelo cliente-servidor, pois operações podem ser paralelizadas e mensagens podem ser processadas de forma orientada a eventos. Para realizar a distribuição de informações relevantes a cada *subscriber*, o *broker* possui os seguintes tipos de filtragem: baseada na parte de tópico de cada mensagem; baseada no conteúdo da mensagem em si; baseada no tipo ou classe da mensagem [16].

A Figura 2.4 tem como objetivo ilustrar o que seria a aplicação do padrão publicar-assinar no projeto em si. O *publisher* nesse caso seria uma placa de desenvolvimento *NodeMCU* com um sensor de temperatura (como por exemplo um *LM35*) que publica o valor da temperatura no ambiente para um *broker* em um determinado intervalo de tempo. O *broker* ilustrado implementa o protocolo MQTT e se encontra presente no servidor de automação. Os *subscribers*, que recebem as informações de temperatura por meio do *broker*, seriam o auto-falante inteligente *Google Home Mini* e um aparelho *smartphone* com o aplicativo do *Google Assistant*. É importante ressaltar que na implementação do projeto, só existe um *subscriber* que é uma entidade na interface do próprio servidor de

automação, mas para fins ilustrativos, o mesmo foi divido de acordo com o dispositivo sendo utilizado para interação com o assistente de voz.

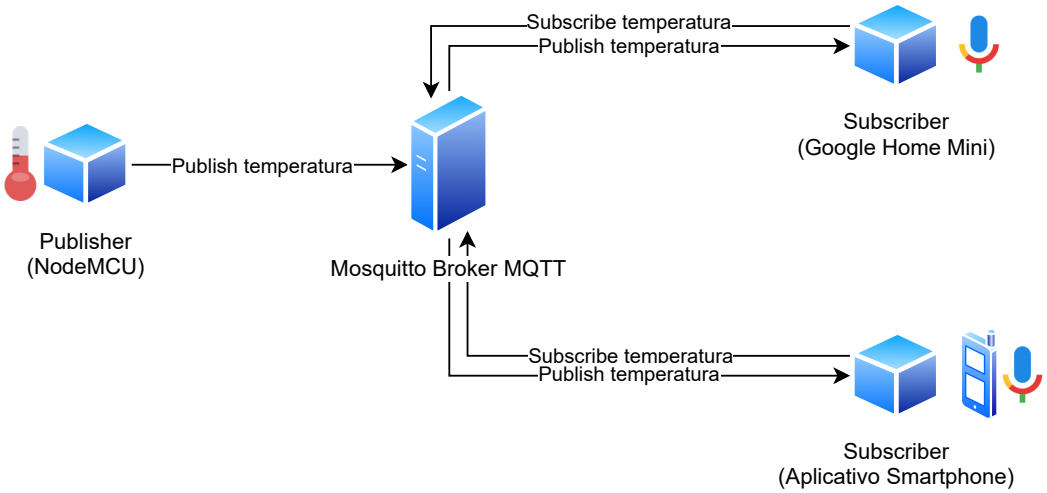

Figura 2.4: Diagrama exemplificando o padrão publicar-assinar em um contexto ilustrativo relacionado ao trabalho.

Especificando as características relacionadas ao padrão publicar-assinar implementadas no protocolo MQTT [16]: desacoplamento espacial, para enviar e receber mensagens os clientes necessitam somente do endereço IP e porta de execução do *broker*; desacoplamento temporal, apesar de não ser a intenção na maioria dos casos de uso do protocolo, o *broker* pode enviar mensagens posteriormente para clientes que não se encontram disponíveis; desacoplamento de sincronização, na maioria dos casos de uso, os clientes não são bloqueados no processo de recebimento ou envio de mensagens; filtragem de mensagens baseada em tópicos, toda mensagem possui um campo de tópico que é utilizado pelo *broker* para distribuir mensagens somente aos clientes inscritos no tópico em questão.

O processo de conexão entre um cliente e um *broker* no protocolo MQTT é estabelecido em duas etapas [16]:

1. o cliente envia uma mensagem *CONNECT* ao *broker*, essa mensagem pode possuir diversos argumentos (*username*, *password*, *keepAlive*, entre outros), de forma obrigatória a mesma deve possuir um identificador único do cliente (*clientId*) e um atributo que define se a sessão a ser estabelecida será persistente (*clientSession*);

2. o *broker* responde ao cliente com uma mensagem *CONNACK*, que possui um indicador de persistência da sessão (*sessionPresent*) e um código de retorno que indica se a conexão foi aceita (*returnCode*).

No caso em que a mensagem de conexão não estiver de acordo com a especificação ou se muito tempo passou entre a abertura da *socket* e o envio da mensagem, o *broker* fecha a conexão, evitando a sobrecarga do mesmo devido a clientes maliciosos [16].

A qualidade de serviço ou *Quality of Service* (QoS) [16] é definida como "um acordo entre o remetente e o destinatário de uma mensagem que define a garantia de entrega desta mensagem". O *publisher* que publica a mensagem ao *broker* define o nível de QoS da mensagem. O *broker* transmite a mensagem para os *subscribers* de acordo com o nível de QoS definido por cada *subscriber* no processo de *subscribe*, se o nível definido por um dos clientes *subscribers* for menor que o nível definido pelo cliente *publisher*, o *broker* transmite a mensagem com o nível de QoS de menor valor. O protocolo MQTT possui três níveis de qualidade de serviço:

- **QoS 0 - *At most once***: não há garantia de entrega. O destinatário não confirma o recebimento da mensagem (*PUBLISH*) e a mensagem não é armazenada e retransmitida pelo remetente, provendo a mesma garantia de entrega que o protocolo TCP.
- **QoS 1 - *At least once***: garante que a mensagem (*PUBLISH*) foi entregue pelo menos uma vez. O remetente armazena a mensagem até receber um pacote de confirmação de recebimento da mensagem (*PUBACK*) por parte do destinatário. É possível que a mensagem seja enviada ou recebida múltiplas vezes.
- **QoS 2 - *Exactly once***: garante que a mensagem (*PUBLISH*) foi recebida apenas uma única vez pelos destinatários. O destinatário confirma o recebimento da mensagem enviando um pacote (*PUBREC*) ao remetente. Ao receber esse pacote, o remetente descarta a mensagem inicial (*PUBLISH*) e responde ao destinatário com um pacote (*PUBREL*). Com o recebimento desse pacote, o destinatário então descarta informações relacionadas à mensagem inicial e envia o pacote final (*PUBCOMP*) ao remetente. Com o término do processo tanto o remetente quanto o destinatário garantem que a mensagem foi entregue e que o remetente está ciente da entrega da mensagem.

Com a implementação da qualidade de serviço, o protocolo permite a utilização de um nível de garantia de entrega de acordo com a lógica da aplicação e da confiabilidade da rede, facilitando por exemplo a comunicação por meio de redes instáveis [16].

Se comparado ao *Hypertext Transfer Protocol* (HTTP), que também pode ser utilizado para troca de mensagens no padrão publicar-assinar, o MQTT além de ser mais leve, também é mais eficaz em condições adversas de conectividade intermitente e instabilidade na rede. Um cenário onde diversos dispositivos devem se comunicar utilizando o modelo publicar-assinar com baixa latência e menor consumo possível de largura de banda é ideal

para a utilização do MQTT. Alguns exemplos de domínios de aplicação que correspondem ao cenário descrito são: monitoramento de ambientes e tráfego; automação residencial; RFID; telemática automotiva; medicina; entre outros [17].

Os principais desafios que o protocolo busca solucionar podem ser sumarizados em: possibilitar a transmissão de um alto volume de dados sem grandes quantidades de *overhead*; facilitar a transmissão de dados de um cliente para vários clientes; suporte à arquitetura orientada a eventos com entrega de mensagens de forma bidirecional e assíncrona; funcionamento em redes instáveis com entrega confiável e com baixíssima latência, mais próxima possível de entrega em tempo real; oferecer segurança e privacidade para os dados transmitidos [17].

2.6 *Application Programming Interface* (API)

Uma API (interface de programação de aplicações, em português) [18] "é um conjunto de definições e protocolos usado no desenvolvimento e na integração de software de aplicações". A utilização de uma API permite a comunicação entre diferentes serviços abstraindo a implementação dos mesmos por meio do estabelecimento de um contrato entre eles. Esse contrato determina como será a estrutura das requisições e das respostas que serão trocadas durante o processo de comunicação entre as aplicações.

A maioria das APIs são remotas, ou seja, os dados a serem obtidos ou manipulados em uma requisição não se encontram na máquina que enviou a requisição, e utilizam o padrão *web* devido a necessidade de comunicação via *internet*. Geralmente, as mensagens de requisição utilizam o procolo HTTP e a estrutura utilizada para as mensagens de resposta são arquivos de fácil manipulação como *JavaScript Object Notation* (JSON) e *Extensible Markup Language* (XML). Existem três tipos de políticas de lançamento de API: privada, onde a API é utilizada somente para uso interno, possibilitando maior controle sobre a mesma; parceira, quando uma API é compartilhada com alguns parceiros de negócio; pública, que permite a utilização da API ao público geral [18].

Para padronizar a troca de informações por meio de APIs, duas especificações foram desenvolvidos [18]:

- ***Simple Object Access Protocol* (SOAP)**: protocolo que define o XML como formato padrão das mensagens e onde as requisições realizadas por meio do HTTP ou *Simple Mail Transfer Protocol* (SMTP);
- ***Representational State Transfer* (REST)**: não é um protocolo e sim um estilo de arquitetura, pois não possui uma especificação oficial. Apesar das restrições de arquitetura definidas por Fielding [19], é a especificação mais adotada que o protocolo SOAP.

Fielding [19] define seis restrições de arquitetura para uma API ser considerada REST: arquitetura cliente-servidor e requisições realizadas por meio do protocolo HTTP; sem monitoramento de estado, nenhum dado do cliente é armazenado no servido durante a comunicação, sendo assim o estado da sessão é armazenada apenas no cliente; *cache*, para aumentar a eficiência da comunicação; sistema em camadas, as funcionalidades que podem ser implementadas em diferentes camadas são balanceamento de carga, segurança ou compartilhamento de *cache*; código sob demanda, uma restrição opcional, que define a capacidade do servidor transferir código executável ao cliente; interface uniforme, principal restrição que define uma API RESTful.

A restrição de interface uniforme possui quatro facetas [19]: identificação de recursos nas requisições, recursos são identificados e distinguidos das representações retornadas ao cliente; manipulação de recursos por meio de representações, os clientes recebem representações de recursos que devem permitir a modificação e remoção dos mesmos; mensagens auto-descritivas, as mensagens retornadas aos clientes devem possuir informação suficiente para descrever como realizar o processamento das mesmas; hipermídia como mecanismo do estado da aplicação, ao acessar um recurso, os clientes devem possuir a capacidade de descobrir todas as ações disponíveis por meio de hiperligações (*hyperlinks*, em inglês).

Um *webhook* [20] é uma ferramenta com funcionamento semelhante ao das APIs. Na maioria das implementações de API, uma resposta só é enviada quando há uma requisição. Este não é o caso para *webhooks*, pois a resposta é enviada quando a informação requerida se encontra disponível, sem a realização de uma requisição explícita. Para utilizar um *webhook*, o registro de uma *Uniform Resource Locator* (URL) onde a informação é disponibilizada se faz necessário.

2.7 Comunicação por infravermelho

O espectro de luz infravermelho, ilustrado na Figura 2.5, é definido por todas as ondas eletromagnéticas que possuem comprimento de onda de 700 nm até 1 mm, ou seja, o espectro se inicia com ondas de comprimento maiores que o da luz visível e termina com ondas de comprimento menores que os das micro-ondas. Devido ao grande intervalo definido para o espectro *Infrared* (IR), o mesmo é divido em duas regiões: *Near Infrared* (NIR) ou infravermelho próximo em português; e *Long Wave Infrared* (LWIR) ou infravermelho de onda longa em português [21].

A região NIR começa com ondas de comprimento de 700 nm e se estende até 1.400 nm. Essa região é a relevante para a eletrônica, sendo utilizada por exemplo para a operação de controles remotos e sensores de detecção de proximidade. A maioria dos emissores infravermelhos operam na faixa de 850 nm a 950 nm, e apesar de emitirem

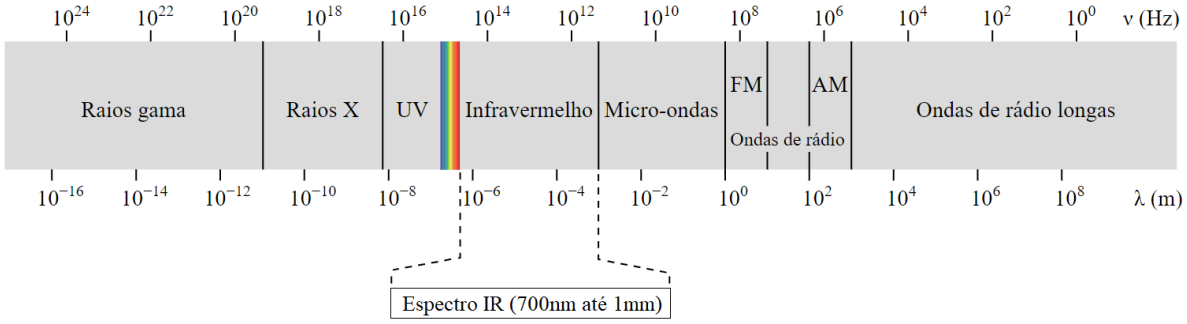

Figura 2.5: O espectro eletromagnético, com destaque ao espectro infravermelho (Fonte: [22]).

sinais invisíveis a olho nu, a maioria das câmeras digitais conseguem captar esses sinais mesmo com a utilização de filtros físicos para bloqueio dos mesmos. Devido à grande quantidade de radiação infravermelha nessa região, existe o risco de interferência nos sinais e sensores IR. Para contornar o problema, a maioria dos sistemas de emissão de sinais infravermelhos adota uma modulação de frequência fixada ao invés de filtrar luzes de comprimento de onda indesejado. Já a outra região, LWIR, se inicia com comprimento de onda 8.000 nm e termina com 15.000 nm, sendo utilizada para termografia [21].

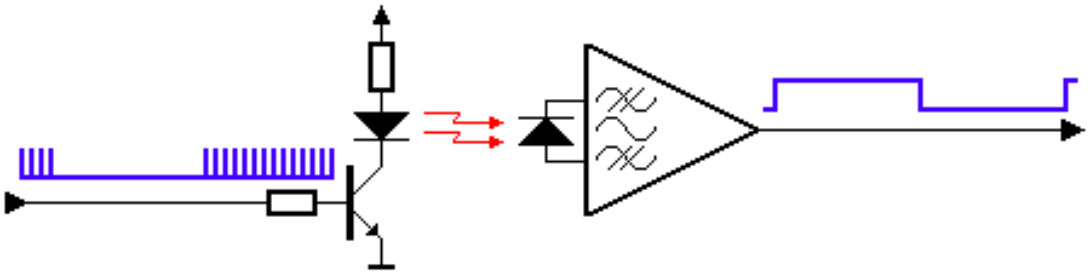

Figura 2.6: Visualização conceitual da comunicação entre um transmissor e receptor infravermelhos (Fonte: [23]).

A comunicação por infravermelho, realizada na região NIR, funciona por meio da modulação de sinais IR. Por exemplo, ao acionar um botão no controle remoto de um aparelho de televisão, o mesmo envia sinais modulados em uma frequência incomum com relação à fontes de radiação infravermelha comuns (como o Sol e lâmpadas) para o sensor IR do televisor, evitando assim a interferência de quaisquer fontes de luz e calor do ambiente. Um exemplo de valor de frequência utilizado para modulação é de 38 kHz [23].

A Figura 2.6 ilustra o sinal modulado enviado ao sensor infravermelho, que realiza a demodulação do sinal, convertendo-o em uma forma de onda binária que pode ser lida

por um MCU e decodificada em um fluxo de *bits*, que geralmente representam comandos ao sistema receptor dos sinais IR [23].

A biblioteca utilizada no projeto para suportar os diferentes protocolos de comunicação IR das diversas fabricantes que utilizam essa tecnologia foi a *IRremoteESP8266* [24]. Essa biblioteca é provida pela extensão *ESPHome* do servidor de automação *Home Assistant*. A biblioteca suporta quase uma centena de protocolos IR diferentes, enquanto a extensão *ESPHome* implementa apenas os seguintes protocolos: *LG*, *NEC*, *Panasonic*, *Pioneer*, *JVC*, *Samsung*, *Samsung36*, *Sony*, *RC5* e *Raw* (Protocolo arbitrário).

No desenvolvimento do projeto, os protocolos *Samsung*, *LG* e *NEC* foram utilizados para controlar uma TV *Samsung*, uma TV *LG* e um aparelho de TV a cabo da operadora *NET*, respectivamente.

Tabela 2.1: Códigos IR do protocolo *Samsung* com suas respectivas operações em uma TV *Samsung* compatível.

Operação	***data***
Ligar	0xE0E06798
Desligar	0xE0E06798

O protocolo *Samsung* suporta os seguintes modelos de controles remotos infravermelho e de aparelhos de televisão de acordo com a página de protocolos IR suportados pela biblioteca *IRremoteESP8266* [24]: BN59-01178B *TV remote*, DB63-03556X003 *TV remote*, DB93-16761C *remote*, IEC-R03 *remote* e UA55H6300 *TV*.

A televisão controlada no projeto possui um controle remoto de modelo BN59-01329D, que não está presente na lista de controles remotos suportados pela biblioteca. Dessa forma, é possível concluir que o protocolo IR implementado na biblioteca pode funcionar em dispositivos que não estão listados na página de protocolos IR suportados. A Tabela 2.1 ilustra os códigos IR do protocolo *Samsung* utilizados para controlar uma TV *Samsung*.

Tabela 2.2: Códigos IR do protocolo *LG* com suas respectivas operações em uma TV *LG* compatível.

Operação	***data***	***nbits***
Ligar	0x20DF10EF	32
Desligar	0x20DF10EF	32
Volume +	0x20DF40BF	32
Volume -	0x20DFC03F	32
Silenciar	0x20DF906F	32
Entrada	0x20DFD02F	32

O protocolo *LG* suporta os seguintes modelos de controles remotos infravermelho e de ar condicionado de acordo com a página de protocolos IR suportados pela biblioteca

IRremoteESP8266 [24]: 6711A20083V *remote*, AKB74395308 *remote*, AKB75215403 *remote* e S4-W12JA3AA *A/C*.

De uma forma semelhante ao que aconteceu no controle de uma TV *Samsung*, a TV *LG* controlada no trabalho possui um controle remoto de modelo AKB72914210, que não está presente na lista de controles remotos suportados pela biblioteca. A Tabela 2.2 ilustra os códigos IR do protocolo *LG* utilizados para controlar uma TV *LG*.

Tabela 2.3: Códigos IR do protocolo *NEC* com suas respectivas operações em um aparelho de TV a cabo *NET* compatível.

Operação	***address***	***command***
Ligar	0xE17A	0x48B7
Desligar	0xE17A	0x48B7
Volume +	0xE17A	0xB04F
Volume -	0xE17A	0x708F
Canal +	0xE17A	0x08F7
Canal -	0xE17A	0x58A7

Para realizar o controle de aparelhos de TV a cabo *NET*, o protocolo *NEC* foi utilizado. A Tabela 2.3 ilustra os códigos IR do protocolo *NEC* utilizados para controlar dois aparelhos de TV a cabo *NET*. Os modelos de aparelho que foram controlados no projeto são DXC5000KNB e HDC 74X1.

Os protocolos reconhecidos pela biblioteca quando o controle envia o sinal IR ao receptor infravermelho foram *JVC*, *LG*, *NEC* e *Raw*. Ao visualizar os sinais IR no *log* do MCU quando diferentes operações eram realizadas pelo controle remoto da TV a cabo, não foi possível distinguir os comandos de ligar e aumentar o volume no protocolo *JVC*, pois o mesmo utiliza apenas 16 bits. Sendo assim, não é possível utilizar o protocolo *JVC* para comunicação com os aparelhos da *NET*. Dessa forma, como o protocolo *LG* já havia sido utilizado para controlar uma TV *LG*, o protocolo *NEC* foi adotado para comunicação com os aparelhos de TV a cabo *NET*.

Capítulo 3

Projeto

Este capítulo tem como objetivo explanar, assim como detalhar o desenvolvimento da centralização da gerência de diversos aparelhos eletrônicos controlados por luz infravermelha por meio da utilização de um controle remoto universal IR de forma conjunta com um assistente de voz. Primeiramente, a visão geral da arquitetura do projeto será exposta, e posteriormente cada componente que compõe essa arquitetura será evidenciado.

3.1 Arquitetura

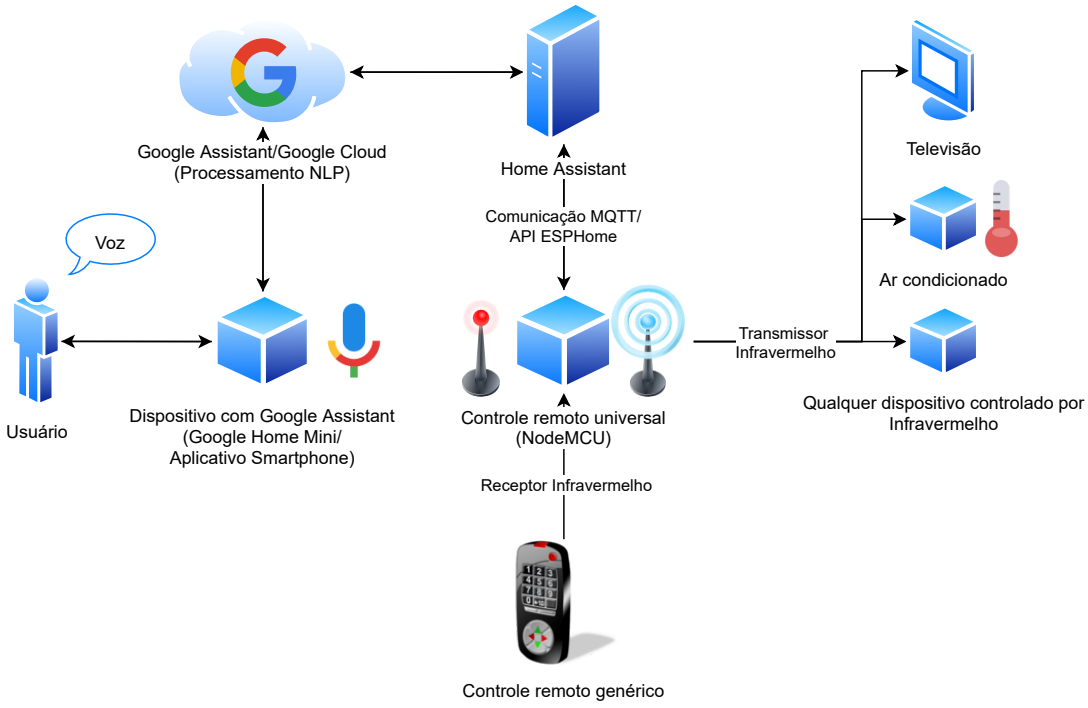

Figura 3.1: Arquitetura do projeto.

A Figura 3.1 ilustra os componentes presentes na implementação do projeto e também o fluxo de informação entre estes componentes. Nos parágrafos a seguir, serão apresentados os papéis de cada componente na arquitetura do trabalho e o motivos específicos para a escolha e utilização dos mesmos na implementação do projeto.

O primeiro integrante da arquitetura é o usuário em si, ele irá efetuar comandos de voz e também receber informações relacionadas aos comandos realizados, por exemplo, ao executar a ação de falar "Ligar televisão" para o assistente de voz, o usuário receberá uma resposta do assistente semelhante a "Ligando televisão" ou "Televisão ligada", assim como poderá visualizar o aparelho de TV ligado se o comando tiver sido bem-sucedido.

O assistente de voz é composto tanto pelo dispositivo com microfone que possui o assistente, quanto pelo processamento NLP, necessário para a interpretação dos comandos executados pelo usuário. No caso deste projeto, o assistente de voz escolhido foi o *Google Assistant*, sendo assim o componente responsável pelo processamento de linguagem natural e integração com o servidor que realiza a comunicação direta com o controle remoto projetado, é o *Google Cloud*.

Existem outras opções de assistentes de voz, como *Apple Siri*, *Microsoft Cortana* e *Amazon Alexa*, o assistente de voz do *Google* foi escolhido devido ao menor preço[1] do *Google Home Mini* em relação às outras opções de auto-falantes inteligentes e a grande quantidade de aparelhos celulares com sistema operacional *Android* no mercado, que em sua maioria já possuem o *Google Assistant* instalado de fábrica. Em 2020, 90% dos *smartphones* no Brasil possuíam o sistema operacional *Android*, de acordo com relatório do *Google* [25].

Para a implementação da comunicação entre o assistente de voz e o controle remoto universal IR, se fez necessária a utilização de um servidor intermediário, que tem as funções de integração com o *Google Assistant* e troca de mensagens com o microcontrolador utilizando o protocolo MQTT ou equivalente.

Em um protótipo inicial do projeto, o papel do servidor era realizado pelo *Adafruit IO*, para comunicação MQTT, em conjunto com o *IFTTT*, para integração com o *Google Assistant*, porém a utilização de ambos os serviços se mostrou limitada, pois o *IFTTT* não suporta a língua portuguesa para conversação com o assistente de voz, e também possivelmente dispendiosa, já que o *IFTTT* usa o modelo de negócio *freemium*, ou seja, algumas funcionalidades são pagas. As principais plataformas de automação residencial gratuitas e *open-source* que possuem as funções desejadas são *Home Assistant*, *openHAB* e *Domoticz*, a primeira opção foi escolhida arbitrariamente para a implementação do projeto.

[1] O *Google Home Mini* pode ser comprado por mais ou menos duzentos reais, enquanto seu concorrente *Amazon Echo Dot* pode ser de quarenta reais a cem reais mais caro.

O principal componente da arquitetura do projeto, o controle remoto universal IR, foi idealizado como um dispositivo capaz de se conectar à internet e de suportar o protocolo MQTT para se comunicar com o servidor, possuindo também um transmissor e receptor infravermelhos para o envio de comandos para aparelhos eletrônicos e recebimento de sinais infravermelho de controles remotos, respectivamente. O receptor infravermelho tem como intuito cadastrar novos comandos, possibilitando o controle por voz de qualquer dispositivo que possua um controle remoto com tecnologia infravermelha.

A escolha do componente mais importante do controle universal, o microcontrolador, foi baseada no suporte à comunicação por Wi-Fi, dessa forma, o MCU escolhido foi o *ESP8266*, devido ao seu baixo custo e sua capacidade de comunicação por Wi-Fi. A placa de desenvolvimento *NodeMCU* além de possuir o microcontrolador *ESP8266*, adiciona: *firmware NodeMCU*, que possibilita a programação da placa em linguagem *Lua*, sendo compatível também com a *Integrated Development Environment* (IDE) do *Arduino*; um conversor USB/Serial integrado, eliminando a necessidade de um conversor externo para comunicação com o computador; uma porta Micro-USB, que simplifica a alimentação e a programação do microcontrolador. Sendo assim, a placa de desenvolvimento foi escolhida para exercer o papel do controle remoto universal IR na implementação do projeto.

A partir da visão geral da arquitetura apresentada nesta seção, as próximas seções irão detalhar a implementação desta arquitetura na seguinte sequência: estabelecimento do servidor de automação; API nativa *ESPHome*, uma alternativa ao protocolo MQTT; comunicação do servidor com o microcontrolador utilizando o protocolo MQTT e a API nativa *ESPHome*; integração do servidor com o assistente de voz; utilização do transmissor e receptor infravermelhos na placa de desenvolvimento para controlar aparelhos eletrônicos; requisições da temperatura ambiente por meio do assistente de voz; e finalizando com o protótipo do controle remoto universal IR construído.

3.2 O servidor de automação *Home Assistant*

O *Home Assistant* é descrito em seu site oficial [26] como "automação residencial *open-source* privilegiando o controle local e a privacidade, movida por uma comunidade mundial de inventores e entusiastas *Do It Yourself* (DIY), sendo idealizada para execução em um *Raspberry Pi* ou servidor local.", além disso, existem três pilares fundamentais para esta implementação de automação residencial, são eles:

- **Automatizar:** possibilidade de configuração de diversas regras para os dispositivos inteligentes, como por exemplo, ligar as luzes da casa quando o sol se põe, desligar as luzes quando a reprodução de um filme é iniciada, entre diversos outros exemplos produzidos pela comunidade;

- **Controlar:** o controle da casa inteligente é feito por meio de uma única interface, sem a necessidade de armazenamento de dados em nuvem, mantendo a privacidade do usuário;
- **Observar:** o servidor irá acompanhar o estado dos dispositivos que fazem parte da automação residencial, utilizando por exemplo, o protocolo MQTT, integração com o *Google Assistant*, entre outros.

Existem quatro tipos de instalação para o *Home Assistant* [26]: sistema operacional otimizado para o funcionamento do *Home Assistant*, que possui o *Supervisor* para gerenciar o núcleo do sistema assim como as extensões (*OS*); instalação independente do núcleo baseada em *container* (*Container*); instalação manual do núcleo utilizando um ambiente virtual *Python* (*Core*); e instalação manual do *Supervisor* (*Supervised*). O método de instalação utilizado no projeto foi o *OS*, pois o mesmo é o mais completo e também é o recomendado pelos desenvolvedores do servidor de automação.

Para realizar a instalação do *Operational System* (OS), se faz necessária a utilização de um *Raspberry Pi* ou a criação de uma *Virtual Machine* (VM). Considerando a acessibilidade do projeto, o método escolhido foi a criação de uma VM, que pode ser feita em qualquer computador pessoal que possua como sistema operacional *Windows*, *macOS* ou *Linux*. Neste projeto, a criação da VM foi realizada no *Windows 10* utilizando a aplicação *Oracle VM VirtualBox Manager* Versão 6.1.

As configurações mínimas recomendadas para a máquina virtual [26]: 2 Gigabytes (GB) de memória RAM; 32 GB de armazenamento; e 2vCPU foram utilizadas na criação da VM. Com relação às configurações de rede, o adaptador de rede da máquina virtual foi conectada a um adaptador de rede do computador *host* em modo *bridge*, ou seja, uma nova interface de rede é criada em *software*, simulando a conexão física por cabo de rede da VM na interface de rede criada na máquina *host* [27]. Após a finalização do processo de instalação e inicialização da VM criada, o acesso ao *Home Assistant* é realizado por meio do endereço URL `http://homeassistant.local:8123` em um navegador na máquina *host*.

O estabelecimento de um endereço IP estático para a VM também pode ser realizado, e é recomendado caso o adaptador de rede em modo *bridge* seja de tecnologia *wireless*, pois a especificação do protocolo Wi-Fi não inclui a implementação do modo *bridge*. Tanto a placa de rede quanto o roteador podem não permitir a utilização de *bridging* devido à implementações estritas do protocolo Wi-Fi, dessa forma, o modo *bridge* nem sempre funciona quando um adaptador de rede *wireless* é utilizado [27]. Para configurar um IP estático, é necessário entrar na VM como superusuário (*root user*, em inglês) no OS e utilizar a ferramenta de linha de comando *nmcli* [28], utilizada para gerenciamento de rede, e executar os comandos ilustrados no Código 3.1. Caso o endereço IP estático seja

utilizado, o acesso ao *Home Assistant* é realizado por meio do endereço IP da VM na porta 8123.

```
nmcli connection edit "HassOS default"
nmcli> print ipv4
nmcli> set ipv4.addresses $endereco_ip_estatico
nmcli> set ipv4.method manual
nmcli> set ipv4.dns $endereco_ip_roteador
nmcli> set ipv4.gateway $endereco_ip_roteador
nmcli> save
nmcli> quit
```

Código 3.1: Configuração de um endereço IP estático para a VM do *Home Assistant*.

No primeiro acesso, o cadastro de um usuário administrador com permissões de superusuário deve ser realizado. A Figura 3.2 mostra um exemplo da interface visual do servidor de automação ao acessar o endereço do servidor e efetuar o *login* com um usuário cadastrado anteriormente.

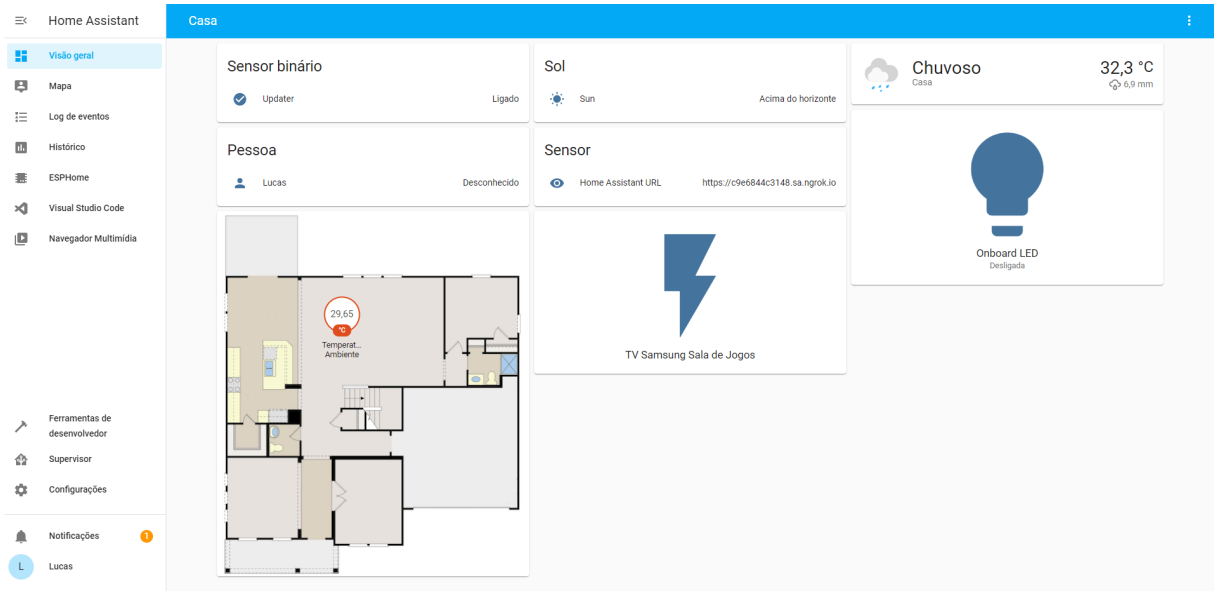

Figura 3.2: Página inicial da interface visual do *Home Assistant*.

O *Supervisor*, responsável pela gerência do sistema, também possui uma loja de extensões, que permite a instalação de ferramentas para o desenvolvimento deste projeto e de aplicações IoT em geral. As extensões instaladas no *Home Assistant* para a implementação deste trabalho são:

- ***ESPHome***: sistema utilizado para controlar dispositivos que utilizam os microcontroladores *ESP8266* e *ESP32* por meio de arquivos de configuração de formato

YAML Ain't Markup Language (YAML) em um ambiente de automação residencial [29];

- ***Mosquitto broker***: utilizada para instalação do *Eclipse Mosquitto*, um *broker* de mensagens *open-source* que implementa o protocolo MQTT [30];
- ***ngrok Client***: realiza a implementação de um túnel *ngrok*[2] sobre HTTP e HTTPS, que expõe o servidor sendo executado localmente para acesso externo. Também provê uma API RESTful para o monitoramento do estado do túnel no *Home Assistant* [31];
- ***Samba share***: permite o compartilhamento de arquivos entre diferentes sistemas operacionais conectados em uma mesma rede por meio do *Server Message Block/-Common Internet File System* (SMB/CIFS) [32];
- ***Visual Studio Code***: habilita a utilização do editor de texto *Visual Studio Code* no próprio navegador para editar os arquivos de configuração do *Home Assistant* [33].

O funcionamento da extensão *ESPHome* é baseado na criação de um *firmware* customizado armazenado em um arquivo binário a partir de um arquivo de configuração no formato YAML. O arquivo de configuração pode possuir diversos tipos de componentes, como sensores, dispositivos, luzes, entre outros que são adicionados automaticamente na interface visual do *Home Assistant*. Além disso, essa extensão possui ferramentas para simplificar o processo de atualização do *firmware* do microcontrolador, como o componente que implementa a tecnologia *Over The Air* (OTA) [29].

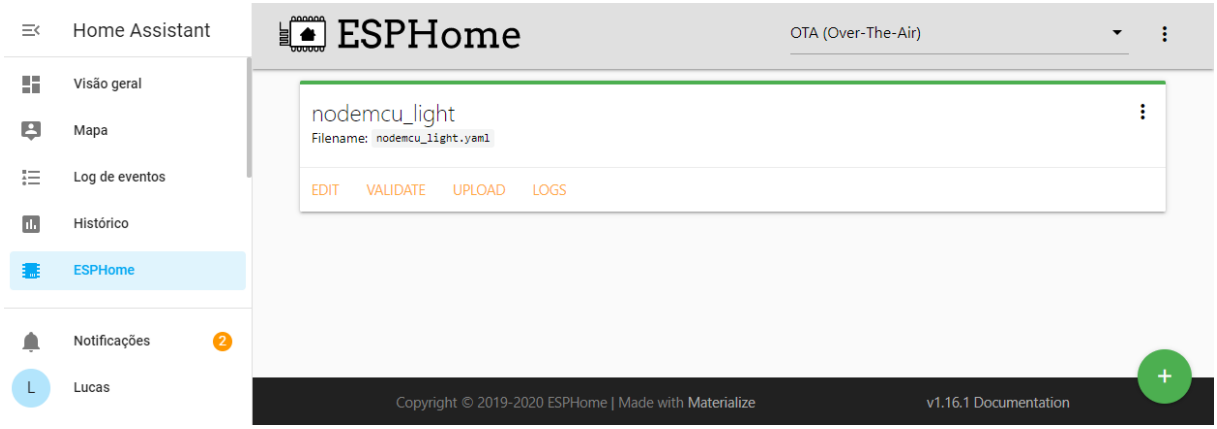

Figura 3.3: Interface visual da extensão *ESPHome*.

[2] `https://ngrok.com`

A interface visual do *ESPHome*, ilustrada na Figura 3.3, apresenta as opções *edit*, *validate*, *upload* e *logs* para um nó adicionado por meio do botão verde no canto inferior direito da interface visual da extensão. O botão *edit* permite a edição do arquivo YAML associado ao nó; *validate* realiza a validação da formatação do arquivo de configuração associado ao nó; *upload* compila o *firmware* de acordo com o arquivo de configuração e atualiza o mesmo por meio da tecnologia OTA ou por conexão *Universal Serial Bus* (USB); *logs* exibe os arquivos de *log* pertencentes ao nó. Além das opções citadas, também é possível compilar e realizar o *download* do arquivo binário por meio da opção *compile*, que se torna visível ao clicar no botão localizado no canto superior direito do nó. Para parametrizar informações sensíveis dos arquivos de configuração, o arquivo YAML *secrets* é utilizado. O arquivo *secrets* específico da extensão pode ser editado ao clicar nas opções no canto superior direito da interface do *ESPHome*, essa parametrização será evidenciada nos códigos que serão ilustrados posteriormente [29].

O método de *upload* adotado no projeto utiliza OTA ao invés de USB, pois o *upload* via USB requer a conexão física da placa de desenvolvimento ao servidor de automação e a reinicialização da extensão para o *ESPHome* reconhecer o dispositivo conectado. Essa necessidade de reinicialização após a conexão da placa se deve à restrições relacionadas ao *docker* na implementação da extensão [29]. Para utilização do OTA, um *firmware* inicial deve estar presente na placa de desenvolvimento, dessa forma, se faz necessário compilar e realizar o *download* do *firmware* inicial e instalação do mesmo na placa de desenvolvimento por meio da ferramenta *ESPHome-flasher* [34]. Após a instalação do *firmware* inicial, as próximas atualizações são feitas por meio da rede Wi-Fi, sem a necessidade de conexão do MCU na porta USB do servidor.

A extensão *ESPHome* foi escolhida para a programação do microcontrolador *ESP8266*, pois a mesma possui suporte ao protocolo MQTT (Requer um *broker* de mensagens MQTT, provido pela extensão *Mosquitto broker*) e também possui uma alternativa, a sua própria API nativa para comunicação com o microcontrolador. Mais detalhes sobre essa API, assim como a exposição da criação do arquivo YAML para programação do *NodeMCU ESP8266* no contexto do projeto e também a descrição dos componentes utilizados no arquivo de configuração serão evidenciados nas seções subsequentes.

Para utilização do túnel proporcionado pela extensão *ngrok Client*, se faz necessária a criação de uma conta no site oficial do *ngrok* para obtenção de uma chave de autorização (*authorization token*, em inglês) que deve ser utilizada na tela de configuração da extensão, ilustrada na Figura 3.4.

Os argumentos de configuração da extensão *ngrok Client* necessários para a criação do túnel são: *log_level*, utilizado para determinar o nível de detalhamento do *log* de dados da extensão; *auth_token*, a chave de autorização associada a conta criada para utilização

Figura 3.4: Arquivo de configuração padrão da extensão *ngrok Client*.

do túnel; *region*, região onde o domínio utilizado para acesso externo do servidor será reservado; *tunnels*, utilizado para configurar o túnel a ser criado, onde *name* é o nome do túnel em si, *proto* é o protocolo que o túnel irá usar, *addr* o número da porta local para onde o tráfego será redirecionado, *inspect* habilita a inspeção de requisições HTTP [35].

Os parâmetros alterados na configuração do *ngrok* para implementação do projeto foram apenas *auth_token*, para a chave de autorização obtida após a criação da conta no *ngrok*, e *region*, para o valor *sa* que representa a região da América do sul. Os argumentos restantes não precisaram ser alterados pois o protocolo utilizado no túnel foi o HTTP dentre as opções HTTP, TCP e TLS, e a porta local para redirecionamento de tráfego é a 8123, utilizada no endereço local de acesso ao servidor de automação.

A conta gratuita não permite a utilização de certificado próprio para encriptar os dados que trafegam no túnel, a encriptação é realizada utilizando um certificado controlado pelo próprio *ngrok*, vulnerabilizando estes dados com relação à ataques *man-in-the-middle* (interceptação, alteração ou registro dos dados trafegados) advindos do próprio *ngrok* [31]. Para solucionar este problema de segurança, é possível utilizar um túnel com protocolo *Transport Layer Security* (TLS) que possui um certificado privado, oferecido pelo *ngrok* mediante assinatura.

O túnel *ngrok* foi adotado no projeto pois ele permite a integração do servidor de automação com o *Google Assistant* sem a necessidade de redirecionamento de portas no roteador, que pode ser um processo dispendioso e até proibido por alguns provedores de serviço de internet (*Internet Service Provider*, em inglês). Essa proibição pode acontecer por meio do bloqueio efetivo de portas (*port blocking*, em inglês), ou devido à utilização do *Carrier-grade Network Address Translation* (CGNAT), onde o roteador do ISP que executa o CGNAT bloqueia o redirecionamento de portas configurado no roteador do consumidor [31]. Mais detalhes sobre como o túnel *ngrok* foi utilizado serão discutidos na seção de exposição do assistente de voz.

As extensões *Samba share* e *Visual Studio Code* são utilizadas apenas para dar suporte

ao processo de desenvolvimento, portanto elas não são essenciais para o funcionamento do projeto como as extensões *ESPHome*, *Mosquitto broker* e *ngrok Client*.

3.3 API nativa *ESPHome*

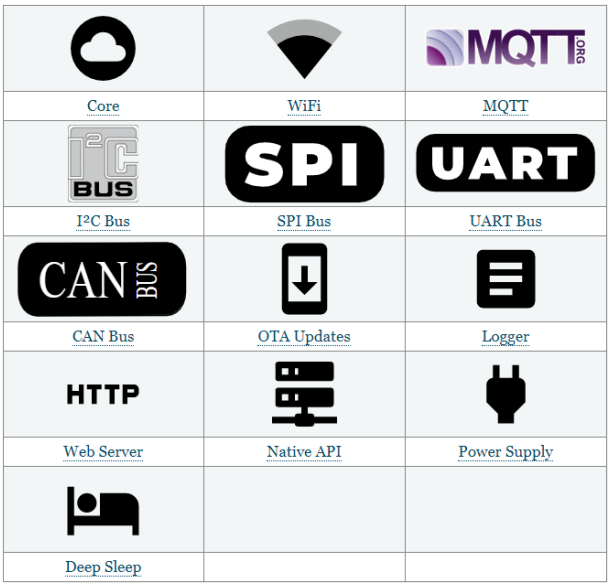

Figura 3.5: *Core components* da extensão *ESPHome* (Fonte: [36]).

O *ESPHome* possui diversos componentes presentes no núcleo de sua implementação (ilustrados na Figura 3.5), dentre eles está o *Native API Component*. Esse componente é descrito como "um protocolo de rede otimizado para comunicação direta com um dispositivo cliente." [37]. No caso deste projeto, o dispositivo cliente é o microcontrolador *ESP8266* presente na placa de desenvolvimento.

A API nativa é baseada em um protocolo TCP customizado que faz utilização de *Protocol Buffers* (também conhecido como *protobuf*) [37]. O *protobuf* é um mecanismo criado e utilizado pelo *Google* para serialização de dados estruturados de forma independente de linguagem ou plataforma [38].

O protocolo customizado da API nativa é estruturado como um *socket* TCP com mensagens binárias curtas e codificadas em formato *protobuf*. Uma mensagem nesse protocolo possui a seguinte estrutura: um *byte* composto por zeros; um inteiro que denota o tamanho do objeto que possui o conteúdo da mensagem em si (objeto mensagem); um inteiro para informar o tipo da mensagem; e finalmente, o objeto mensagem, codificado em formato *protobuf* [39].

O processo de estabelecimento de conexão neste protocolo é composto por quatro etapas:

1. o cliente se conecta ao servidor e envia uma *Hello Request* para se identificar;
2. o servidor responde com uma *Hello Response* e seleciona a versão do protocolo;
3. após receber a *Hello Response*, o cliente tenta se autenticar por meio da senha e de uma *Connect Request*;
4. o servidor responde com uma *Connect Response*, notificando o cliente se a senha estiver incorreta.

Se ocorrer algum erro durante este processo, a conexão deve ser fechada imediatamente tanto pelo cliente quanto pelo servidor, sem a necessidade de troca de mensagens de desconexão [39].

A API *ESPHome* é uma alternativa ao protocolo MQTT, e segundo seus desenvolvedores, ela possui as seguintes vantagens em um contexto onde o *Home Assistant* é utilizado como servidor de automação residencial [37]:

- **Baixa latência**: a API nativa é otimizada para latências muito baixas, na faixa de ms, se tornando imperceptível a olho nu;
- **Configuração simplificada**: sem necessidade de configuração de um *broker* de mensagens, como no protocolo MQTT;
- **Estabilidade**: o protocolo próprio facilita atualizações de estabilidade por parte dos desenvolvedores do *ESPHome*;
- **Maior eficiência**: a codificação das mensagens utilizando o formato otimizado *protobuf* reduz o tamanho das mesmas em até dez vezes;
- **Um ponto único de falha a menos**: cada microcontrolador é seu próprio servidor, a falha na comunicação com o servidor em um MCU não afeta a capacidade de comunicação de outro MCU. No caso do MQTT, se houver uma falha no *broker*, todos os microcontroladores ficam indisponíveis para comunicação.

A utilização desta API nativa só é possível se o servidor de automação for o *Home Assistant*, para outros tipos de servidores de automação, como *openHAB* e *Domoticz*, apenas o protocolo MQTT é suportado na extensão *ESPHome* [37].

3.4 A placa de prototipação *NodeMCU ESP8266*

Segundo seu fabricante *Espressif* [40], o *ESP8266* "entrega um *System-On-a-Chip* (SoC) altamente integrado com a tecnologia Wi-Fi para suprir a contínua demanda dos usuários em eficiência energética, *design* compacto e performance confiável no contexto de IoT.".

O módulo Wi-Fi implementado pelo microcontrolador suporta os protocolos *IEEE 802.11b*, *IEEE 802.11g* e *IEEE 802.11n*, operando na faixa de frequência de 2.4 GHz a 2.4835 GHz, suportando também a pilha TCP/IP completa. A CPU utilizada no *ESP8266* é uma versão melhorada do processador *L106 Diamond series 32-bit* da fabricante *Tensilica*, que opera com velocidade de *clock* de 80 MHz a 160 MHz. Além da CPU e das funcionalidades de Wi-Fi do microcontrolador, ele também possui uma *Static Random Access Memory* (SRAM) integrada no *chip*; uma memória *flash Serial Peripheral Interface* (SPI) externa de 4 Megabytes (MB) para armazenamento de programas de usuário; e diversas interfaces periféricas para utilização de sensores e outros dispositivos. As interfaces relevantes no contexto do projeto são: *General Purpose Input/Output Interface* (GPIO), dezessete pinos implementam essa interface; *Analog-to-Digital Converter* (ADC), apenas um pino possui essa interface; *IR Remote Control*, transmissor implementado no pino MTMS e receptor implementado no pino GPIO5 [40].

Existem diversos tipos de modelos (*ESP-01*, *ESP-07*, *ESP-201*, entre outros) de diferentes fabricantes dos módulos que utilizam o *ESP8266*, os modelos mais recentes são da família *ESP-12*. O modelo *ESP-12E* possui uma antena de tipo *Printed Circuit Board* (PCB) *Trace* integrada no *chip*, assim como o modelo *ESP-12F*, a principal diferença entre os dois modelos é a antena em si, que é melhor otimizada no *ESP-12F*. Os valores de operação do *ESP8266 ESP-12E* se encontram na Tabela 3.1. O modelo *ESP-12E* fabricado pela *Ai-Thinker*, ilustrado na Figura 3.6, foi o adotado no projeto devido à sua utilização na placa de prototipação *NodeMCU V2*.

Tabela 3.1: Valores de operação do *ESP8266 ESP-12E* (Fonte: [40]).

Tensão de operação	3 V a 3,6 V
Corrente de operação	Valor médio de 80 mA
Corrente máxima de um pino GPIO	12 mA
Temperatura de operação	−40° C a 125° C

O *NodeMCU* [42] propriamente dito, é descrito como "um *firmware open-source* baseado na linguagem *Lua* para o SoC *ESP8266*". O *firmware* é implementado na linguagem *C*, e foi inicialmente desenvolvido como um projeto associado às placas de desenvolvimento *NodeMCU ESP8266*, porém atualmente o projeto é suportado pela comunidade, e o *firmware* é compatível com quaisquer módulo *ESP* [42].

A placa de desenvolvimento *NodeMCU ESP8266* em si, possui o *firmware NodeMCU* já instalado no microcontrolador *ESP8266*, assim como um conversor USB/Serial integrado e uma porta Micro-USB. O *firmware* e os componentes integrados na placa têm como intuito facilitar a prototipação de aplicações que utilizam Wi-Fi, principalmente no contexto de projetos IoT.

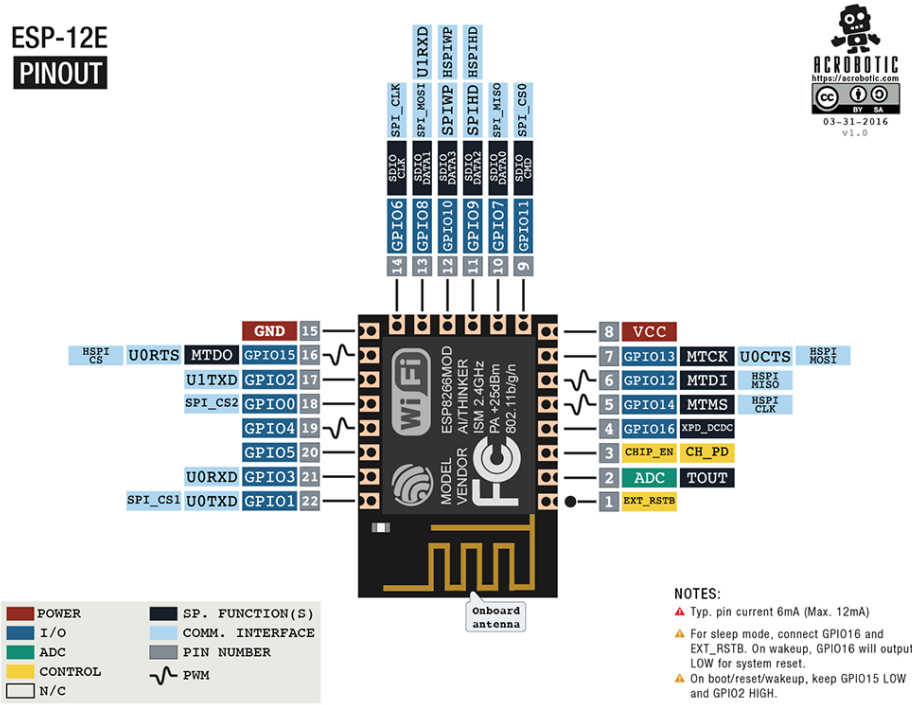

Figura 3.6: Pinagem do microcontrolador *ESP8266 ESP-12E* (Fonte: [41]).

Atualmente, existem três gerações ou modelos da placa *NodeMCU*, são elas [43]:

- ***NodeMCU 0.9 (ESP-12 Module)* ou *NodeMCU V1***: primeira geração da placa. Possui um microcontrolador *ESP8266 ESP12* (antecessor dos modelos *ESP12-E* e *ESP12-F*). O conversor USB/Serial presente nessa versão é o *CH340G*. Além de apresentar um MCU desatualizado, a placa possui dimensões grandes, que dificultam a utilização da mesma em uma *protoboard*;
- ***NodeMCU 1.0 (ESP-12E Module)* ou *NodeMCU V2***: segunda geração da placa, produzida pela fabricante *Amica*. Possui um microcontrolador *ESP8266 ESP12-E*. O regulador de tensão *AMS1117 3.3* é utilizado para ajustar a tensão de alimentação do microcontrolador para 3.3 V. O conversor USB/Serial presente nessa versão é o *CP2102* da fabricante *Silabs*. Essa versão apresenta dimensões reduzidas e possui o espaçamento de pinos ideal para encaixe em uma *protoboard*;
- ***LoLin NodeMCU* ou *NodeMCU V3***: terceira geração da placa, produzida pela fabricante *LoLin*. Possui um microcontrolador *ESP8266 ESP12-E* e o regulador de tensão *AMS1117 3.3*, assim como a segunda geração. O conversor USB/Serial presente nessa versão é o *CH340G*, utilizado também na primeira geração. A largura

dessa versão é maior que a da segunda geração, impossibilitando o encaixe ideal para utilização em *protoboards*.

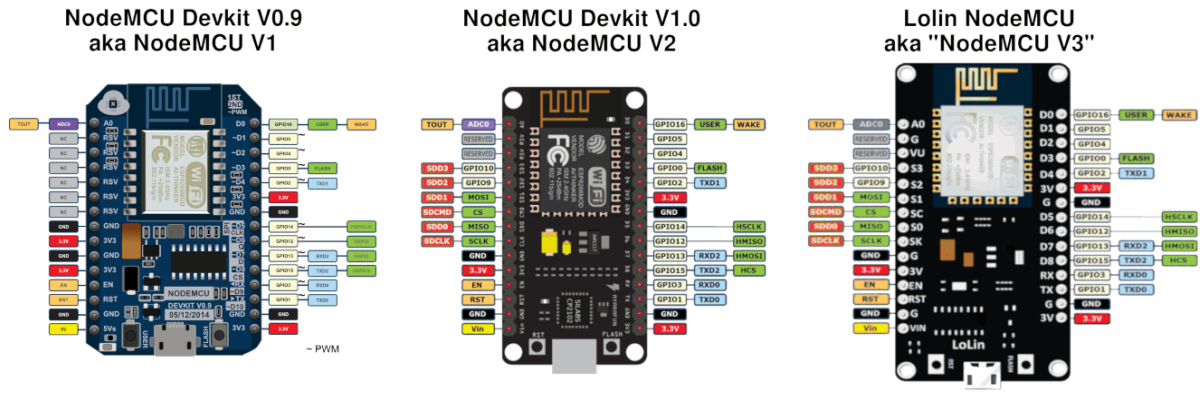

Figura 3.7: Diferentes modelos da placa *NodeMCU*, com suas respectivas pinagens (Fonte: [44]).

A Figura 3.7 ilustra as três gerações, assim como suas pinagens. O modelo adotado no desenvolvimento do projeto foi o *NodeMCU V2*, ilustrado em detalhes na Figura 3.8. Não há grandes diferenças entre a segunda e a terceira gerações da placa, o fator decisivo para a escolha da segunda geração foi o seu encaixe projetado para *protoboards* comuns. A placa foi adquirida pelo valor de cinquenta e cinco reais com o cabo Micro-USB incluso.

Com a exposição da extensão *ESPHome* do servidor de automação *Home Assistant* e da placa de desenvolvimento *NodeMCU*, a programação do microcontrolador em si será abordada a seguir.

Antes de implementar as funções de um controle remoto propriamente dito, o desenvolvimento de um programa para testar a comunicação (utilizando tanto o protocolo MQTT, quanto a API nativa) entre o MCU e o servidor de automação foi elaborado. Esse programa tem como objetivo controlar um *Light Emitting Diode* (LED) presente na placa de desenvolvimento pela interface visual do servidor de automação.

Para adicionar um nó na extensão *ESPHome*, se faz necessária a criação de um arquivo de configuração YAML que será utilizado para geração do *firmware* para a placa de desenvolvimento associada a este nó.

A primeira parte do arquivo YAML contém a configuração do componente núcleo do *ESPHome* (*ESPHome Core Component*, em inglês), ilustrada no Código 3.2 e necessária para a criação do *firmware* para a placa de desenvolvimento. Os parâmetros básicos e obrigatórios para configuração são: *name*, nomeia o nó a ser adicionado. O nome deve ser único e deve conter apenas letras minúsculas, dígitos, hífen e sublinhado (*underscore*, em inglês); *platform*, determina qual o microcontrolador está presente na placa (*ESP32*

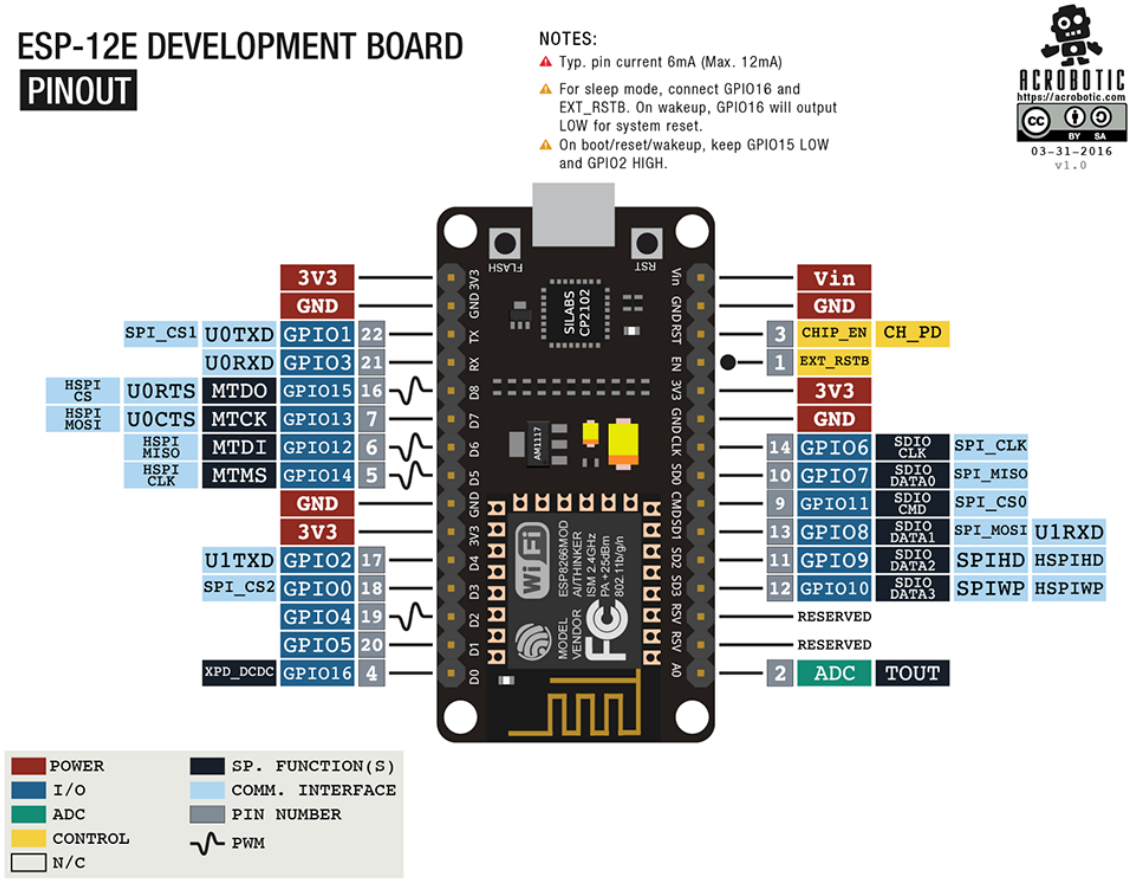

Figura 3.8: Pinagem da placa de desenvolvimento *NodeMCU 1.0 (ESP-12E Module)*, também denominada de *NodeMCU V2* (Fonte: [41]).

ou *ESP8266*); *board*, indica o modelo da placa de desenvolvimento em si [45]. Os valores *ESP8266* para *platform* e *nodemcuv2* para *board* foram utilizados no projeto devido ao modelo da placa de desenvolvimento *NodeMCU V2* ter sido adotado.

```
esphome:
  name: nodemcu
  platform: ESP8266
  board: nodemcuv2
```

Código 3.2: Configuração do núcleo do *ESPHome*.

A configuração do componente Wi-Fi, ilustrada no Código 3.3, é utilizada para conectar a placa de desenvolvimento a um ponto de acesso de rede, possibilitando a utilização do protocolo MQTT ou da API nativa para comunicação entre a placa de desenvolvimento e o servidor de automação. Os parâmetros necessários para configuração são: *ssid*, o nome do ponto de acesso Wi-Fi que a placa deve se conectar; *password*, a senha do ponto de

acesso Wi-Fi [46]. O ponto de acesso Wi-Fi deve operar na faixa de frequência de 2.4 GHz para a conexão ser realizada, pois o microcontrolador não suporta redes Wi-Fi que operam em 5 GHz.

Para melhorar o tempo de conexão da placa com a rede Wi-Fi e evitar problemas com o *Dynamic Host Configuration Protocol* (DHCP), protocolo que permite a utilização de um IP dinâmico pelo *NodeMCU*, um IP estático foi configurado para a placa. Os parâmetros necessários para a configuração desse endereço IP estático são: *static_ip*, endereço IP da placa de desenvolvimento em si; *gateway*, endereço IP do roteador; *subnet*, sub-rede da rede em si [46]. O endereço IP do roteador e a sub-rede foram obtidos por meio do comando *ipconfig*, executado no *prompt* de comando do *Windows*.

Além dos parâmetros de configuração obrigatórios e da configuração do endereço IP estático, a configuração opcional do modo *access point* foi realizada. O modo *access point* é utilizado para a criação de um ponto de acesso Wi-Fi, com os valores de *ssid* e *password* especificados no arquivo de configuração, de forma automática quando a conexão à rede Wi-Fi especificada não conseguiu ser realizada dentro do tempo especificado pelo parâmetro *ap_timeout* [46].

O componente *captive portal*, também configurado no Código 3.3, adiciona uma interface *web* para o modo *access point*, ou seja, o acesso à essa interface é realizado por meio da conexão ao ponto de acesso do modo *access point*. A interface permite reconfigurar os parâmetros de configuração da rede Wi-Fi, assim como atualizar o *firmware* da placa de desenvolvimento [47].

```
wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  manual_ip:
    static_ip: !secret manual_ip_static_ip
    gateway: !secret manual_ip_gateway
    subnet: !secret manual_ip_subnet

  ap:
    ssid: "Nodemcu Fallback Hotspot"
    password: !secret ap_password
    ap_timeout: 1min

captive_portal:
```

Código 3.3: Configuração do Wi-Fi e *captive portal*.

Para fins de depuração, a configuração do componente *logger* foi realizada e é ilustrada no Código 3.4. O *logger* exibe todas as mensagens de *log* tanto por meio de uma porta serial *Universal Asynchrounous Receiver/Transmiter* (UART) quanto por meio de tópicos MQTT. As propriedades utilizadas no arquivo de configuração são: *baud_rate*, define o *baud rate*, ou seja, a taxa de *bits* por segundo transmitida para a porta serial de exibição do *log*; *level*, filtra as mensagens de *log* de acordo com a severidade, os níves de severidade em ordem decrescente são *NONE*, *ERROR*, *WARN*, *INFO*, *DEBUG*, *VERBOSE* e *VERY_VERBOSE* [48]. Os valores utilizados no projeto foram 0 para *baud_rate*, que desabilita o *log* na porta serial UART0 (composta pelos pinos GPIO1 e GPIO3, utilizada por padrão como porta serial de *log*), e *DEBUG* para *level*, valor adotado como padrão para o nível de severidade.

```
1 logger:
2   baud_rate: 0
3   level: DEBUG
```

Código 3.4: Configuração do *logger*.

O cliente MQTT é necessário para implementar o protocolo MQTT na placa de desenvolvimento. O componente requer a utilização da extensão *Mosquitto broker* no servidor de automação, pois a extensão realiza a instalação do *broker Eclipse Mosquitto*, necessário para a comunicação MQTT. A configuração do componente, ilustrada no Código 3.5, possui os seguintes parâmetros: *broker*, endereço IP do *broker*; *port*, a porta do *broker*; *username*, nome do usuário utilizado para autenticação com o *broker*; *password*, senha associada ao nome de usuário; *discovery*, habilita a descoberta automática de dispositivos MQTT na inicialização do *Home Assistant*, possibilitando a adição destes dispositivos na interface visual do servidor; *discovery_prefix*, prefixo utilizado para a descoberta de dispositivos MQTT do servidor de automação; *topic_prefix*, prefixo utilizado para as mensagens associadas à placa de desenvolvimento; *keepalive*, define o tempo que o *socket* se mantém aberto [49].

Após a instalação do *broker*, a criação de um usuário específico para a autenticação do *broker* foi realizada no servidor de automação. Além da criação do usuário, a integração *Local IP Address* [50] foi configurada para expor o endereço IP local do *Home Assistant*. O endereço IP local é utilizado no parâmetro *broker*. O atributo *port* possui como valor padrão 1883, que é porta onde o *broker* instalado no servidor de automação é executado. As credenciais do usuário de autenticação MQTT criado anteriormente são utilizadas nos parâmetros *username* e *password*. Com relação à descoberta de dispositivos, *discovery* e *discovery_prefix* são configurados em seus valores padrões para a habilitar a descoberta de dispositivos MQTT. O nome associado à placa, configurado no Código 3.2, é utilizado no *topic_prefix*, que tem como função a filtragem dos dispositivos MQTT entre as diferentes

possíveis placas de desenvolvimento. O *socket* MQTT é configurado em seu valor padrão de 15 segundos, diminuir este valor aumenta o tráfego na rede Wi-Fi, mas também pode melhorar a estabilidade da comunicação.

```
mqtt:
  broker: !secret home_assistant_local_ip
  port: 1883
  username: !secret mqtt_username
  password: !secret mqtt_password
  discovery: true
  discovery_prefix: homeassistant
  topic_prefix: nodemcu
  keepalive: 15s
```

Código 3.5: Configuração do cliente MQTT.

A alternativa ao cliente MQTT, a API nativa *ESPHome* (discutida na seção anterior) possui sua configuração ilustrada no Código 3.6. Os atributos necessários para a configuração da API são: *port*, possui como valor padrão 6053 e define a porta onde o servidor da API é executado na placa de desenvolvimento; *password*, utilizado para configurar a senha de acesso ao servidor da API. Para disponibilizar os dispositivos API de um nó específico na interface do servidor de automação, se faz necessária a utilização de uma integração *ESPHome* [51], essa integração exige os seguintes parâmetros: *Host*, endereço IP do nó a ser integrado; *Port*, mesmo valor de porta utilizado na configuração do componente; *Password*, senha utilizada na configuração do componente [37].

No caso em que a API nativa é utilizada para comunicação entre a placa de desenvolvimento e o servidor de automação ao invés do protocolo MQTT, não há necessidade da instalação da extensão *Mosquitto broker* e nem da integração *Local IP Address*, pois elas são utilizadas para executar o *broker* de mensagens MQTT e configurar o protocolo MQTT, respectivamente.

```
api:
  port: 6053
  password: !secret api_password
```

Código 3.6: Configuração da API nativa.

O último *core component* do *Home Assistant* configurado no contexto do projeto foi o componente de atualização OTA. A configuração, ilustrada no Código 3.7, possui os seguintes parâmetros: *safe_mode*, habilita o modo de segurança que desabilita todos os componentes, exceto *logger* via porta serial, Wi-Fi e OTA, para permitir a atualização do *firmware* no caso de falhas sucessivas no processo de inicialização da placa (também

conhecido como *boot*); *password*, senha necessária para a realização de atualizações; *port*, porta utilizada para a atualização, possui valor padrão 8266, caso o microcontrolador da placa de desenvolvimento seja o *ESP8266*, e 3232, caso seja o *ESP32*; *num_attempts*, número máximo de falhas sucessivas no processo de *boot* antes da ativação do modo de segurança [52].

```
ota:
  safe_mode: true
  password: !secret ota_password
  port: 8266
  num_attempts: 10
```

Código 3.7: Configuração da atualização OTA.

Com a configuração dos componentes que compõem o núcleo do *ESPHome* finalizada, a criação de qualquer programa utilizando a placa de desenvolvimento se faz possível. Os componentes utilizados no programa elaborado (ilustrado no Código 3.8) para testar a comunicação entre o servidor e a placa por meio do controle de um LED na interface do *Home Assistant* serão abordados a seguir.

Existem dois LEDs na placa de desenvolvimento: o primeiro está presente no microcontrolador *ESP8266* (Figura 3.6), identificado como o pino *D*4 no modo invertido; o segundo se encontra no *NodeMCU* (Figura 3.8), ao lado do conversor USB/Serial embutido, identificado como o pino *D*0, também no modo invertido [53].

Para acender o LED no pino *D*0 invertido, um componente *output* é configurado no pino utilizando um *Pulse Width Modulation* (PWM) via *software* no microcontrolador *ESP8266* [54] de frequência padrão de 1 kHz. O PWM via *software* pode oscilar quando o tráfego na rede Wi-Fi, o mesmo foi adotado pois o PWM via *hardware* não está disponível para configuração no microcontrolador *ESP8266* [54]. A configuração em si, ilustrada no Código 3.8, possui os seguintes argumentos: *plataform*, define a saída do componente que pode ser binária ou ponto flutuante [55], nesse caso a saída é configurada com o valor *esp8266_pwm*, sendo portanto ponto flutante; *id*, identificador do componente, que será utilizado no controle de estado; *frequency*, frequência do PWM; *pin* [56], pino que terá a saída configurada, que nesse caso possui os argumentos *number* e *inverted*, que definem o número do pino e se o mesmo está no modo invertido, respectivamente.

O controle de estado do LED na interface do servidor de automação é realizado por meio de uma entidade *light* [57] que permite apenas os estados ligado e desligado. Os argumentos utilizados na configuração da entidade são: *plataform*, define o tipo da luz em si, nesse caso a luz é um LED que não possui opções de coloração ou de luminosidade, sendo portanto uma luz binária (ligada ou desligada); *id*, identificador do componente; *name*, nome dado a luz em si, e que é utilizado para controlar a mesma na interface do

Home Assistant; *output*, componente de saída associado à luz, que é ativado ou desativado de acordo com o estado.

```
output:
  - platform: esp8266_pwm
    id: nodemcu_led_output
    frequency: 1000 Hz
    pin:
      number: D0
      inverted: true

light:
 - platform: binary
   id: nodemcu_led
   name: "Onboard LED"
   output: nodemcu_led_output
```

Código 3.8: Programação do LED embutido na placa de desenvolvimento.

Após a programação do LED, o mesmo pode ser ligado ou desligado via interface visual do *Home Assistant*. O próximo passo, que será discutido na próxima seção, é a integração do servidor de automação com o *Google Assistant* para proporcionar o controle dos dispositivos presentes na interface por meio da voz.

3.5 O assistente de voz *Google Assistant*

O *Google Assistant* [11] é um assistente virtual desenvolvido pela *Google* que utiliza tecnologias de inteligência artificial e NLP para facilitar a realização de tarefas em dispositivos como *smartphones*, *smart speakers* (auto-falantes inteligentes, em português), *smart displays*, automóveis, televisores *smart*, computadores, *tablets* e *smart watches*. A principal forma de interação com o assistente é pela voz, porém o mesmo também suporta interação não-verbal, como por exemplo, via teclado em *smartphones* por meio do aplicativo do *Google Assistant*. No contexto do trabalho, tanto o *Google Home Mini* (ilustrado na Figura 3.9, quanto o aplicativo do assistente foram utilizados como forma de interação com o assistente virtual.

A integração do *Google Assistant* com o servidor de automação *Home Assistant*, pode ser realizada de duas formas: pelo serviço de assinatura do *Home Assistant*, denominado *Home Assistant Cloud*, que habilita tanto o uso do *Google Assistant* quanto do *Amazon Alexa* e também facilita o acesso remoto ao servidor de automação; por meio da plata-

Figura 3.9: O auto-falante inteligente *Google Home Mini* (Fonte: [11]).

forma de desenvolvimento do assistente de voz, a *Actions on Google*[3], de forma gratuita utilizando uma conta da *Google* [58]. A criação de um projeto na plataforma de desenvolvimento foi a forma de integração utilizada e será descrita a seguir.

Ao realizar a criação de um projeto de categoria *smart home* na *Actions on Google*, se faz necessário nomear a ação que será vinculada a este projeto. O nome da ação é fundamental para o funcionamento da integração, pois o mesmo é utilizado para habilitar a invocação do *Home Assistant* no *Google Assistant*, que será explanada posteriormente. Existem dois tipos de invocação no assistente de voz [59]: a invocação explicita, onde o usuário realiza a invocação da ação por meio do nome da mesma, como por exemplo, utilizando a frase "Ok *Google*, fale com (nome da ação)"; a invocação implícita, que ocorre quando o usuário não utiliza o nome da ação em si, e sim descreve uma tarefa que deve ser realizada, alguns exemplos são "Ok *Google*, ligar o ar condicionado", "Ok *Google*, aumentar o volume da televisão", entre outros. No contexto do projeto, a invocação implícita foi idealizada como a principal forma de interação do usuário com o assistente de voz, devido a esse tipo de invocação ser mais intuitivo e direto.

Após a nomeação da ação do projeto, a configuração do endereço URL de *fulfillment* é utilizada para a comunicação efetiva do *Google Assistant* com o *Home Assistant*. O *fulfillment* [60] é descrito como "um código implementado como um *webhook* que permite a geração de respostas dinâmicas para tarefas de *smart home*", habilitando assim a utilização da tecnologia NLP provida pelo *Google Cloud* para extração de dados da conversação com o usuário no projeto, provendo respostas e executando tarefas no *back-end*. O processo de *fulfillment* (ilustrado na Figura 3.10) é iniciado quando o usuário invoca a ação, o assistente de voz então realiza requisições ao endereço de *fulfillment*, implementado no servidor de automação, que processa a requisição e à responde.

O acesso ao endereço de *fulfillment* na rede local é realizado por meio da adição do subdiretório `/api/google_assistant` no endereço de acesso ao *Home Assistant*, identi-

[3] `https://console.actions.google.com`

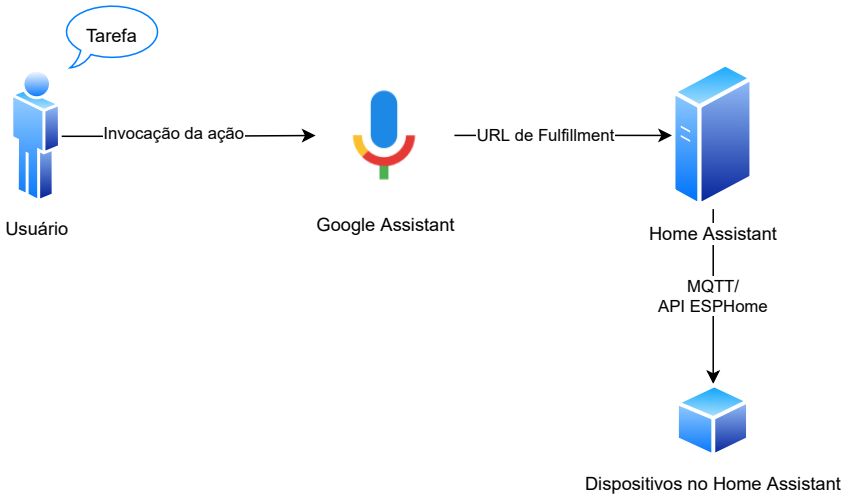

Figura 3.10: Processo de *fulfillment*.

ficado anteriormente como `http://homeassistant.local:8123` ou endereço IP estático do servidor na porta 8123. Para viabilizar a comunicação entre o assistente de voz e o servidor de automação, se faz necessário a exposição do endereço local de *fulfillment* em redes externas. A extensão do *Home Assistant* adotada no trabalho para estabelecimento de acesso remoto foi a *ngrok Client*, cuja escolha foi justificada anteriormente na seção do servidor de automação. Uma alternativa à extensão adotada, que requer a utilização de redirecionamento de portas para funcionar, é a extensão *Duck DNS* [61], que é um serviço gratuito de *Domain Name System* (DNS) dinâmico que permite a utilização de um sub-domínio de `duckdns.org` para externalização do endereço local de acesso ao *Home Assistant*.

A configuração da API RESTful disponibilizada pela extensão *ngrok* [31], ilustrada no Código 3.9, é utilizada para a criação de uma entidade do tipo *sensor* [62] na interface visual do servidor de automação para exibir o valor do endereço de acesso remoto do servidor. Essa configuração é realizada no arquivo `configuration.yaml` do diretório `config`, que se encontra no diretório raiz do servidor de automação.

```
sensor:
  - platform: rest
    resource: http://localhost:4040/api/tunnels/hass
    name: Home Assistant URL
    value_template: "{{ value_json.public_url }}"
```

Código 3.9: Configuração do sensor de monitoramento do túnel *ngrok*.

Com o acesso remoto ao endereço de *fulfillment* (sub-domínio de `ngrok.io` acrescido do sub-diretório `/api/google_assistant`), e consequentemente ao servidor de auto-

mação, a próxima etapa da configuração do projeto é a vinculação de conta (*account linking*, em inglês) [63]. O protocolo de autorização adotado na *Actions on Google* para a vinculação de conta é o *OAuth 2.0*[4], e as credenciais do cliente *OAuth 2.0* necessárias para essa vinculação são: *Client ID*, que é o *OAuth 2.0 Client ID* utilizado para identificar o *Google* no servidor de automação, sendo composto pelo endereço URL `https://oauth-redirect.googleusercontent.com/r` acrescido do identificador do projeto da plataforma de desenvolvimento (*Project ID*, gerado de acordo com o nome do projeto) no sub-diretório; *Client secret*, *OAuth 2.0 Client Secret* associado ao *OAuth 2.0 Client ID* utilizado para identificação do *Google*; *Authorization URL*, endereço externo do método *authorize* implementado pela API de autenticação do servidor de automação, que é composto pelo endereço de acesso remoto acrescido do sub-diretório `/auth/authorize`; *Token URL*, endereço externo do método *token* implementado pela API de autenticação do servidor de automação, que é composto pelo endereço de acesso remoto acrescido do sub-diretório `/auth/token`. Os escopos de autorização do cliente *OAuth 2.0*, ou seja, as permissões que o usuário deve consentir para utilização do serviço são configuradas para *email* e *name*. Com a conclusão da configuração do cliente *OAuth 2.0*, a ativação da ação é realizada por meio do botão *Test* na tela de vinculação de conta.

A configuração do projeto na plataforma de desenvolvimento em si se encontra finalizada, para suportar a sincronização dos dispositivos do *Home Assistant* no aplicativo *Google Home* e a comunicação ativa de estado dos dispositivos com o *Google Cloud*, a criação de uma conta de serviço nas credenciais de APIs e serviços[5] e a ativação da API *HomeGraph* na biblioteca de APIs[6] no painel de controle da *Google Cloud Plataform* associada ao projeto da *Actions on Google* são necessárias. A sincronização pode ser realizada pelo comando de voz "Ok *Google*, sincronizar dispositivos", permitindo a atualização dos dispositivos no servidor de automação sem a necessidade de revinculação de conta [63].

A conta de serviço é utilizada como uma credencial para autenticação entre os servidores no nível da aplicação por meio de contas robô (*robot accounts*, em inglês), enquanto as credenciais do cliente *OAuth 2.0*, explanadas anteriormente, são utilizadas para solicitar o consentimento do usuário no acesso de dados do *Home Assistant* por parte do *Google Assistant*. Para realizar a criação da conta apenas o nome da conta de serviço é necessário, pois o identificador é gerado de acordo com o nome. O papel exercido pela conta é o de criador de *token*, que permite a criação de *tokens* de acesso *OAuth 2.0*. Os argumentos requeridos no arquivo de integração são o *email* da conta de serviço e a chave privada que é gerada e armazenada em um arquivo JSON, que também contém outras informações que serão utilizadas na configuração de integração no servidor de automação.

[4] `https://oauth.net/2/`

[5] `https://console.cloud.google.com/apis/credentials`

[6] `https://console.cloud.google.com/apis/library/homegraph.googleapis.com`

O *Home Graph* [64] é essencial para projetos de *smart home* desenvolvidos na plataforma *Actions on Google*, sendo definido como "uma base de dados que armazena e provê dados contextuais sobre a casa e seus dispositivos". A base de dados divide as informações em: estruturas, como casa ou escritório de trabalho; ambientes, como quarto, sala de estar e cozinha; dispositivos, como *Google Home Mini*, lâmpadas, televisores. Essas informações são disponibilizadas para o *Google Assistant* com o intuito de executar determinadas tarefas em um contexto apropriado. Dados relacionados aos estados dos dispositivos são efêmeros, ou seja, não são armazenados a longo prazo. O *Home Graph* auxilia o assistente de voz por meio de duas principais formas: execução de comandos implícitos, exemplificando, ao requisitar a tarefa "Ok *Google*, desligue a luz", a luz presente no mesmo cômodo em que o usuário e o auto-falante inteligente se encontram deve ser desligada; e maior controle sobre alvos explícitos, como por exemplo, ao requisitar a tarefa "Ok *Google*, ligue as luzes da sala de estar", todas as luzes do ambiente, independente do fabricante, devem ser ligadas, mesmo quando o usuário se encontra em um outro cômodo da casa. A ativação da API *HomeGraph* no projeto habilita a utilização desse mapa lógico nos dispositivos associados ao servidor de automação.

Com a finalização da configuração do projeto na plataforma de desenvolvimento e no painel de controle do *Google Cloud*, a próxima etapa para integração do assistente de voz com o servidor de automação é a configuração do componente *google_assistant* no arquivo `configuration.yaml` do servidor de automação, que utiliza informações armazenadas no arquivo JSON de chave privada associado à conta de serviço.

Os parâmetros utilizados para a configuração do componente *google_assistant*, ilustrada no Código 3.10, são [63]: *project_id*, identificador do projeto criado na plataforma de desenvolvimento, presente no arquivo JSON de chave privada; *service_account*, que apresenta dois argumentos *private_key* e *client_email*, ambos se encontram presentes no arquivo JSON, o primeiro é a chave privada em formato *Privacy-Enhanced Mail* (PEM) da conta de serviço, e o segundo é o endereço de *email* utilizado pela conta de serviço; *report_state*, que habilita a comunicação ativa de mudança nos estados dos dispositivos com o *Google Cloud*; *exposed_domains*, lista de domínios de entidades presentes no servidor de automação que são expostas para o assistente de voz. Além dos parâmetros ilustrados no Código 3.10, o componente possibilita a configuração de entidades específicas por meio do parâmetro *entity_config* que possui as variáveis *name*, *expose*, *aliases* e *room* que são utilizadas para definir um nome customizado da entidade para o *Google Assistant*, a exposição da entidade para o assistente de voz, os apelidos que podem se referir à entidade e o ambiente que a entidade se encontra, respectivamente. O servidor de automação deve ser reinicializado para a aplicação das configurações do componente de integração do assistente de voz.

```
google_assistant:
  project_id: !secret project_id
  service_account:
    private_key: !secret service_account_private_key
    client_email: !secret service_account_client_email
  report_state: true
  exposed_domains:
    - light
    - sensor
    - switch
    - climate
    - media_player
```

Código 3.10: Configuração da integração do assistente de voz *Google Assistant* com o servidor de automação.

A última etapa da integração é realizada no aplicativo *Google Home*, utilizando a mesma conta *Google* associada ao projeto criado na plataforma de desenvolvimento. Ao adicionar um novo dispositivo no aplicativo existem duas opções: configurar um novo dispositivo, como por exemplo, *smart speakers* e *smart displays*; vinculação de um dispositivo já existente ou de serviços. A vinculação de serviço, onde esse serviço é o projeto da plataforma *Actions on Google*, é a opção utilizada para integração do *Google Assistant* com o servidor de automação. Ao selecionar o serviço, que é nomeado como "[test]" seguido do nome do projeto, a tela de *login* do *Home Assistant* é acessada pelo aplicativo, onde as credenciais de acesso ao servidor de automação são requisitadas para permitir o acesso dos dados do servidor ao serviço. Ao finalizar esse processo, as entidades presentes no servidor são disponibilizadas para controle de voz pelo *Google Assistant*.

O projeto que é utilizado como serviço possui o marcador "[test]" em seu nome, pois o mesmo não foi publicado como os outros serviços que aparecem no aplicativo, se encontrando em uma fase de teste que possui duração de trinta dias. Após o período de teste, a sincronização de dispositivos é desabilitada, para habilitar novamente basta acessar o projeto na *Actions on Google* e clicar novamente no botão *Test* na tela de vinculação de conta para reativação do período de testes do serviço [63].

Após a finalização do processo de integração, o LED programado no Código 3.8 pode ser controlado por meio do assistente de voz, mais detalhes serão apresentados na seção específica deste código de teste no próximo capítulo.

3.6 Infravermelho: receptor e transmissor

O controle via interface *web* do servidor de automação e o controle por voz via *Google Assistant* dos dispositivos presentes nessa interface foram configurados nas seções anteriores. Para simular o funcionamento de um controle remoto infravermelho de qualquer aparelho eletrônico que utiliza essa tecnologia, dois componentes são necessários: um receptor IR, utilizado para captar os sinais infravermelhos emitidos pelos controles individuais dos aparelhos; um transmissor IR, que transmitirá os sinais captados para comandar os aparelhos, emulando os controles proprietários. Os componentes foram adquiridos em um conjunto que possui um controle genérico, um receptor e um LED infravermelhos.

Tabela 3.2: Valores de operação do módulo receptor infravermelho *KY-022* (Fonte: [65]).

Tensão de operação	2, 7 V a 5, 5 V
Corrente de operação	0, 4 mA a 1, 5 mA
Distância de recepção	18 m
Ângulo de recepção	±45° ou 90°
Frequência da onda portadora	38 kHz

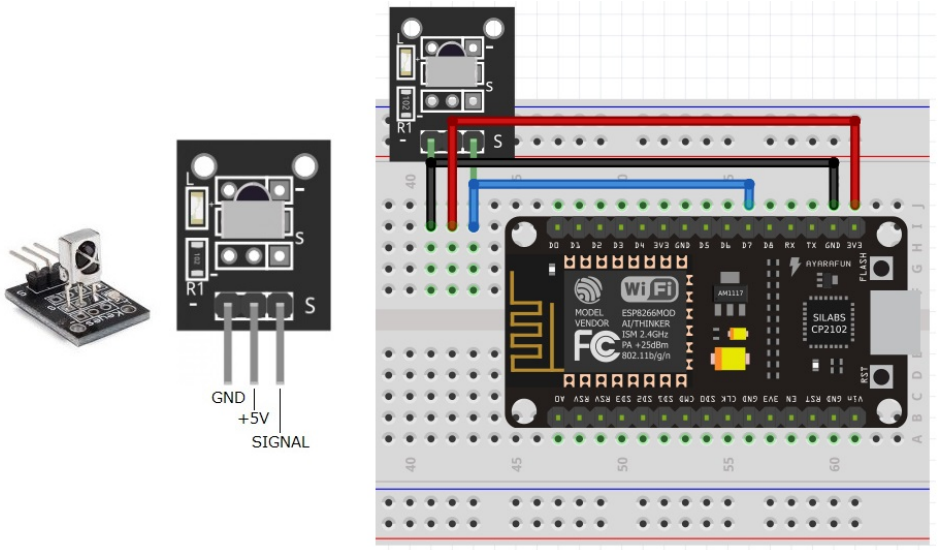

Figura 3.11: Módulo receptor IR *KY-022* e esquemático do circuito receptor infravermelho (Fonte: [65]).

O módulo receptor IR que exerce a função de captar sinais infravermelhos dos controles remotos é o *KY-022*, ilustrado na Figura 3.11. O módulo é composto por um fotorreceptor IR 1838, um resistor de 1 kΩ e um LED [65]. O pino de entrada e saída D7, também conhecido como GPIO13, foi o escolhido entre os pinos sem capacidade de

PWM para recebimento dos sinais IR no MCU da placa de desenvolvimento. Os esquemáticos dos circuitos receptor IR (ilustrado na Figura 3.11), transmissor IR (ilustrado na Figura 3.12) e sensor de temperatura (ilustrado na Figura 3.13) foram projetados na ferramenta *Fritzing*[7].

Para utilizar o receptor IR na placa de desenvolvimento, o componente da extensão *ESPHome remote_receiver* [66] é configurado no arquivo YAML de programação do microcontrolador. Esse componente habilita o recebimento e a decodificação de sinais de controles remotos IR e de sinais *Radio frequency* (RF) na faixa de 433 MHz.

Os argumentos definidos na configuração do *remote_receiver*, ilustrada no Código 3.11, são [66]: *pin* define o pino D7 como o receptor dos sinais IR, o atributo *inverted* nesse caso não é utilizado para identificar o pino na placa de desenvolvimento como no Código 3.8, mas sim para inverter os valores recebidos no pino; *dump* define como os valores decodificados são visualizados no *log* do microcontrolador. As opções disponíveis para visualização são: códigos IR associados a aparelhos eletrônicos de uma determinada fabricante (*lg*, *nec*, *panasonic*, *pioneer*, *jvc*, *samsung* e *sony*); códigos IR que utilizam o protocolo *RC-5* desenvolvido pela empresa *Philips* e adotado por diversas fabricantes ocidentais (*rc5*); códigos RF de frequência 433 MHz (*rc_switch*); códigos em sua forma bruta, ou seja, sem associação com um protocolo específico, útil para decodificação de códigos arbitrários (*raw*). O valor de visualização *all* adotado no Código 3.11 ilustra todos os protocolos possíveis que o sinal recebido pode pertencer.

```
1 remote_receiver:
2   pin:
3     number: D7
4     inverted: true
5   dump: all
```

Código 3.11: Configuração do receptor infravermelho.

Tabela 3.3: Valores de operação do LED 5 mm emissor infravermelho.

Tensão de operação	1,6 V *Direct current* (DC)
Corrente de operação	20 mA
Distância de transmissão	1 m
Comprimento de onda	940 nm
Ângulo de transmissão	15° a 30°

Com a configuração do receptor infravermelho, a parte de captura de sinais de qualquer controle remoto IR independente de seu fabricante se torna possível. A transmissão desses

[7] `https://fritzing.org`

sinais aos aparelhos desejados necessita de um LED IR, ilustrado na Figura 3.12. O LED adquirido possui 5 mm de diâmetro e os valores de operação aproximados (não foi possível determinar o modelo exato do LED) se encontram na Tabela 3.3. O pino de entrada e saída D6 da placa de desenvolvimento, também conhecido como GPIO12, foi o escolhido entre os pinos com capacidade de PWM para transmissão dos sinais na frequência de modulação de 38 kHz, utilizada para comunicação por infravermelho.

Tomando como base os valores ilustrados na Tabela 3.1 e na Tabela 3.3, a corrente máxima de um pino GPIO do microcontrolador (12 mA) é menor que o valor da corrente de operação do LED IR (20 mA), esse fato permite o funcionamento do LED conectando-o de forma direta seu ânodo ao pino GPIO e seu cátodo ao pino de referência. Essa solução não é ideal pois exige muita corrente do microcontrolador e não atinge o valor da corrente de operação, prejudicando o alcance dos sinais IR enviados pelo LED.

Para se aproximar ao máximo do valor da corrente de operação do LED IR sem requerer grandes quantidades de corrente do MCU, a utilização de um transistor para aumentar a corrente se faz necessária. O transistor selecionado foi o *2N2222* do tipo NPN, ilustrado na Figura 3.12. O ganho de corrente DC (h_{FE}) desse transistor é de valor 100 e a queda de tensão estimada entre a base e o emissor (V_{BE}) é de $0,7$ V [67].

Utilizando a primeira lei de Ohm [68], os seguintes cálculos foram realizados para determinar um valor de resistência mínimo para limitar a corrente do LED IR ao seu valor de operação:

$$\begin{aligned} V_{CC} - V_{LED} &= I_{LED} \times R_{LED} \\ 3,3V - 1,6V &= 20mA \times R_{LED} \\ R_{LED} &= 85\Omega \end{aligned} \tag{3.1}$$

Uma resistência de 100 Ω foi conectada em série com o LED IR, limitando a corrente de alimentação para o valor de 17 mA, valor que se aproxima de melhor forma à corrente de operação, possibilitando assim um melhor alcance dos sinais enviados e mantendo a durabilidade do LED.

O ganho de corrente DC de um transistor NPN possui a seguinte equação [69]:

$$h_{FE} = \frac{I_C}{I_B} \tag{3.2}$$

Com os valores do ganho (h_{FE}) e da corrente do coletor (I_C) conhecidos, temos que a corrente da base (I_B) que será exigida do pino GPIO do microcontrolador é de:

$$\begin{aligned} 100 &= \frac{17mA}{I_B} \\ I_B &= 170\mu A \end{aligned} \tag{3.3}$$

Para obter o valor da resistência da base do transistor, que se encontra conectada ao pino GPIO do MCU, a primeira lei de Ohm [68] é utilizada novamente:

$$\begin{aligned} V_{CC} - V_{BE} &= I_B \times R_B \\ 3,3V - 0,7V &= 170\mu A \times R_B \\ R_B &= 15.294\Omega = 15,294k\Omega \end{aligned} \tag{3.4}$$

Uma resistência de 15 kΩ foi conectada em série com a base do transistor, fazendo com que a corrente requerida da porta GPIO do microcontrolador seja de aproximadamente 173,333 μA. O esquemático do circuito transmissor infravermelho é ilustrado na Figura 3.12.

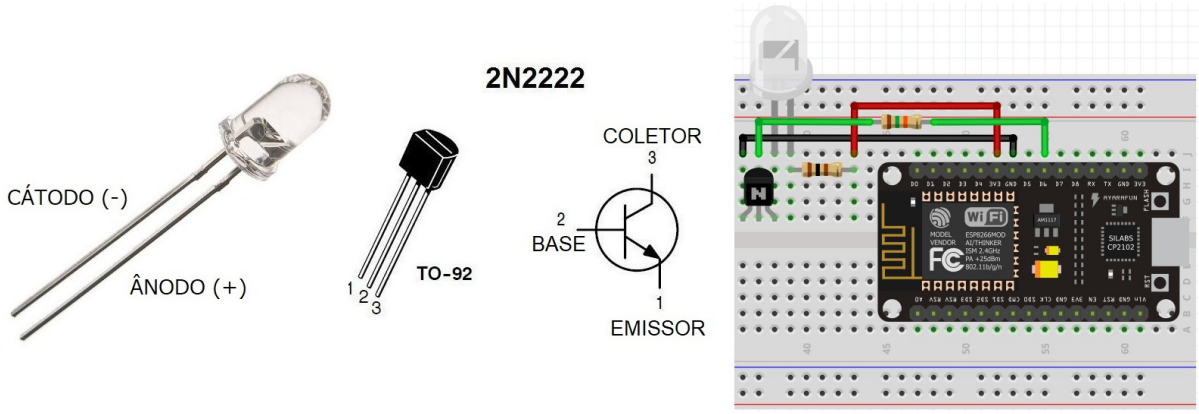

Figura 3.12: LED emissor IR, Transistor *2N2222 TO-92* e esquemático do circuito transmissor infravermelho (Fonte: [70]).

Após a construção do circuito transmissor IR, o componente *remote_transmitter* [71] da extensão *ESPHome* no servidor de automação deve ser configurado no arquivo YAML associado ao nó correspondente ao microcontrolador. O componente permite a transmissão de dados IR e RF para dispositivos que têm a capacidade de controle via infravermelho.

A condfiguração do *remote_transmitter* em si, ilustrada no Código 3.12, possui os seguintes parâmetros [71]: *pin* define o pino D6 como o transmissor dos sinais IR; *carrier_duty_percent*, estabelece a quantidade de tempo em que o sinal é ativo em um determinado período, sendo diretamente proporcional a frequência, o valor de 50% foi adotado para transmissão de sinais IR na frequência de modulação de 38 kHz.

```
remote_transmitter:
  pin: D6
  carrier_duty_percent: 50%
```

Código 3.12: Configuração do transmissor infravermelho.

Com a finalização da montagem e configuração dos circuitos de recepção e transmissão de sinais IR, um programa (ilustrado no Código 3.13) foi elaborado para testar o funcionamento destes sistemas. O receptor IR foi utilizado para captar o sinal IR transmitido pelo controle remoto de uma televisão da fabricante *Samsung* modelo *Q70T 2020* ao pressionar o botão de ligar do controle em questão. Ao receber e visualizar o sinal captado pelo receptor IR no *log* do microcontrolador, os dados decodificados do sinal são adicionados à uma ação "ligar" vinculada ao transmissor IR em uma entidade do tipo *switch* na interface visual do servidor de automação.

A entidade *switch* [72] da extensão *ESPHome* permite a criação de um interruptor de dois estados (ligado e desligado) na interface visual do servidor de automação, onde o mesmo pode ser controlado por voz devido a integração do servidor com o *Google Assistant*. O parâmetro *name* define o nome do interruptor para a interface visual e para o controle via assistente de voz. A programação de um *switch* com o objetivo de simular quaisquer botão de um controle remoto infravermelho requer a utilização de uma automação. Um *template* [73] permite a implementação de diversos tipos de automações em uma determinada entidade.

A automação implementada no programa de teste define a transmissão do sinal infravermelho $0xE0E06798$ no padrão utilizado pela fabricante *Samsung* na realização da ação "ligar" da entidade denominada "TV Samsung", que pode ser requerida via interface visual ou pelo assistente de voz. As opções disponibilizadas pelo componente *remote_transmitter* para transmissão de sinais IR são equivalentes às opções de visualização do *remote_receiver*: *transmit_lg*, *transmit_nec*, *transmit_panasonic*, *transmit_pioneer*, *transmit_jvc*, *transmit_samsung*, *transmit_sony*, *transmit_rc5* e *transmit_raw*.

```
1 switch:
2   - platform: template
3     name: "TV Samsung"
4     turn_on_action:
5       remote_transmitter.transmit_samsung:
6         data: 0xE0E06798
```

Código 3.13: Programação do transmissor infravermelho para ligar ou desligar um aparelho de televisão *Samsung* modelo *Q70T 2020*.

Dessa forma, o Código 3.13 pode ser utilizado como base para a programação de entidades que têm como objetivo substituir a função de um botão de um controle remoto qualquer. Os dados decodificados do sinal IR, obtidos por meio do receptor IR, são transmitidos de acordo com o padrão adotado pela fabricante do controle.

3.7 Sensor de temperatura

Com a conclusão da implementação das partes fundamentais de um controle remoto infravermelho controlado por voz, a adição de um sensor de temperatura para enriquecer as funcionalidades do protótipo foi realizada. O controle remoto universal IR pode ser utilizado para controlar um dispositivo climático como um ar condicionado e ao possuir uma forma de medição da temperatura ambiente, automações podem ser programadas no servidor de automação para o controle inteligente dessa temperatura.

Tabela 3.4: Valores de operação de um sensor de temperatura *LM35* básico (Fonte: [74]).

Tensão de operação	4 V a 20 V
Corrente de operação	$< 60\mu A$
Faixa de temperatura	$+2°$ C a $+150°$ C
Acurácia	$0,5°$ C

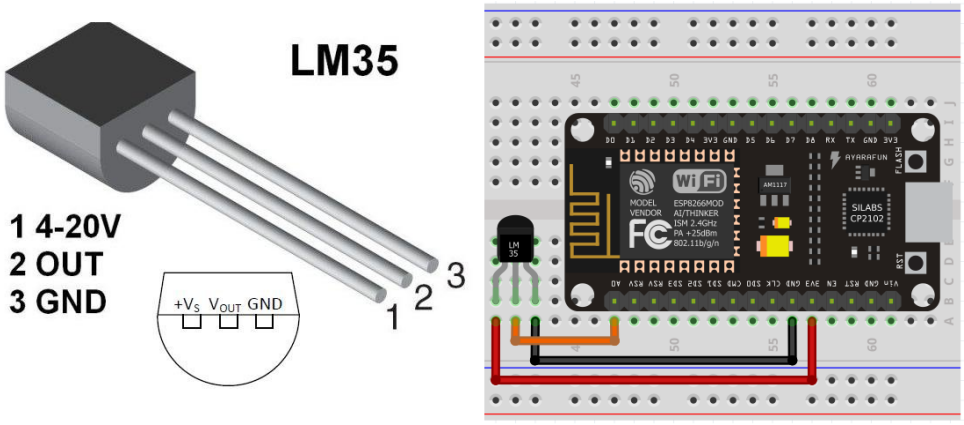

Figura 3.13: Sensor de temperatura *LM35DZ* e esquemático do circuito sensor de temperatura (Fonte: [75]).

O sensor de temperatura adotado foi da família *LM35* [74] com encapsulamento *TO-92* (assim como o transistor *2N2222*), denominado *LM35DZ* (ilustrado na Figura 3.13). A série de sensores de temperatura *LM35* é definida como "circuito integrado de precisão, cuja tensão de saída é linearmente proporcional à temperatura em graus Celsius". O fator de escala linear da tensão de saída em relação à temperatura é descrito pela equação:

$$V_{OUT} = 0mV + \frac{10mV}{1°C} \tag{3.5}$$

Para obter a temperatura medida pelo sensor a partir da tensão de saída, a utilização da interface ADC do microcontrolador é necessária. Como dito anteriormente, o *ESP8266*

possui apenas um pino que implementa essa interface, denominado como A0 ou GPIO17. O esquemático do circuito sensor de temperatura é ilustrado na Figura 3.13.

Para disponibilizar as informações medidas pelo sensor de temperatura na interface do servidor de automação, a criação de uma entidade do tipo *sensor adc* é realizada. A programação da entidade, ilustrada no Código 3.14, possui os argumentos [76]: *pin*, define o pino A0 como o alvo da medição de voltagem da entidade sensor tipo ADC; *name*, atribui um nome ao sensor na interface visual do servidor de automação; *update_interval*, intervalo de checagem do valor de voltagem no pino A0; *unit_of_measurement*, unidade de medida do sensor que nesse caso é em graus Celsius; *filters*, um filtro realiza uma transformação básica de valores, o filtro de tipo *lambda* é utilizado para calcular a temperatura a partir da tensão observada no pino A0.

O cálculo realizado na filtro *lambda* utiliza a Equação 3.5 da seguinte forma:

$$\begin{aligned} x = V_{OUT} &= 0mV + \frac{10mV}{1°C} \\ T = \frac{x \times 3,3V}{\frac{10mV}{1°C}} &= \frac{x \times 3,3}{0,01} \times 1°C \end{aligned} \tag{3.6}$$

O valor lido no pino A0 é obtido em Volts (x na Equação 3.6) e deve ser multiplicado pelo valor da tensão de alimentação do sensor ($3,3V$ na Equação 3.6) para obtenção do valor real da tensão no pino A0. Com esse valor real do pino A0, a temperatura é obtida por meio da Equação 3.5.

```
sensor:
  - platform: adc
    pin: A0
    name: "Temperatura Ambiente"
    update_interval: 60s
    unit_of_measurement: "Celsius"
    filters:
      - lambda: return (x * 3.3)/0.01;
```

Código 3.14: Programação do sensor de temperatura *LM35DZ*.

A programação da entidade sensor de temperatura permite o controle via interface visual do servidor de automação, porém não é possível obter os dados de temperatura do sensor via assistente de voz. Esse problema ocorre pois o *Google Assistant* suporta determinados tipos de dispositivos *smart home* [77], e apesar de um dos tipos ser descrito como *SENSOR*, a entidade *sensor adc* no servidor de automação não se encaixa nesse tipo de dispositivo *smart home*. A solução para esse problema é a criação de uma entidade

no servidor de automação que corresponde ao tipo de dispositivo *THERMOSTAT* no assistente de voz.

Uma entidade *climate* [78] pode representar diversos tipos de *hardware*, pois a mesma é definida pela capacidade de configuração de uma temperatura alvo e diferentes modos de operação como quente, frio, automático e desligado. O *Google Assistant* identifica entidades *climate* como dispositivos *smart home* do tipo *THERMOSTAT*. Para habilitar requisições do usuário sobre a temperatura ambiente medida pelo sensor de temperatura via assistente de voz, a integração do *Home Assistant* MQTT *Heating, Ventilating and Air Conditioning* (HVAC) foi adotada na entidade *climate* criada no arquivo `configuration.yaml` e ilustrada no Código 3.15.

```
climate:
  - platform: mqtt
    name: "Temperatura do Termostato"
    qos: 1
    send_if_off: true
    availability_topic: "nodemcu/status"
    current_temperature_topic: "nodemcu/sensor/temperatura_ambiente/state"
    temperature_state_topic: "nodemcu/sensor/temperatura_ambiente/state"
    mode_state_topic: "void/hvac/mode/state"
    modes:
      - "heat"
```

Código 3.15: Criação de um termostato MQTT HVAC a partir dos dados medidos pelo sensor de temperatura *LM35DZ*.

Os parâmetros utilizados na configuração do termostato são [79]: *name*, nome da entidade, utilizado para identificar o termostato na interface visual e na interação com o assistente de voz; *qos*, nível de QoS na comunicação MQTT, definido como 1 ao invés do nível padrão 0; *send_if_off*, define se a entidade continua enviando mensagens mesmo se o modo do termostato estiver no modo "desligado", é um parâmetro obrigatório que não tem relevância nessa implementação pois a entidade não possui este modo; *availability_topic*, tópico MQTT para verificar a disponibilidade de comunicação com o sensor, os parâmetros *payload_available* (valor padrão definido como *online*) e *payload_not_available* (valor padrão definido como *offline*) definem os valores de verificação de disponibilidade a partir do conteúdo da mensagem publicada pelo tópico; *current_temperature_topic*, tópico MQTT para obtenção do valor da temperatura atual reportada pelo sensor de temperatura; *temperature_state_topic*, tópico MQTT para verificação de mudanças no

valor da temperatura alvo do termostato, também definido como o valor de temperatura reportado pelo sensor; *mode_state_topic*, tópico MQTT para verificação de mudanças no modo de operação do termostato; *modes*, lista de modos de operação disponíveis para o termostato, os modos disponíveis são *auto*, *off*, *cool*, *heat*, *dry* e *fan_only*.

Com a finalização da criação do termostato, requisições do valor de temperatura ambiente reportado pelo sensor via assistente de voz são possíveis. Ao requerer a temperatura do termostato via *Google Assistant* por meio da invocação "Ok *Google*, qual a Temperatura do Termostato?", um exemplo de resposta obtida é "Temperatura do Termostato está no modo desligado e está 29 graus.". Pode-se perceber que o assistente de voz informa além da temperatura, o modo de operação do termostato. O objetivo do termostato criado é apenas reportar o valor da temperatura, portanto o modo de operação não é relevante e não deve ser incluído na reposta do assistente. Para obter uma resposta semelhante a "No momento está 27,5 graus.", o termostato deve ser configurado no modo *heat*.

Para configurar o modo de operação do termostato para o valor desejado, uma mensagem contendo esse valor no *payload* deve ser publicada no tópico MQTT *mode_state_topic*. Para fazer essa publicação de uma forma automatizada, uma automação foi criada no *Home Assistant*. Essa automação realiza um *publish* MQTT quando o servidor de automação é iniciado, o conteúdo da mensagem publicada é ilustrado no Código 3.16.

```
1 topic: void/hvac/mode/state
2 payload: heat
```

Código 3.16: Conteúdo do pacote publicado no tópico MQTT para verificação do modo de operação do termostato quando o servidor de automação é iniciado.

Com a criação da automação e a reinicialização do servidor de automação, requisições sobre o valor de temperatura do sensor *LM35DZ* são respondidas sem a informação do modo de operação do termostato, como por exemplo "No momento está 28,5 graus.". A utilização de um sensor de temperatura no projeto pode ter diversas utilidades além de informar a temperatura atual, automações podem ser criadas para controlar quaisquer dispositivo HVAC de acordo com a temperatura por meio de um processo semelhante ao descrito na seção anterior, onde o receptor IR e o transmissor IR substituem a necessidade de utilização do controle remoto proprietário do dispositivo.

3.8 Protótipo do controle universal infravermelho

Os esquemáticos dos circuitos receptor IR (Figura 3.11), transmissor IR (Figura 3.12) e sensor de temperatura (Figura 3.13) compõem o esquemático do circuito do protótipo do controle universal IR, ilustrado na Figura 3.14.

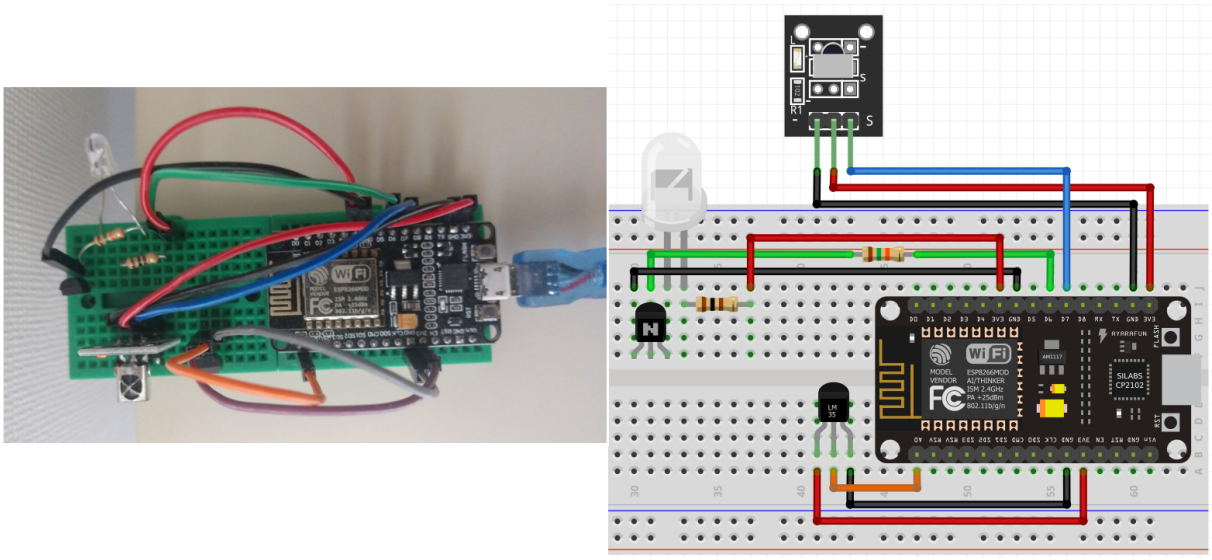

Figura 3.14: Protótipo do controle remoto universal IR e esquemático do circuito do protótipo.

O protótipo do controle universal, também ilustrado Figura 3.14, tem como objetivo validar o funcionamento de todos os componentes explorados nas seções anteriores. A programação do microcontrolador é ilustrada no código disponibilizado no Anexo I e essa validação é realizada por meio de requisições por voz ao *Google Assistant* relacionadas às funcionalidades de controle remoto implementadas no protótipo. Os testes de validação realizados serão detalhados no próximo capítulo.

A relevância do problema abordado no Capítulo 1 é verificada pela quantidade de controles universais infravermelhos inteligentes disponíveis no mercado, citando apenas alguns exemplos: o Smart Controle Universal da Positivo Casa Inteligente[8]; e o Controle Universal KaBuM! Smart da KaBuM! Smart[9]. Comparando esses produtos com o protótipo desenvolvido, se faz possível perceber que ambos necessitam de um aplicativo próprio e conexão Wi-Fi para configuração, assim como o protótipo é configurado via Wi-Fi por meio do servidor de automação. As maiores vantagens dos produtos são a compatibilidade tanto com o *Google Assistant* quanto com a *Amazon Alexa*, assim como uma distância de transmissão muito maior, de 8 m, do emissor infravermelho. O diferencial do protótipo DIY, que pode ser uma vantagem ou desvantagem, é a customização, que também é oferecida pelos produtos citados, mas não ao nível de um servidor de automação próprio. Além disso, a construção do protótipo é menos custosa financeiramente, mas exige muito mais tempo para implementar as funções já disponíveis nos produtos comercializados. O

[8] `https://www.positivocasainteligente.com.br/smart-controle-universal/p`

[9] `https://www.kabum.com.br/cgi-local/site/produtos/descricao_ofertas.cgi?codigo=107960`

protótipo também é sujeito a problemas como *bugs* que devem ser resolvidos pelo usuário, enquanto os produtos possuem suporte oferecido pelas fabricantes.

Capítulo 4

Testes

Este capítulo descreve os testes realizados para validação das funcionalidades idealizadas para o controle remoto universal IR projetado. Os testes foram elaborados para evidenciar a solução adotada para o problema exposto no Capítulo 1 e aplicar métricas para comparação de diferentes abordagens no contexto do trabalho. O código utilizado nos testes é ilustrado no Anexo I.

4.1 Controle por voz do LED embutido no *NodeMCU*

Para testar o controle do LED, programado com uma entidade *light* nomeada como "Onboard LED", as frases de invocação utilizadas no auto-falante inteligente *Google Home Mini* e no aplicativo *Google Assistant* de um *smartphone Android* foram:

- "Ok *Google*, ligar Onboard LED."
- "Ok *Google*, desligar Onboard LED."

Como as informações de estado dos dispositivos são efêmeras, a reposta do assistente de voz nesse caso não considera se o LED já se encontra ligado e desligado. Isso faz com que a ação de ligar ou desligar seja executada novamente a cada invocação. O tempo médio mensurado entre a invocação e a execução da ação nesse caso é de 7 a 8 segundos.

No caso em que a invocação é requerida via *Google Home Mini*, um sinal sonoro é emitido quando a ação é realizada. Se a invocação for realizada via aplicativo, o assistente de voz responde com a frase "Tudo bem, ligando Onboard LED" ao realizar a ação. Na situação em que o assistente de voz não consiga se comunicar com o LED e como consequência não realiza a ação requerida, a frase de resposta é "Desculpe, parece que Onboard LED não esta disponível no momento."

Em alguns casos o assistente de voz responde a invocação com uma pergunta de confirmação, como por exemplo "Você quis dizer desligar Onboard LED?", fazendo com que

a execução da ação dependa da resposta dessa pergunta. No caso em que a resposta dessa pergunta é afirmativa e realizada de forma imediata, o tempo médio entre a invocação e a execução da ação nesse caso é de 14 a 15 segundos.

Para comparar a comunicação via protocolo MQTT com a API nativa, dois botões representando o LED embutido são adicionados na interface visual do servidor de automação. Um botão realiza a comunicação via MQTT e o outro utiliza a API nativa da extensão *ESPHome*. Os testes evidenciam que não há diferença perceptível no tempo de execução da ação entre os protocolos, ambos realizam a ação em um tempo de aproximadamente 1 segundo ao clicar no botão da interface no servidor.

4.2 Controle por voz de aparelhos de televisão

Com o objetivo de testar o conceito da principal funcionalidade idealizada para o controle remoto universal IR, o gerenciamento de diversos dispositivos com controles remotos próprios por meio de comandos de voz, as seguintes frases de invocação foram utilizadas no *Google Assistant* para controlar um aparelho de televisão *Samsung* modelo *Q70T 2020*:

- "Ok *Google*, ligar TV Samsung"
- "Ok *Google*, desligar TV Samsung"

O tempo médio entre a invocação e execução da ação é de 8 a 9 segundos e as frases de resposta do assistente são similares à "Claro, ligando a TV Samsung." e "Sem problemas, desligando a TV Samsung.". Assim como no caso do LED embutido na placa de desenvolvimento, informações sobre o estado do aparelho de televisão são efêmeras, fazendo com que as invocações executem as ações sem considerar se o aparelho se encontra ligado ou desligado. O código emitido pelo transmissor infravermelho (obtido por meio do receptor infravermelho a partir do controle proprietário da televisão) é o mesmo para as ações de ligar e desligar, isso faz com que repetir o comando 'Ok *Google*, ligar TV Samsung" duas vezes desligue a televisão e vice-versa.

No primeiro teste realizado, o microcontrolador foi alimentado por uma porta USB a uma distância de aproximadamente 45 cm e angulação de 0° do receptor IR do televisor. Para melhorar a mobilidade do controle projetado com o objetivo de medir a distância máxima de funcionamento do transmissor infravermelho, pilhas alcalinas de tipo AA foram utilizadas para alimentar o microcontrolador.

Para o segundo teste, o microcontrolador foi alimentado pelo pino Vin e GND com duas pilhas alcalinas do tipo AA de tensão de $1,5$ V cada, fornecendo então 3 V para o protótipo. Nesse teste foi possível mensurar a distância máxima de funcionamento do controle universal, sendo essa distância por volta de 2 m, duas vezes maior que a distância

de transmissão definida na Tabela 3.3. A angulação entre o controle universal e o receptor da televisão também foi de 0° na realização desse teste.

Com o objetivo de comparar a tensão de alimentação fornecida pela porta USB com a tensão fornecida por pilhas AA, assim como verificar o valor de voltagem fornecido pelas pilhas conectadas ao pino Vin, uma entidade *sensor* de tipo ADC foi programada no código do MCU.

A entidade, ilustrada no Código 4.1, realiza a medição da tensão que o microcontrolador está recebendo por meio de um pino ADC. Como o microcontrolador possui apenas um pino que implementa a interface ADC, o sensor de temperatura foi temporariamente desconectado do pino A0 durante os testes de medição de tensão [76].

```
1 sensor:
2   - platform: adc
3     pin: VCC
4     name: "Voltagem no MCU"
```

Código 4.1: Programação do sensor de medição da voltagem no microcontrolador.

Tabela 4.1: Voltagem medida no microcontrolador de acordo com a fonte de alimentação.

Fonte de alimentação	**Voltagem no MCU**
Porta USB	2,95 V
2 Pilhas AA	1,73 V
3 Pilhas AA	2,95 V

Os valores de voltagem obtidos se encontram na Tabela 4.1. Quando o protótipo é alimentado via porta USB, a tensão medida no MCU se mantém em um valor constante de 2,95 V. Esse é o caso pois a trilha da tensão fornecida pela porta USB passa pelo regulador de tensão presente na placa de desenvolvimento (*AMS1117 3.3*), fazendo com que a tensão de 5 V fornecida pela porta seja regulada para o valor de 3,3 V.

No caso em que o controle universal é alimentado por duas pilhas AA, a tensão mensurada é de apenas 1,73 V. Além disso, foi observado uma queda na tensão para 1,68 V durante o período dos testes de verificação da distância máxima de transmissão do emissor IR. Esse valor baixo confirma que a trilha do pino Vin também passa pelo regulador de tensão, e como as pilhas fornecem apenas 3 V, o *AMS1117 3.3* não consegue regular a tensão para o valor de 3,3 V. Dessa forma, duas pilhas AA não são suficientes para alimentar o protótipo do controle remoto universal IR.

Alimentando o protótipo por meio de três pilhas AA que fornecem uma tensão de alimentação total de 4,5 V, a voltagem medida no microcontrolador é a mesma do caso em que o controle universal é alimentado via USB, possuindo valor de 2,95 V. Em um caso que o controle universal gerencie pelo menos mais de um dispositivo, onde cada aparelho

utilize um controle remoto próprio que necessite de duas pilhas AA para funcionar, a utilização do protótipo já é teoricamente mais vantajosa. Na prática, a durabilidade das pilhas utilizadas também deve ser considerada para verificar que a utilização do controle universal realmente é melhor no aspecto de consumo das pilhas.

Um importante aspecto a ser considerado os testes dos circuitos transmissor e receptor IR é a implementação da interface *IR Remote Control* por parte do *ESP8266*. Como foi explicitado anteriormente, O transmissor infravermelho é implementado no pino MTMS (D5 no *NodeMCU*) e o receptor no pino GPIO5 (D1 no *NodeMCU*). Os pinos programados para a comunicação por infravermelho no trabalho não foram os pinos que implementam a interface *IR Remote Control*, e o funcionamento dos circuitos de transmissão e recepção observado nos testes foi o desejado.

Para fins de comparação, alguns testes foram realizados com os circuitos conectados aos pinos de interface *IR Remote Control* e nenhuma grande diferença no funcionamento do protótipo foi percebida. Esse é o caso pois a geração do sinal IR é implementada via *software* para todos os pinos, a biblioteca de comunicação IR utilizada pela extensão *ESPHome* não usufrui da capacidade de geração de sinais modulados via *hardware* pelo pino MTMS [80]. Para utilizar essa geração de sinal por *hardware*, uma implementação da função própria de geração de sinais IR deve ser realizada.

Com o objetivo de testar o funcionamento do controle universal em mais aparelhos de diferentes fabricantes que implementam seus próprios protocolos de comunicação, os sinais IR emitidos pelos controles de uma televisão *LG* e de um aparelho de TV a cabo da operadora *NET* em determinadas operações foram capturados e se encontram ilustrados na Tabela 4.2.

Tabela 4.2: Códigos IR de protocolos *LG* e *NEC* (utilizado no aparelho de TV a cabo *NET*) com suas respectivas operações correspondentes.

Operação	**TV *LG* com controle AKB72914210**	**TV a cabo *NET* modelo DXC5000KNB**
Ligar	data=0x20DF10EF, nbits=32	address=0xE17A, command=0x48B7
Desligar	data=0x20DF10EF, nbits=32	address=0xE17A, command=0x48B7
Volume +	data=0x20DF40BF, nbits=32	address=0xE17A, command=0xB04F
Volume -	data=0x20DFC03F, nbits=32	address=0xE17A, command=0x708F
Silenciar	data=0x20DF906F, nbits=32	não utilizado
Entrada	data=0x20DFD02F, nbits=32	não utilizado
Canal +	não utilizado	address=0xE17A, command=0x08F7
Canal -	não utilizado	address=0xE17A, command=0x58A7

Utilizando como base a entidade *switch* explorada anteriormente, utilizada para ligar e desligar uma TV *Samsung*, uma nova entidade *switch* foi criada para cada operação desejada nos respectivos aparelhos. Ao realizar os testes das novas funções programadas,

foi percebido que para ativar e desativar uma entidade do tipo *switch*, as frases de invocação que devem ser utilizadas no assistente de voz são "Ok *Google*, ligar entidade" e "Ok *Google*, desligar entidade". Essa frase se adequa apropriadamente para a ação de ligar e desligar um aparelho, mas para aumentar ou diminuir o volume do mesmo, a frase não fica intuitiva para o usuário, como por exemplo "Ok *Google*, ligar Volume + TV LG" e "Ok *Google*, ligar Volume - TV LG". Isso acontece pois o *Google Assistant* reconhece as entidades programadas como sendo do tipo de dispositivo *smart home SWITCH*, o que é correto, já que as entidades foram criadas como sendo do tipo *switch* no código de programação do microcontrolador.

Para resolver essa inconveniência, uma nova integração foi utilizada no servidor de automação, a *Universal Media Player*. Essa integração "combina diversas entidades existentes no *Home Assistant* em uma entidade reprodutora de mídia (*media_player*)." [81]. No contexto do projeto, as entidades *media_player* programadas (ilustrada no Anexo II) têm como objetivo melhorar as frases de invocação utilizadas para realizar certas ações. Por exemplo, ao invés de invocar a frase "Ok *Google*, ligar Volume + TV LG" para realizar a ação, a frase mais intuitiva "Ok *Google*, aumentar volume da TV LG" pode ser invocada. As frase de invocação da TV *LG* que foram melhoradas por meio dessa entidade são:

- "Ok *Google*, ligar TV LG"
- "Ok *Google*, desligar TV LG"
- "Ok *Google*, aumentar volume TV LG"
- "Ok *Google*, diminuir volume TV LG"
- "Ok *Google*, silenciar TV LG"

Os serviços da entidade *media_player* utilizados nas frases citadas foram: *turn_on*, *turn_off*, *volume_up*, *volume_down volume_mute* [81]. Vale ressaltar que os comandos relacionados ao volume do aparelho, o acréscimo e decréscimo no valor atual de volume é feito por 9 unidades. O tempo médio entre a invocação e a execução da tarefa também foi por volta de 8 a 9 segundos. As frases de resposta do assistente são semelhantes a "Claro" e "Sem problemas" quando o assistente é utilizado via *smartphone*. No caso em que o auto-falante inteligente *Google Home Mini* é utilizado, a tarefa é executada e apenas um sinal visual é emitido pelos LEDs presentes no dispositivo.

O serviço *select_source*, que seria utilizado para mudar a entrada de exibição do aparelho de televisão também foi programado inicialmente, mas a sua frase de invocação "Ok *Google*, mudar entrada TV LG", retornou a seguinte resposta pelo assistente de voz "A TV LG não é compatível com essa funcionalidade.". Essa resposta indicou que o

Google Assistant não estava considerando a televisão como sendo do tipo de dispositivo *smart home TV*. Ao verificar o tipo reconhecido pelo assistente de voz no aplicativo *Google Home*, percebeu-se que a televisão foi identificada como sendo um dispositivo *smart home SETTOP*, que pode ser um conversor, decodificador ou receptor de televisão que são fornecidos por operadoras como a *NET*. Apesar desse tipo de dispositivo possuir a capacidade de mudança de entradas, durante os testes realizados não foi possível realizar essa ação por comandos de voz.

Para não perder a funcionalidade de troca da entrada de exibição da televisão, a entidade do tipo *switch* programada para realizar essa ação foi mantida. Durante os testes, uma frase de invocação mais intuitiva foi percebida:

- "Ok *Google*, acionar entrada TV LG"

Essa invocação controla a entidade *switch* como um botão qualquer, dando um sentido diferente da invocação anterior, que utilizava de forma explicita as palavras ligar e desligar na frase. Apesar da mudança na semântica na frase de invocação, o tipo de dispositivo *smart home* continua o mesmo, portanto a reposta do assistente de voz pode ser "Claro, ligando entrada TV LG".

No caso do aparelho de TV a cabo, as seguintes frases de invocação também foram melhoradas:

- "Ok *Google*, ligar NET"
- "Ok *Google*, desligar NET"
- "Ok *Google*, aumentar volume NET"
- "Ok *Google*, diminuir volume NET"

Os serviços utilizados, o tempo médio para execução da ação e as respostas do assistente de voz são os mesmos que os já citados anteriormente para a entidade *media_player* da TV *LG*. O acréscimo e decréscimo no valor atual de volume também é feito por 9 unidades.

O tipo de dispositivo *smart home* reconhecido pelo assistente de voz nesse caso também foi o *SETTOP*, que possui as funcionalidades de avançar e retroceder mídias. Ao invés de tentar utilizar o serviço *select_source* nessa entidade, os serviços *media_previous_track* e *media_next_track* foram programados e testados. Da mesma forma que não foi possível utilizar a frase de invocação específica do serviço *select_source*, as frases dos serviços *media_previous_track* e *media_next_track* também obtiveram respostas apontando que o dispositivo não é compatível com as respectivas funcionalidades.

O objetivo de programar esses serviços era a habilitação de frases de invocação para trocar o canal exibido pelo aparelho de TV a cabo, como por exemplo "Ok *Google*, próximo

NET" ou "Ok *Google*, anterior NET". Para não perder essas funcionalidades, o mesmo processo que foi realizado no caso da operação de entrada da TV *LG* foi repetido para a troca de canais na TV a cabo. As frases de invocação testadas para a realização dessas tarefas foram:

- "Ok *Google*, acionar próximo canal NET"
- "Ok *Google*, acionar canal anterior NET"

Assim como foi explicitado anteriormente, a resposta do *Google Assistant* para uma dessas frases de invocação pode ser 'Tudo bem, ligando próximo canal NET", o que não é ideal, mas foi a solução adotada para o problema de incompatibilidade de tipos entre o servidor de automação e o assistente de voz.

Apesar do controle via *Google Assistant* não possuir as respostas mais adequadas, o controle por voz de quaisquer aparelho que tenha um controle remoto IR com as funções mapeadas em entidades de tipo *switch* no servidor de automação é viável tanto pelo *Google Home Mini* quanto por um *smartphone* com o *Google Assistant* instalado.

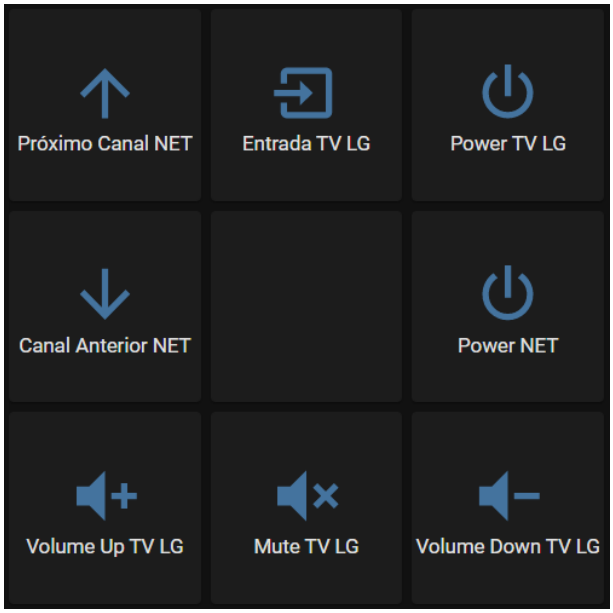

Figura 4.1: Exemplo de um controle centralizado (TV *LG* e TV a cabo *NET*) criado na interface visual do servidor de automação.

Levando em consideração o controle dos dispositivos sem a utilização da voz, podendo ser realizado por meio do aplicativo *Google Home*, pela interface visual do servidor de automação e pelo aplicativo do *Home Assistant*, o papel de centralização dos diversos controles de um ambiente em uma casa por meio do controle universal projetado pode ser facilmente percebido.

A Figura 4.1 exemplifica a centralização de dois controles remotos proprietários na interface visual do *Home Assistant*. Considerando o ambiente em questão, que possui uma TV e um aparelho de TV a cabo, se o mesmo ambiente possuir mais dispositivos que se comunicam via IR, as funções dos controles remotos dos dispositivos podem ser adicionadas ao controle centralizado ilustrado. Ao invés de possuir diversos controles com dezenas de botões, apenas um controle, representado pela interface visual do servidor de automação, com os botões que representam as funcionalidades mais utilizadas pode ser programado e adotado como o controle remoto universal infravermelho do ambiente.

4.3 Requisição da temperatura ambiente por voz

As frases de invocação utilizadas nos testes de obtenção da temperatura medida pelo sensor *LM35DZ* via assistente de voz foram:

- "Ok *Google*, qual a Temperatura do Termostato?"
- "Ok *Google*, qual a Temperatura do Ambiente?"

Vale ressaltar que a invocação deve conter o nome dado ao termostato MQTT HVAC ("Temperatura do Termostato", configurado na interface visual do servidor de automação) para obtenção dos dados do sensor de temperatura. Inicialmente, o nome do termostato era "Temperatura do Ambiente", mas após a realização dos testes, uma variação na resposta da invocação "Ok *Google*, qual a Temperatura do Ambiente?" foi percebida. Para evitar essa ambiguidade na resposta, o nome do termostato foi alterado para "Temperatura do Termostato".

Ao repetir a invocação "Ok *Google*, qual a Temperatura do Ambiente?" durante os testes, a resposta do assistente de voz reportou tanto a temperatura do termostato, quanto dados sobre a temperatura ambiente obtidos via *internet*. Alguns exemplos dessas respostas do assistente de voz que variam de acordo com o contexto: "No momento, em Brasília, faz 73 graus.", temperatura em Fahrenheit obtida via *internet*; "No momento está 26 graus.", temperatura em graus Celsius medida pelo sensor de temperatura do controle universal.

A resposta do assistente de voz é a mesma para invocações de temperatura do termostato realizadas no *Google Home Mini* e no aplicativo do *Google Assistant*. O tempo médio entre a invocação da ação e a resposta do assistente de voz é de 9 a 10 segundos. Invocações semelhantes à "Ok *Google*, qual a temperatura?" e "Ok *Google*, qual a temperatura atual?" são respondidas pelo assistente com informações de temperatura baseadas em localização obtidas +via *internet* como "No momento, a temperatura em Brasília é de 73 graus." e "A temperatura atual em Brasília é de 73 graus.", respectivamente.

4.4 Observações sobre o servidor de automação

Durante a realização dos testes apresentados neste capítulo, certas características sobre o funcionamento do servidor de automação foram observadas e serão explicitadas a seguir.

Inicialmente, a configuração de um IP estático não foi necessária e o acesso ao servidor era realizado via `http://homeassistant.local:8123`. Após alguns casos de instabilidade de conexão com o meu ISP, problemas com o DHCP tornaram o acesso ao servidor inviável por meio do endereço URL padrão. Sendo assim, a configuração de um IP estático para o servidor de automação foi necessária (explicitada no Capítulo 3). Um cenário semelhante ocorreu com a configuração de rede da placa de desenvolvimento, inicialmente um IP era atribuído automaticamente pelo DHCP, mas devido à instabilidade desse método, a configuração de um IP estático foi realizada no microcontrolador (explicitada no Capítulo 3).

Ao realizar a invocação para atualizar as entidades programadas no microcontrolador "Ok *Google*, sincronizar dispositivos", a reposta do assistente de voz é "Claro, sincronizando os dispositivos para Home Assistant.", onde *Home Assistant* nesse caso é o nome da ação do projeto criado na plataforma de desenvolvimento *Actions on Google*. Essa sincronização deve ser realizada sempre que o código de programação do microcontrolador for alterado. Se novas entidades forem adicionadas em uma nova versão do código, a invocação permite que as mesmas sejam controladas via *Google Assistant*, supondo que os tipos das entidades estão expostos na configuração de integração do assistente de voz com o servidor de automação.

Em casos em que o servidor de automação perca conexão à internet, o assistente de voz responde às invocações realizadas com a seguinte frase "Desculpe, parece que Home Assistant não está disponível nesse momento.". Para checar a conexão do servidor, o comando ilustrado no Código 4.2 é utilizado na linha de comando da VM do servidor. A resposta do comando possui diversos parâmetros que contém informações sobre a conexão. O parâmetro que define se o acesso ao servidor está disponível é o *supervisor_internet*, ilustrado no Código 4.2. Quando o parâmetro possui o valor *false*, o acesso ao servidor está indisponível e vice-versa. Para reconectar o servidor à rede, o desligamento e inicialização da VM onde o mesmo se encontra devem ser efetuados.

```
1 ha > network info
2 ...
3 supervisor_internet: true
```

Código 4.2: Verificação de conexão à internet no servidor de automação.

Reiniciando o servidor via desligamento e inicialização da máquina virtual, um novo túnel *ngrok* é estabelecido. O novo túnel possui um endereço URL diferente do configurado

na integração com o assistente de voz, sendo assim, o novo endereço deve ser atualizado nos campos de endereços URL *fullfilment*, *authorization* e *token* do projeto criado na plataforma *Actions on Google*. Realizei esse processo de atualização de endereço URL toda vez que a máquina virtual do servidor de automação foi desligada e iniciada novamente. A reinicialização do servidor via interface visual não requer a realização desse processo de atualização do endereço URL externo obtido via túnel *ngrok*.

Durante a realização da bateria de testes que iriam ser gravados em vídeo para ilustrar o funcionamento do protótipo durante a apresentação do trabalho, alguns problemas foram encontrados. O problema principal foi a ocorrência de instabilidades no túnel *ngrok*.

Inicialmente pensei que a causa do empecilho estivesse relacionada à extensão *ngrok Client* utilizada no servidor de automação, e após algumas tentativas de consertar o problema, acabei desvinculando o projeto da plataforma de desenvolvimento *Actions on Google* do aplicativo *Google Home*. Após diversas investidas sem sucesso de revinculação do projeto no aplicativo, tomei a decisão de configurar outro projeto e vinculá-lo ao *Google Home*. A habilitação de testes no novo projeto configurado não estava sendo possível por algum problema na plataforma de desenvolvimento. Depois de um dia, tentei novamente vincular o novo projeto e dessa vez fui bem-sucedido, resolvendo o problema. Esse *bug* é citado na seção de integração com o *Google Assistant* no site oficial do *Home Assistant* [63], mas a solução sugerida no site não funcionou nesse caso.

As instabilidades no túnel *ngrok* continuaram, e desinstalando a extensão do servidor de automação e tentando instalá-la novamente, percebi que não era possível reinstalar a extensão por um erro de rede. Verificando o acesso à internet do servidor de automação por meio do comando *ping* ao site do *Google* no terminal, percebi que o DNS do *gateway* configurado não estava permitindo o acesso do servidor à internet. O comando ilustrado no Código 4.2 apontava que o *supervisor* estava conseguindo acesso à rede, porém não era possível instalar novas extensões e nem obter respostas via *ping*. Para resolver essa questão, configurei um novo DNS, utilizando o DNS público do *Google* (IP 8.8.8.8). Após essa configuração, o acesso à internet por parte do servidor de automação foi normalizado.

Com a resolução dos problemas citados acima, foi possível reinstalar a extensão *ngrok Client* e realizar os testes ilustrativos com sucesso, porém não foi possível determinar a causa dessa instabilidade isolada do túnel *ngrok*. Suspeito que o problema seja na própria implementação do túnel por parte do *ngrok* e não na extensão do servidor de automação, pois pesquisando sobre percebi que esse problema ocorre em diversos contextos de utilização do túnel. De qualquer forma, reportei o problema encontrado no *GitHub*[1] da extensão com o objetivo de investigar a causa desse problema de instabilidade e possivelmente contribuir para a solução do mesmo.

[1] `https://github.com/dylrio/hassio-ngrok`

Capítulo 5

Conclusão

No Capítulo 1, o problema da utilização de diversos controles remotos foi abordado, exemplificando situações em que o gerenciamento de aparelhos se torna dispendiosa, apresentando a ideia de unificação do controle por meio de um controle remoto universal IR e definindo os objetivos que deveriam ser cumpridos no trabalho. O Capítulo 2 evidenciou os conceitos e ferramentas utilizados no planejamento e execução do trabalho. O processo de desenvolvimento do trabalho foi exposto no Capítulo 3, apresentando a arquitetura do trabalho e descrevendo os processos atrelados a cada componente utilizado na implementação da arquitetura idealizada. No Capítulo 4, os testes utilizados para validar a implementação da arquitetura foram descritos e alguns aspectos observados sobre a centralização de controles remotos e sobre o comportamento do servidor de automação implementado foram ressaltados.

Retomando os objetivos estabelecidos no Capítulo 1:

1. configuração das ferramentas disponibilizadas pelo *Google Assistant*, empregando o *Google Home Mini* e um aparelho celular como receptores dos comandos de voz do controle universal;
2. implementação de um servidor para o estabelecimento da comunicação entre o *Google Assistant* e o controle universal utilizando o protocolo MQTT ou semelhante para troca de mensagens entre o servidor e o controle;
3. construção de um protótipo de controle universal por comando de voz a partir do *NodeMCU* com capacidade de receber e transmitir sinais infravermelhos, explorando os diferentes protocolos de comunicação infravermelha das diversas marcas de aparelhos eletrônicos que possuem controles remotos com essa tecnologia.

Com relação ao primeiro objetivo, as ferramentas do *Google Assistant* configuradas foram: a plataforma de desenvolvimento *Actions on Google*, utilizado para criação de um

projeto de integração do assistente de voz com o servidor de automação; o aplicativo *Google Home*, utilizado para vinculação do projeto criado com o assistente de voz, permitindo o controle dos dispositivos presentes no servidor de automação. Tanto o auto-falante inteligente *Google Home Mini* quanto um *smartphone Android* com o assistente de voz instalado foram utilizados como receptores de comandos para o controle universal.

O segundo objetivo foi cumprido com a criação de uma máquina virtual com sistema operacional do *Home Assistant*. O servidor presente na VM foi utilizado para programação do microcontrolador do controle via extensão *ESPHome*, realizou a comunicação com o controle universal via protocolo MQTT e API nativa *ESPHome*, e possibilitou o controle por voz via assistente de voz por meio da configuração de integração do servidor com o *Google Assistant*.

A construção de um protótipo do controle universal foi realizada, onde o protótipo possui três principais componentes, dois utilizados para a comunicação via infravermelho e um para aferição da temperatura no ambiente. Os testes realizados verificaram o funcionamento da arquitetura para recebimento de sinais IR de quaisquer controle remoto, assim como para a transmissão desses sinais, fazendo com que o protótipo centralize as funções de diversos controles remotos em sua programação. Com a realização da integração do servidor de automação com o assistente de voz, o protótipo pode ser controlado pela voz via *Google Assistant*. Os protocolos explorados nesse documento foram os utilizados pelas fabricantes *Samsung*, *LG* e pela operadora de TV a cabo *NET* em seu aparelho. Os pontos citados verificam o cumprimento do terceiro e último objetivo específico do trabalho, com a adição da implementação de um sensor de temperatura além da comunicação via IR.

5.1 Trabalhos futuros

Visualizando a arquitetura do trabalho, é possível notar que existem diversos pontos de bifurcação nas decisões tomadas no processo de implementação. As decisões tomadas, assim como as alternativas que poderiam ter sido implementadas serão discutidas a seguir. Essas alternativas são consideradas como trabalhos futuros na implementação do projeto.

O primeiro ponto é a utilização do assistente de voz da *Google*. Existem vários assistentes pessoais que poderiam ter sido adotados no trabalho ao invés do escolhido. Tomando como exemplo os produtos citados anteriormente, um trabalho futuro para o projeto seria o suporte ao assistente de voz da *Amazon*. A adoção de outros assistentes pessoais no trabalho, como *Microsoft Cortana* e *Apple Siri*, seria um diferencial em relação aos produtos disponíveis no mercado, mas o esforço de implementação deve ser bem maior se comparado ao *Google Assistant* e *Amazon Alexa*.

A entidade responsável pela implementação da comunicação entre o assistente de voz e o controle prototipado também é outro ponto de bifurcação. Outras plataformas de automação residencial bastante conhecidas e utilizadas como o *openHAB* e *Domoticz* poderiam ter sido escolhidas para realizar o papel do servidor de automação na arquitetura do trabalho.

O principal componente da arquitetura, o *NodeMCU V2*, também poderia ser outro modelo, ou até outra placa de desenvolvimento com outro microcontrolador, o que provavelmente mudaria de forma drástica a implementação realizada no projeto. Alguns exemplos de componentes que poderiam substituir o *NodeMCU V2* são *ESP8266 ESP-01* em conjunto com um *Arduino Uno*, *Wemos D1 Mini*, *Arduino Nano 33 IoT* e *Arduino MKR1000*.

Além dos trabalhos futuros relacionados às escolhas tomadas na implementação do trabalho, um aspecto presente no servidor de automação *Home Asssitant* também poderia ser incluído posteriormente no projeto. Esse aspecto é relacionado aos diversos tipos de integrações disponíveis para utilização no servidor. Tomando como exemplo dispositivos de mídia inteligentes, existem integrações que possibilitam o controle via *internet* desses dispositivos. Alguns exemplos de aparelhos de televisão que possuem essas integrações são *Android TV*, *Apple TV*, *LG WebOS Smart TV*, *Philips TV*, *Samsung Smart TV*, *Sony Bravia TV*, entre outros. Essas integrações não foram abordadas no projeto pois o escopo definido foi para controle via comunicação infravermelho, sendo esse tipo de comunicação mais comum, mais genérica e mais simples de implementar que a utilização de integrações específicas para cada tipo de dispositivo.

Referências

[1] Bing, K., L. Fu, Y. Zhuo e L. Yanlei: *Design of an Internet of Things-based smart home system.* Em *2011 2nd International Conference on Intelligent Control and Information Processing*, volume 2, páginas 921–924, 2011. 1

[2] Atzori, Luigi, Antonio Iera e Giacomo Morabito: *The Internet of Things: A survey.* Computer Networks, 54(15):2787 – 2805, 2010, ISSN 1389-1286. `http://www.sciencedirect.com/science/article/pii/S1389128610001568`, Acessado em: 16-05-2021. 3

[3] *INFSO D.4 Networked Enterprise RFID INFSO G.2 Micro Nanosystems (DG INFSO), in co-operation with the RFID Working Group of the European Technology Platform on Smart Systems Integration (EPOSS), Internet of Things in 2020: A Roadmap for the Future, Version 1.1*. maio 2008. 3

[4] Gubbi, Jayavardhana, Rajkumar Buyya, Slaven Marusic e Marimuthu Palaniswami: *Internet of Things (IoT): A vision, architectural elements, and future directions.* Future Generation Computer Systems, 29(7):1645 – 1660, 2013, ISSN 0167-739X. `http://www.sciencedirect.com/science/article/pii/S0167739X13000241`, Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. 4

[5] Lutolf, R.: *Smart Home concept and the integration of energy meters into a home based system.* Em *Seventh International Conference on Metering Apparatus and Tariffs for Electricity Supply 1992*, páginas 277–278, 1992. 5

[6] Alam, Muhammad Raisul, Mamun Bin Ibne Reaz e Mohd Mohd Ali: *A Review of Smart Homes – Past, Present, and Future.* IEEE Transactions on Systems, Man, and Cybernetics -Part C: Applications and Reviews, 42:1190–1203, novembro 2012. 6

[7] Chan, Marie, Eric Campo, Daniel Estève e Jean Yves Fourniols: *Smart homes — Current features and future perspectives.* Maturitas, 64(2):90 – 97, 2009, ISSN 0378-5122. `http://www.sciencedirect.com/science/article/pii/S0378512209002606`, Acessado em: 16-05-2021. 6

[8] Winkler, B.: *An Implementation of an Ultrasonic Indoor Tracking System Supporting the OSGI Architecture of the ICTA Lab.* University of Florida, 2002. `https://books.google.com.br/books?id=ZCfZjwEACAAJ`, Acessado em: 16-05-2021. 6

[9] Cambria, E. e B. White: *Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]*. IEEE Computational Intelligence Magazine, 9(2):48–57, 2014. 7

[10] Collobert, Ronan and Weston, Jason and Bottou, Leon and Karlen, Michael and Kavukcuoglu, Koray and Kuksa, Pavel: *Natural language processing (almost) from scratch*. Computing Research Repository - CORR, 12, março 2011. 7

[11] *Hey Google - Overview - Discover what Google Assistant is.* https://assistant.google.com. Acessado em: 16-05-2021. 7, 36, 37

[12] *Microsoft - O que é a Cortana?* https://support.microsoft.com/pt-br/topic/o-que-é-a-cortana-953e648d-5668-e017-1341-7f26f7d0f825. Acessado em: 16-05-2021. 7

[13] *Amazon Alexa Official Site: What is Alexa?* https://developer.amazon.com/en-US/alexa. Acessado em: 16-05-2021. 7

[14] Ibrahim, Dogan: *Chapter 1 - Introduction*. Em Ibrahim, Dogan (editor): *ARM-Based microcontroller projects using MBED*, páginas 1 – 7. Newnes, 2019, ISBN 978-0-08-102969-5. http://www.sciencedirect.com/science/article/pii/B978008102969500001X, Acessado em: 16-05-2021. 9

[15] Banks, Andrew e Rahul Gupta (editores): *MQTT Version 3.1.1 Plus Errata 01*. OASIS Standard Incorporating Approved Errata 01, December 2015. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html, Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html, Acessado em: 16-05-2021. 9

[16] *HiveMQ - MQTT Essentials - The Ultimate Kickstart For MQTT Beginners.* https://www.hivemq.com/mqtt-essentials/. Acessado em: 16-05-2021. 9, 10, 11, 12

[17] Hillar, Gaston C: *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017. 10, 13

[18] *RedHat - Interface de programação de aplicações - O que é API?* https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces. Acessado em: 16-05-2021. 13

[19] Fielding, Roy Thomas: *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, Acessado em: 16-05-2021. 13, 14

[20] *Twilio SendGrid - Webhook vs API: What's the Difference Between Them?* https://sendgrid.com/blog/webhook-vs-api-whats-difference/. Acessado em: 16-05-2021. 14

[21] *SparkFun Electronics - Tutorials - IR Communication.* https://learn.sparkfun.com/tutorials/ir-communication/all. Acessado em: 16-05-2021. 14, 15

[22] *Espectro eletromagnético apresentando as regiões de radiações ionizantes e não ionizantes.* https://commons.wikimedia.org/wiki/File:EM_spectrum_pt_2.svg. Acessado em: 16-05-2021. 15

[23] *SparkFun Electronics - Tutorials - Light - Infrared Light.* https://learn.sparkfun.com/tutorials/light/infrared-light. Acessado em: 16-05-2021. 15, 16

[24] *IRremoteESP8266 - Infrared remote library for ESP8266/ESP32: send and receive infrared signals with multiple protocols.* https://github.com/crankyoldgit/IRremoteESP8266. Acessado em: 24-05-2021. 16, 17

[25] *TechTudo - 9 em cada 10 brasileiros usam celular Android, diz relatório do Google.* https://www.techtudo.com.br/noticias/2020/09/9-em-cada-10-brasileiros-usam-celular-android-diz-relatorio-do-google.ghtml. Acessado em: 16-05-2021. 19

[26] *Home Assistant - Awaken your home.* https://www.home-assistant.io. Acessado em: 16-05-2021. 20, 21

[27] *Oracle® VM VirtualBox® User Manual - Chapter 6. Virtual Networking.* https://www.virtualbox.org/manual/ch06.html. Acessado em: 16-05-2021. 21

[28] *nmcli — command-line tool for controlling NetworkManager.* https://developer.gnome.org/NetworkManager/stable/nmcli.html. Acessado em: 16-05-2021. 21

[29] *ESPHome - Getting Started with ESPHome and Home Assistant.* https://esphome.io/guides/getting_started_hassio.html. Acessado em: 16-05-2021. 23, 24

[30] *Home Assistant - MQTT Integration.* https://www.home-assistant.io/integrations/mqtt/. Acessado em: 16-05-2021. 23

[31] *Hass.io Add-on: ngrok Client.* https://community.home-assistant.io/t/hass-io-add-on-ngrok-client/164528. Acessado em: 16-05-2021. 23, 25, 38

[32] *Home Assistant Add-on: Samba share.* https://github.com/home-assistant/addons/blob/master/samba/README.md. Acessado em: 16-05-2021. 23

[33] *Home Assistant Community Add-on: Visual Studio Code.* https://community.home-assistant.io/t/home-assistant-community-add-on-visual-studio-code/107863. Acessado em: 16-05-2021. 23

[34] *ESPHome - Frequently Asked Questions - I can't get flashing over USB to work.* https://esphome.io/guides/faq.html#esphome-flasher. Acessado em: 16-05-2021. 24

[35] *ngrok Documentation.* https://ngrok.com/docs. Acessado em: 16-05-2021. 25

[36] *ESPHome - ESPHome is a system to control your ESP8266/ESP32 by simple yet powerful configuration files and control them remotely through Home Automation systems.* https://esphome.io/index.html. Acessado em: 16-05-2021. 26

[37] *ESPHome - Native API Component*. https://esphome.io/components/api.html. Acessado em: 16-05-2021. 26, 27, 34

[38] *Protocol Buffers*. https://developers.google.com/protocol-buffers. Acessado em: 16-05-2021. 26

[39] *ESPHome native API protocol data structure definitions*. https://github.com/esphome/esphome/blob/dev/esphome/components/api/api.proto. Acessado em: 16-05-2021. 26, 27

[40] *ESP8266EX Datasheet - Version 4.3 - Espressif Systems IoT Team*. https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf. Accessed: 2021. 27, 28

[41] *Acrobotic - Fun projects with DIY Electronics and Programming!* https://acrobotic.com. Acessado em: 16-05-2021. 29, 31

[42] *NodeMCU 3.0.0*. https://github.com/nodemcu/nodemcu-firmware. Acessado em: 16-05-2021. 28

[43] *NodeMCU – Uma plataforma com características singulares para o seu projeto IoT*. https://blogmasterwalkershop.com.br. Acessado em: 16-05-2021. 29

[44] *NodeMCU V1 V2 V3 size and pinout comparison*. https://www.reddit.com/r/esp8266/comments/9q3314/nodemcu_v1_v2_v3_size_and_pinout_comparison/. Acessado em: 16-05-2021. 30

[45] *ESPHome - ESPHome Core Configuration*. https://esphome.io/components/esphome.html. Acessado em: 16-05-2021. 31

[46] *ESPHome - WiFi Component*. https://esphome.io/components/wifi.html. Acessado em: 16-05-2021. 32

[47] *ESPHome - Captive Portal*. https://esphome.io/components/captive_portal.html. Acessado em: 16-05-2021. 32

[48] *ESPHome - Logger Component*. https://esphome.io/components/logger.html. Acessado em: 16-05-2021. 33

[49] *ESPHome - MQTT Client Component*. https://esphome.io/components/mqtt.html. Acessado em: 16-05-2021. 33

[50] *Home Assistant - Local IP Address Integration*. https://www.home-assistant.io/integrations/local_ip/. Acessado em: 16-05-2021. 33

[51] *Home Assistant - ESPHome Integration*. https://www.home-assistant.io/integrations/esphome/. Acessado em: 16-05-2021. 34

[52] *ESPHome - OTA Update Component*. https://esphome.io/components/ota.html. Acessado em: 16-05-2021. 35

[53] *ESPHome - NodeMCU ESP8266*. https://esphome.io/devices/nodemcu_esp8266.html. Acessado em: 16-05-2021. 35

[54] *ESPHome - ESP8266 Software PWM Output*. https://esphome.io/components/output/esp8266_pwm.html. Acessado em: 16-05-2021. 35

[55] *ESPHome - Output Component*. https://esphome.io/components/output/index.html. Acessado em: 16-05-2021. 35

[56] *ESPHome - Configuration Types*. https://esphome.io/guides/configuration-types.html. Acessado em: 16-05-2021. 35

[57] *ESPHome - Binary Light*. https://esphome.io/components/light/binary.html. Acessado em: 16-05-2021. 35

[58] *How to integrate Google Assistant with Home Assistant without a subscription*. https://www.juanmtech.com/integrate-google-assistant-with-home-assistant-without-a-subscription. Acessado em: 16-05-2021. 37

[59] *Google Assistant - Conversational Actions - Dialogflow and legacy Actions SDK - Overview*. https://developers.google.com/assistant/conversational/df-asdk/discovery. Acessado em: 16-05-2021. 37

[60] *Google Assistant - Smart Home - Fulfillment*. https://developers.google.com/assistant/smarthome/concepts/fulfillment-authentication. Acessado em: 16-05-2021. 37

[61] *Home Assistant - Duck DNS Integration*. https://www.home-assistant.io/integrations/duckdns/. Acessado em: 16-05-2021. 38

[62] *ESPHome - Sensor Component*. https://esphome.io/components/sensor/index.html. Acessado em: 16-05-2021. 38

[63] *Home Assistant - Google Assistant Integration*. https://www.home-assistant.io/integrations/google_assistant/. Acessado em: 16-05-2021. 39, 40, 41, 62

[64] *Google Assistant - Smart Home - Home Graph*. https://developers.google.com/assistant/smarthome/concepts/homegraph. Acessado em: 16-05-2021. 40

[65] *KY-022 Infrared Receiver Module*. https://arduinomodules.info/ky-022-infrared-receiver-module/. Acessado em: 16-05-2021. 42

[66] *ESPHome - Remote Receiver*. https://esphome.io/components/remote_receiver.html. Acessado em: 16-05-2021. 43

[67] *Silicon NPN transistor in a TO-92 Plastic Package*. https://pdf1.alldatasheet.com/datasheet-pdf/view/960521/FOSHAN/2N2222.html. Acessado em: 16-05-2021. 44

[68] *Ohm's law*, ISBN 9780191727641. https://www.oxfordreference.com/view/10.1093/oi/authority.20110803100247443, Acessado em: 16-05-2021. 44, 45

[69] *NPN Transistor*. https://www.electronics-tutorials.ws/transistor/tran_2.html. Acessado em: 16-05-2021. 44

[70] *Tech Sul Eletrônicos*. https://techsuleletronicos.com.br/. Acessado em: 16-05-2021. 45

[71] *ESPHome - Remote Transmitter*. https://esphome.io/components/remote_transmitter.html. Acessado em: 16-05-2021. 45

[72] *ESPHome - Switch Component*. https://esphome.io/components/switch/index.html. Acessado em: 16-05-2021. 46

[73] *ESPHome - Templates (Lambdas)*. https://esphome.io/guides/automations.html. Acessado em: 16-05-2021. 46

[74] *LM35DZ Datasheet (PDF) - National Semiconductor (TI)*. https://pdf1.alldatasheet.com/datasheet-pdf/view/8875/NSC/LM35DZ.html. Acessado em: 16-05-2021. 47

[75] *LM35 Temperature Sensor (With Pinouts)*. https://www.researchgate.net. Acessado em: 16-05-2021. 47

[76] *ESPHome - Analog To Digital Sensor*. https://esphome.io/components/sensor/adc.html. Acessado em: 16-05-2021. 48, 55

[77] *Google Assistant - Smart Home - Smart Home Device Types*. https://developers.google.com/assistant/smarthome/guides. Acessado em: 16-05-2021. 48

[78] *ESPHome - Climate Component*. https://esphome.io/components/climate/index.html. Acessado em: 16-05-2021. 49

[79] *Home Assistant - MQTT HVAC Integration*. https://www.home-assistant.io/integrations/climate.mqtt/. Acessado em: 16-05-2021. 49

[80] *IR Remote Control interface on ESP8266*. https://github.com/esp8266/Arduino/issues/2899. Acessado em: 16-05-2021. 56

[81] *Home Assistant - Universal Media Player Integration*. https://www.home-assistant.io/integrations/universal/. Acessado em: 16-05-2021. 57

Anexo I

Código descrito no Capítulo 3 e utilizado para testes no Capítulo 4

```
esphome:
  name: nodemcu
  platform: ESP8266
  board: nodemcuv2

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  manual_ip:
    static_ip: !secret manual_ip_static_ip
    gateway: !secret manual_ip_gateway
    subnet: !secret manual_ip_subnet

  ap:
    ssid: "Nodemcu Fallback Hotspot"
    password: !secret ap_password
    ap_timeout: 1min

captive_portal:

logger:
  baud_rate: 0
  level: DEBUG
```

```yaml

mqtt:
  broker: !secret home_assistant_local_ip
  port: 1883
  username: !secret mqtt_username
  password: !secret mqtt_password
  discovery: true
  discovery_prefix: homeassistant
  topic_prefix: nodemcu
  keepalive: 15s

api:
  port: 6053
  password: !secret api_password

ota:
  safe_mode: true
  password: !secret ota_password
  port: 8266
  num_attempts: 10

output:
  - platform: esp8266_pwm
    id: nodemcu_led_output
    pin:
      number: D0
      inverted: true

light:
 - platform: binary
   id: nodemcu_led
   name: "Onboard LED"
   output: nodemcu_led_output

remote_transmitter:
  pin: D6
  carrier_duty_percent: 50%
```

```yaml

remote_receiver:
  pin:
    number: D7
    inverted: true
  dump: all

switch:
  - platform: template
    name: "TV Samsung"
    turn_on_action:
      remote_transmitter.transmit_samsung:
        data: 0xE0E06798
    turn_off_action:
      remote_transmitter.transmit_samsung:
        data: 0xE0E06798

  - platform: template
    name: "Power TV LG"
    turn_on_action:
      remote_transmitter.transmit_lg:
        data: 0x20DF10EF
        nbits: 32
    turn_off_action:
      remote_transmitter.transmit_lg:
        data: 0x20DF10EF
        nbits: 32

  - platform: template
    name: "Volume Up TV LG"
    turn_on_action:
      remote_transmitter.transmit_lg:
        data: 0x20DF40BF
        nbits: 32
    turn_off_action:
      remote_transmitter.transmit_lg:
        data: 0x20DF40BF
```

```
99          nbits: 32
100
101   - platform: template
102     name: "Volume Down TV LG"
103     turn_on_action:
104       remote_transmitter.transmit_lg:
105         data: 0x20DFC03F
106         nbits: 32
107     turn_off_action:
108       remote_transmitter.transmit_lg:
109         data: 0x20DFC03F
110         nbits: 32
111
112   - platform: template
113     name: "Mute TV LG"
114     turn_on_action:
115       remote_transmitter.transmit_lg:
116         data: 0x20DF906F
117         nbits: 32
118     turn_off_action:
119       remote_transmitter.transmit_lg:
120         data: 0x20DF906F
121         nbits: 32
122
123   - platform: template
124     name: "Entrada TV LG"
125     turn_on_action:
126       remote_transmitter.transmit_lg:
127         data: 0x20DFD02F
128         nbits: 32
129     turn_off_action:
130       remote_transmitter.transmit_lg:
131         data: 0x20DFD02F
132         nbits: 32
133
134   - platform: template
135     name: "Power NET"
```

```
136     turn_on_action:
137       remote_transmitter.transmit_nec:
138         address: 0xE17A
139         command: 0x48B7
140     turn_off_action:
141       remote_transmitter.transmit_nec:
142         address: 0xE17A
143         command: 0x48B7
144
145   - platform: template
146     name: "Volume Up NET"
147     turn_on_action:
148       remote_transmitter.transmit_nec:
149         address: 0xE17A
150         command: 0xB04F
151     turn_off_action:
152       remote_transmitter.transmit_nec:
153         address: 0xE17A
154         command: 0xB04F
155
156   - platform: template
157     name: "Volume Down NET"
158     turn_on_action:
159       remote_transmitter.transmit_nec:
160         address: 0xE17A
161         command: 0x708F
162     turn_off_action:
163       remote_transmitter.transmit_nec:
164         address: 0xE17A
165         command: 0x708F
166
167   - platform: template
168     name: "Proximo Canal NET"
169     turn_on_action:
170       remote_transmitter.transmit_nec:
171         address: 0xE17A
172         command: 0x08F7
```

```
173     turn_off_action:
174       remote_transmitter.transmit_nec:
175         address: 0xE17A
176         command: 0x08F7
177 
178   - platform: template
179     name: "Canal Anterior NET"
180     turn_on_action:
181       remote_transmitter.transmit_nec:
182         address: 0xE17A
183         command: 0x58A7
184     turn_off_action:
185       remote_transmitter.transmit_nec:
186         address: 0xE17A
187         command: 0x58A7
188 
189 sensor:
190   - platform: adc
191     pin: A0
192     name: "Temperatura Ambiente"
193     update_interval: 60s
194     unit_of_measurement: "Celsius"
195     filters:
196       - lambda: return (x * 3.3)/0.01;
```

Anexo II

Código de programação das entidades *Universal Media Player* criadas no Capítulo 4

```
media_player:
  - platform: universal
    name: "TV LG"
    commands:
      turn_on:
        service: switch.turn_on
        target:
          entity_id: switch.power_tv_lg
      turn_off:
        service: switch.turn_off
        target:
          entity_id: switch.power_tv_lg
      volume_up:
        service: switch.turn_on
        target:
          entity_id: switch.volume_up_tv_lg
      volume_down:
        service: switch.turn_on
        target:
          entity_id: switch.volume_down_tv_lg
      volume_mute:
        service: switch.turn_on
```

```
23          target:
24            entity_id: switch.mute_tv_lg
25      attributes:
26        state: switch.tv_lg
27        is_volume_muted: switch.mute_tv_lg
28      device_class: tv
29
30    - platform: universal
31      name: "NET"
32      commands:
33        turn_on:
34          service: switch.turn_on
35          target:
36            entity_id: switch.power_net
37        turn_off:
38          service: switch.turn_off
39          target:
40            entity_id: switch.power_net
41        volume_up:
42          service: switch.turn_on
43          target:
44            entity_id: switch.volume_up_net
45        volume_down:
46          service: switch.turn_on
47          target:
48            entity_id: switch.volume_down_net
49      device_class: tv
```