MINICURSOS DO

XIV ENCONTRO UNIFICADO DE COMPUTAÇÃO DO PIAUÍ (ENUCOMPI) E XI SIMPÓSIO DE SISTEMAS DE INFORMAÇÃO (SINFO)

ENUCOMPI
ENCONTRO UNIFICADO DE COMPUTAÇÃO DO PIAUÍ

XI SINFO
Simpósio de Sistemas de Informação

Organização

Antonio Oseas de Carvalho Filho (UFPI)
Deborah Maria Vieira Magalhães (UFPI)
Rodrigo Augusto Rocha Souza Baluz (UESPI)
Romuere Rodrigues Veloso e Silva (UFPI)

SBC
Sociedade Brasileira de Computação

23 a 25 de novembro de 2021
Picos - Piauí

Minicursos do XIV Encontro Unificado de Computação do Piauí (ENUCOMPI) e XI Simpósio de Sistemas de Informação (SINFO)

23 a 25 de novembro de 2021, Picos - Piauí
EVENTO ONLINE

Editora

Sociedade Brasileira de Computação

Organizadores

Antonio Oseas de Carvalho Filho (UFPI)
Deborah Maria Vieira Magalhães (UFPI)
Rodrigo Augusto Rocha Souza Baluz (UESPI)
Romuere Rodrigues Veloso e Silva (UFPI)

Realização

Universidade Federal do Piauí

Dados Internacionais de Catalogação na Publicação (CIP)

E56 Encontro Unificado de Computação do Piauí (14. : 2021 : Picos, PI)

Minicursos do XIV Encontro Unificado de Computação do Piauí (ENUCOMPI) e XI Simpósio de Sistemas de Informação (SINFO) [recurso eletrônico] – organizadores: Antonio Oseas de Carvalho Filho, Deborah Maria Vieira Magalhães, Rodrigo Augusto Rocha Souza Baluz, Romuere Rodrigues Veloso e Silva. – Porto Alegre : SBC, 2021.

ISBN 978-65-87003-69-6

1. Sistemas de informação. 2. Computação. I. Carvalho Filho, Antonio Oseas de. II. Magalhães, Deborah Maria Vieira. III. Baluz, Rodrigo Augusto Rocha Souza. IV. Silva, Romuere Rodrigues Veloso. V. Título.

CDU 004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

Capa (Foto):

Firmino Azevedo Neto UFPI, Rodrigo Augusto Rocha Souza Baluz, UESPI

Capa (Edição):

Firmino Azevedo Neto UFPI, Rodrigo Augusto Rocha Souza Baluz, UESPI

Editoração:
Antonio Oseas de Carvalho Filho UFPI, Romuere Rodrigues Veloso e Silva UFPI.

Sociedade Brasileira de Computação

Organização do ENUCOMPI / SINFO 2021

Coordenadores Gerais

Romuere Rodrigues Veloso e Silva, UFPI

Deborah Maria Vieira Magalhães, UFPI

Coordenadores da Trilha de Minicursos

Antonio Oseas de Carvalho Filho UFPI

Realização

Apoio Institucional

Organizações de apoio

Empresas parceiras

Apresentação

O **Encontro Unificado de Computação do Piauí (ENUCOMPI)** nasce na cidade de Parnaíba, litoral do Piauí, organizado por professores pesquisadores das instituições de ensino de Computação. Possui como metas contribuir para a troca de experiências, união entre acadêmicos e pesquisadores; no fortalecimento das parcerias para o desenvolvimento da educação e do empreendedorismo por meio da tecnologia; e, no incentivo à produção de trabalhos científicos ligados à área da Computação. Com a adesão da Secretaria Regional da Sociedade Brasileira de Computação no Piauí, o ENUCOMPI se consolida como referência no eixo Norte-Nordeste do Brasil na área de Computação. Mantemos parcerias com grandes revistas nacionais, como Revista IEEE América Latina, a Revista Brasileira de Computação Aplicada (RBCA), Revista de Informática Aplicada (RIA) e da Revista Learning & Nonlinear Models (L&NLM). O **Simpósio de Sistemas de Informação (SINFO)** tem como objetivo disseminar conhecimento técnico e científico sobre tecnologia de informação e comunicação, assim como incentivar empreendimentos de tecnologia no estado do Piauí, em especial na macrorregião de Picos e regiões vizinhas dos estados do Ceara e Maranhão. Nesse sentido, o SINFO busca a interação entre as instituições de ensino e empreendedores do setor de tecnologia por meio de palestras, painéis de discussões, apresentações de artigos técnicos e minicursos sobre temas e assuntos de vanguarda em tecnologia da informação e comunicação. A primeira edição do SINFO ocorreu em 2008, um ano após a UFPI criar a graduação em Sistemas de Informação no campus da cidade de Picos. O evento, juntamente com o curso de graduação, foi concebido pela UFPI com o objetivo estratégico de promover a inovação tecnológica e a relação entre a comunidade científica e a sociedade no interior do Piauí e semiárido nordestino.

Sinopse

O Livro de Minicursos ENUCOMPI / SINFO 2021 aborda conteúdos relacionados a multidisciplinaridade presente na área da Ciência da Computação. No primeiro capítulo os autores propõe uma forma prática para o “Desenvolvendo ChatBots com o Dialogflow”. Já no segundo capítulo que tem como título a “Detecção de Máscara em Python Usando OpenCV e Deep Learning”, os autores compartilham seus conhecimentos com exemplos práticos voltados a pandemia provocada pelo COVID-19. No capítulo “Emulando Redes de Comunicação para Sistemas de Múltiplos Drones: uma abordagem inicial”, é possível conhecer um pouco sobre abordagens direcionadas a comunicação com drones. No quarto capítulo os autores propõe uma série de “Procedimentos adotados na coleta de sinais mioelétricos para construção de próteses da mão”. No próximo capítulo, os autores apresentam uma “Introdução à Análise de Dados Geoespaciais com Python”. E por fim, no sexto capítulo, são apresentadas técnicas relacionadas a “Realidade Aumentada no contexto de Computação Ubíqua: conceitos, características e ferramentas da plataforma Android”.

Comitês

Comitês de Organização

Coordenadores Gerais

Romuere Rodrigues Veloso e Silva, UFPI – Picos

Deborah Maria Vieira Magalhães, UFPI – Picos

Coordenadores da Trilha de Minicursos

Antonio Oseas de Carvalho Filho, UFPI – Picos

Coordenação do Comitê de Programa

Flávio Henrique Duarte de Araújo, UFPI – Picos

Coordenação MCAS e PSTI

Ricardo de Andrade Lira Rabêlo, UFPI – Teresina

Coordenação de Comunicação

Francisco Airton Pereira da Silva, UFPI – Picos

Secretário Regional SBC Piauí

Rodrigo Augusto Rocha Souza Baluz, UESPI

Representante Institucional SBC UFPI Picos

Glauber Dias Gonçalves, UFPI – Picos

Sumário

Capítulo 1

Desenvolvendo ChatBots com o Dialogflow

Joeckson Correa, Davi Viana, Ariel Teles

Abstract

ChatBots are increasingly present in people's lives, whether in the financial area, customer service, education, and also in the health area. ChatBots bring benefits, for example, to the healthcare area, being able to assist patients and healthcare professionals in administrative tasks of appointment scheduling, in the area of education, and can answer students' questions about re-enrollment. This book chapter aims to present the ChatBot concepts, the origin of the term ChatBot and its evolution and architecture, and how Machine Learning and Natural Language Processing are used in ChatBots. In particular, it introduces the Dialogflow platform. Dialogflow is a web platform that facilitates the ChatBot development process. By using Dialogflow, the developer can create ChatBots that respond to users through text or speech. Finally, this chapter presents the step by step of how to create a ChatBot using Dialogflow to schedule medical appointments.

Resumo

Os ChatBots estão cada vez mais presentes na vida das pessoas, seja na área financeira, atendimento ao cliente, educação, e também na área da saúde. Os ChatBots trazem benefícios, por exemplo, para área da saúde, podendo auxiliar pacientes e profissionais de saúde em tarefas administrativas de agendamento de consulta, na área da educação, podendo tirar dúvidas de alunos sobre rematrícula. Esse capítulo de livro tem como objetivo apresentar os conceitos de ChatBots, a origem do termo ChatBot e sua evolução, arquitetura de um ChatBot, e de que forma o Aprendizado de Máquina e o Processamento de Linguagem Natural são utilizados em ChatBots. Em particular, ele apresenta a plataforma Dialogflow. O Dialogflow é uma plataforma web que facilita a criação de ChatBots. Ao utilizar o Dialogflow, o desenvolvedor pode criar ChatBots que possam responder aos usuários através de texto ou fala. Ao final, o capítulo apresenta o passo a passo de como criar um ChatBot, utilizando o Dialogflow, para agendamento de consultas médica.

1.1. Introdução

Um ChatBot é um agente de conversação que usa Processamento de Linguagem Natural (PLN) para se comunicar com os usuários [Pérez-Soler et al. 2021]. Existem vários tipos de ChatBots sendo utilizados em diversos domínios. Este capítulo explora os conceitos, as tecnologias, as possibilidades de uso dos ChatBots em vários domínios de aplicação, e apresenta uma prática com a plataforma de desenvolvimento de ChatBots Dialogflow.

1.1.1. Origem do termo ChatBot

O conceito de ChatBot, ou robô de conversação, não é novo [Weizenbaum 1966]. Nos últimos anos, os ChatBots foram aplicados a diversos domínios, tais como saúde, educação, área financeira, varejo, dentre outras. Na década de 1950, Alan Turing publicou o artigo *Computing Machinery and Intelligence* [TURING 1950], que adquiriu grande notoriedade, em que foi apresentada e discutida a questão "As máquinas seriam capazes de pensar?". Para responder esse questionamento, Alan Turing desenvolveu um método, chamado Teste de Turing, com a finalidade de identificar a capacidade de uma máquina em ter o comportamento semelhante a um comportamento humano. Esse teste impulsionou os primeiros passos para o surgimento dos ChatBots. O Teste de Turing definia o computador como inteligente quando ele conseguisse dialogar com uma pessoa através de um *chat*, sem que ela percebesse estar conversando com uma máquina.

Os primeiros Chatbots foram projetados para imitar o comportamento humano, como a "ELIZA " [Weizenbaum 1966], em uma conversa baseada em textos, realizando ações específicas e delimitadas dentro de um escopo controlado. O ChatBot ELIZA é um robô conversacional com a finalidade de simular um psicólogo virtual, o qual utiliza a reformulação de trechos das frases que são capturados das entradas dos usuários, fazendo parecer que possui um vasto vocabulário. Tendo em vista ser a primeira tentativa de criar um software que pudesse passar no Teste de Turing, ELIZA é considerada a "mãe dos ChatBots" [Weizenbaum 1966].

O PARRY [Leptourgos and Corlett 2020]foi construído pelo psiquiatra americano Kenneth Colby em 1972. Este ChatBot tinha a finalidade de imitar o comportamento de um paciente com esquizofrenia. O PARRY tinha a característica de parecer estar adotando falsas crenças de ser assediado, subjugado, perseguido, acusado, maltratado, injustiçado, atormentado e depreciado. Enquanto ELIZA foi considerada uma simulação de um terapeuta, PARRY simulava um paciente esquizofrênico. No *script* utilizado pelo PARRY durante o diálogo, foram desenvolvidos diferentes tipos de comportamentos que simulavam o diálogo de forma mais profunda. As conversas não eram baseadas em perguntas e respostas, mas de forma fluida.

Outro ChatBot é ALICE, um acrônimo que significa *Artificial Linguistic Internet Computer Entity*. A primeira edição de ALICE[1] foi implementada em 1995 e o programa ganhou o "Loebner Prize" nos anos de 2000, 2001 e 2004. ALICE possui uma base de conhecimento constituída por centenas de fatos, citações e ideias de seu criador. Esse ChatBot apresenta um vocabulário de mais de 5.000 palavras, sendo programado para dar muitas informações a respeito do serviço que disponibiliza. O projeto ALICE[2]

[1] https://alicebot.org

[2] https://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html

é composto por uma base de conhecimento desenvolvida na linguagem AIML (acrônimo para Linguagem de Marcação de Inteligência Artificial) que descreve o comportamento do ChatBot, e também por um módulo chamado interpretador, que manipula a base de conhecimento e é disponibilizado em diferentes linguagens de programação. O projeto ALICE é disponibilizado através da licença pública GNU e divido em três módulos: especificações técnicas da linguagem AIML, conjunto de interpretadores AIML, e a base de conhecimento [Kraus 2007].

1.1.2. Conceitos

Os Chatbots [Softić et al. 2021] são soluções de software que podem interagir com seres humanos por meio de uma interface de *chat*, podendo ser em forma textual ou por voz. Eles são também conhecidos como *talkbots*, *smartbots*, *bots*, *chatterbots* ou agentes conversacionais. Os ChatBots tendem a se comunicar com o usuário e se comportar como um ser humano. Para isso ocorrer, precisam estar conectados a serviços de mensagem (e.g., Facebook Messenger, Telegram, WhatsApp), páginas web ou aplicativos móveis. Os ChatBots são classificados em dois tipos: baseados em regras e baseados em Inteligência Artificial (IA), apresentados a seguir.

1.1.3. Arquitetura de ChatBots

A arquitetura de um ChatBot baseado em IA, conforme mostra a Figura 1.1, pode ser dividida em três subsistemas: pré-processamento, classificação de intenção, e utilização de contexto e geração de respostas. Inicialmente, o usuário envia uma mensagem de entrada que a entende e produz uma resposta. O pré-processamento executa um conjunto de operações na mensagem para operações adicionais, tais como, tokenização, normalização, eliminação de palavras irrelevantes, identificação das partes do discurso e entidades, stemming, lemmatisation [Karve et al. 2018], conceitos explicados a seguir.

A classificação de intenção tem a finalidade de identificar a intenção da mensagem do usuário. As intenções são rótulos que representam um significado das mensagens de entrada dos usuários. Por exemplo, a mensagem do usuário "Olá, bom dia!" pode ser classificada como "intenção de saudação". Em relação a utilização de contexto e geração de respostas, tem-se a implementação da pilha para tratamento de contextos. Toda resposta está associada a um determinado contexto. Assim, quando uma resposta é gerada, o contexto correspondente é enviado para a pilha. A qualquer momento, a parte superior da pilha indicará o contexto atual da conversa. Esse contexto atual é utilizado como um filtro para selecionar a resposta correspondente à intenção da mensagem do usuário. Caso nenhuma reposta corresponda ao filtro de contexto, a pilha de contexto é exibida para recuperar o contexto anterior. Então as resposta são pesquisadas com o contexto como filtro. Esse processo é repetido até que uma resposta apropriada seja encontrada [Bocklisch et al. 2017].

1.1.4. ChatBot baseado em Regras

A abordagem baseada em regras utiliza o conceito de máquina de estado, a qual consiste em regras que determinam o conjunto de entrada necessário para a transição de um estado para outro [Mellado-Silva et al. 2020]. Isso significa que o ChatBot conduz o usuário com perguntas para chegar à resolução correta. As estruturas das perguntas e respostas

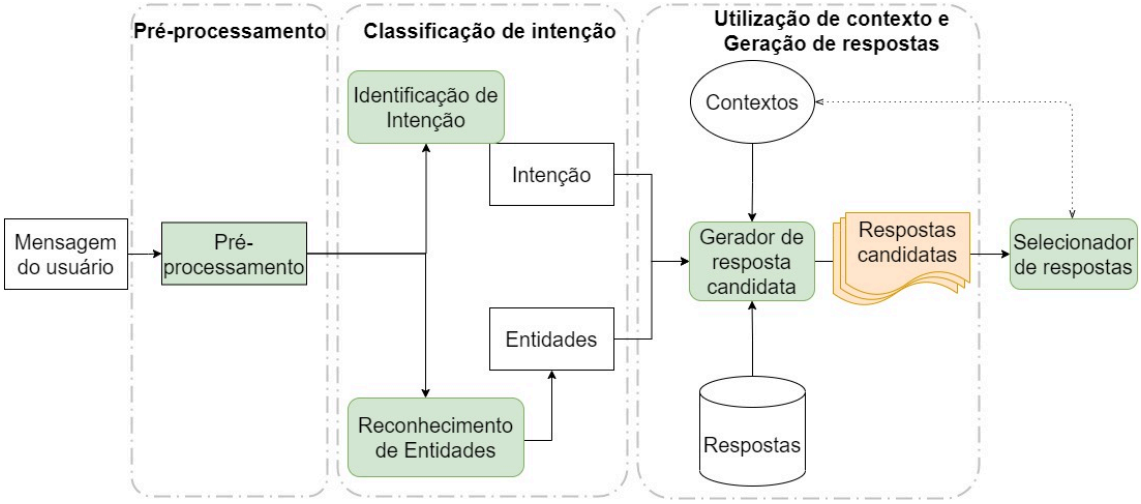


Figura 1.1. Arquitetura de um ChatBot. Adaptado de [Bocklisch et al. 2017].

são pré-definidas para que o ChatBot tenha o controle da conversa. Os ChatBots baseados em regras podem ser importantes para situações em que o objetivo de sua utilização seja pra realizar tarefas específicas, como requisitar segunda via de boleto ou acompanhar a situação de uma encomenda. Nesses casos, não se faz necessário a utilização de abordagens que fazem uso de IA. Os ChatBots baseados em regras realizam ações pré-estabelecidas, deixando claro suas limitações por não terem autonomia suficiente para resolver problemas mais elaborados [Kar and Haldar 2016a].

Como exemplo, considere um Chatbot baseado em regras que seja criado para uma instituição de ensino. Quando um usuário entra em contato com um ChatBot, é comum que ele responda com uma mensagem automática, por exemplo “Em que posso te ajudar?”. Podem ocorrer situações de buscar saber qual é o período de matrícula, documentação para rematrícula ou informação a respeito do contato telefônico da biblioteca. Caso o usuário responda algo diferente do que foi anteriormente sugerido pelo ChatBot, tal como “Quero saber minha nota de Matemática”, o ChatBot pode não conseguir entender a solicitação e provavelmente repetirá a mensagem automática.

1.1.5. ChatBot baseado em Inteligência Artificial

Os ChatBots baseados em IA [Kar and Haldar 2016b] são capazes de entender a linguagem natural e não somente os comandos predefinidos, desenvolvem inteligência à medida que interagem com os usuários. Além disso, eles conseguem manter diferentes contextos de conversas e fornecer ao usuário conversas mais ricas e engajadas. Em razão disso, surgiram conceitos de agentes virtuais e técnicas de reconhecimento de fala utilizadas em agentes virtuais, como por exemplo, Apple Siri, Amazon Alexa e Google Assistent. Certos elementos são extremamente importantes para o funcionamento dos ChatBots baseados em IA. Esses elementos são os classificadores de textos, algoritmos de aprendizado de máquina, redes neurais artificiais e processamento de linguagem natural. Como exemplo, os Chatbots baseados em IA podem realizar uma reserva de hotel após seguir solicitações do usuário ou realizar uma comparação de preços de reserva de hotel de vários sites ao mesmo tempo.

1.2. Processamento de Linguagem Natural

Nesta seção é abordado o PLN. É explicado como os ChatBots utilizam os principais componentes do PLN, tais como preparação dos dados, reconhecimento de entidades nomeadas, classificação de entidades nomeadas, compreensão de linguagem natural, extração de entidades e classificação de intenções.

1.2.1. Conceitos de Linguagem Natural

O PLN surgiu devido à necessidade de compreensão e comunicação automática do ser humano com o computador. O PLN é um mecanismo desenvolvido para extrair informações de textos, facilitar a entrada de dados nos sistemas e a estruturação de dados [Santos et al. 2014]. O PLN é uma área da Ciência da Computação e da Linguística que estuda os métodos formais para analisar textos e gerar frases em um idioma, através do uso de software [Aranha 2017].

Existem quatro etapas no PLN [BULEGON and MORO 2010]: análises morfológica, sintática, semântica e pragmática. A análise morfológica é o estudo de cada palavra presente no texto de forma independente. Essa análise tem a responsabilidade de definir artigos, substantivos, verbos e adjetivos armazenados em um dicionário de palavras com significados semelhantes, dentro de um domínio específico de conhecimento. A análise sintática faz uso do dicionário, primeiramente identificando possíveis relações entre as palavras. Em um segundo momento, ela identifica o sujeito, predicado, complementos nominais e verbais, adjuntos e apostos. Na análise semântica, ocorrem as relações dos termos ambíguos, de sufixos e afixos, ou seja, questões sobre significados associados aos componentes de uma palavra, o sentido real da frase ou palavra. Para a junção e visualização de todas as etapas, a análise pragmática faz a conexão de todo o processo e mostra visualmente o resultado.

1.2.2. Pré-processamento

Para modelar o texto apresentado pelo usuário, possibilitando que a máquina o entenda, é necessário o seu pré-processamento, o qual abstrai e estrutura do texto, deixando apenas o que é informação relevante. Esse pré-processamento reduz o vocabulário do texto e torna os dados menos esparsos, uma característica importante para o processamento computacional [Aranha 2017].

A **tokenização** (do inglês, *tokenization*) é o início do pré-processamento de texto. Ela ocorre quando uma sequência de caracteres é dividida e quando for delimitada por espaço em branco, vírgula, ponto ou outro delimitador. Cada divisão estabelecida é chamada de *token*.

O processo de **remoção de *stopwords*** envolve a eliminação de palavras que não devem ser mantidas no texto. As *stopwords* [Morais and Ambrósio 2007] são conhecidas por serem palavras não importantes na análise do texto. As preposições, pronomes, artigos, advérbios, e outras classes de palavras auxiliares são geralmente classificadas como *stopwords*.

A **normalização morfológica** é uma forma de aumentar a memorização dos diversos significados de um mesmo conceito. Ela visa evitar a repetição das representações de

uma palavra a um mesmo conceito [BULEGON and MORO 2010]. Por exemplo, do conceito de "objeto físico que consiste em um número de páginas atadas juntamente" tem-se a palavra "livro" com as seguintes representações "livro" e "livros". O processo de normalização propõe que essas duas formas sejam agrupadas em apenas uma, indicando que elas têm o mesmo significado.

Stemming faz parte do processo de normalização, podendo ser classificado por radicalização inflexional ou radicalização para a raiz. A radicalização inflexional tem como característica a utilização das flexões verbais, fazendo truncamentos que tornam as palavras, na maioria das vezes, de difícil compreensão. Por exemplo, "livro", "livros", "livreto" são substituídas pelo radical da palavra "livr" [BULEGON and MORO 2010].

A ***lemmatization*** faz parte do processo de normalização. Ele tem o objetivo de fazer a substituição das diversas formas de representação da palavra pela forma primitiva. As formas "livro", "livros" e "livraria" são todas transformadas para sua forma primitiva "livro" [BULEGON and MORO 2010].

1.2.3. Reconhecimento de Entidades Nomeadas

As entidades são utilizadas para representar pessoas, lugares, instituições, acontecimento, tempo, dentre outras. Entretanto, para reconhecer essas entidades, é necessário o reconhecimento dos objetos no texto [BULEGON and MORO 2010]. O Reconhecimento de Entidades Nomeadas (REN) tem como objetivo identificar as entidades nomeadas e classificá-las em categorias pré-definidas, tais como: Pessoa, Organização, Local [Light 1998]. Por exemplo:

"Gabriel Barbosa reside em São Luis e estuda na UFMA (Universidade Federal do Maranhão)"

Realizando o REN do exemplo, temos: [Gabriel Barbosa], [São Luis], [UFMA] e [Universidade Federal do Maranhão], respectivamente, entidades cujas categorias são: Pessoa, Local, Organização e Organização.

1.2.4. Compreensão de Linguagem Natural

A Compreensão de Linguagem Natural (em inglês, *Natural Language Understanding*) é uma subárea do PLN que usa a análise sintática e semântica do texto ou da fala para classificar o significado de uma frase. A sintaxe se refere à estrutura gramatical de uma frase, enquanto a semântica diz respeito ao seu significado. Na arquitetura de ChatBots, a NLU é um dos principais componentes, torna o ChatBot completo e reduz as chances do ChatBot não compreender o que o usuário deseja. As principais funcionalidade da NLU em um ChatBot são a classificação de intenções e extração das entidades [Bocklisch et al. 2017].

A intenção é uma determinada mensagem do usuário, aquilo que o usuário está tentando transmitir ou realizar durante a conversa. Por exemplo, se um usuário deseja consultar a previsão do tempo, ele pode dizer: "consultar a previsão do tempo", ou pode simplesmente escrever "previsão do tempo". Em ambos os casos, o ChatBot não vai se concentrar na frase completa. A ideia é que o ChatBot possa identificar qual a intenção e, assim, consiga dar uma resposta que faça sentido ao usuário [Gupta et al. 2021].

As entidades[3] são palavras-chave extraídas de uma mensagem do usuário. Por exemplo, o número do telefone. As entidades são os elementos sobre os quais a intenção está se referindo. Por exemplo, na frase "Quero um suco de uva", a entidade é a palavra "uva".

1.3. Aprendizado de Máquina

O AM é uma subárea da IA que permite criar programas de computador com a capacidade de aprender e executar tarefas [Forgy 1965]. O que torna o programa capaz de aprender por si só, usando um conjunto de dados que passam a representar experiências passadas. O AM é composto por diferentes áreas de pesquisa, tais como IA, probabilidade e estatística, teoria da complexidade computacional, teoria da informação, filosofia, psicologia, neurobiologia, dentre outros. Tem-se como tarefas do AM a classificação, regressão, agrupamento de dados, previsão de séries temporais, dentre outras. A utilização de AM tem crescido na construção de modelos para resolução de problemas em diversos domínios de aplicação, como visão computacional, reconhecimento de fala, e compreensão de texto [von Rueden et al. 2021].

Como exemplo de solução baseada em AM, considere um programa de computador que deve executar uma tarefa simples, como distinguir entre três variedades diferentes de flor de uma mesma espécie. Em vez de codificar um programa utilizando todo o conhecimento acerca das variedades da flor em questão, algumas características botânicas das três flores são apresentadas a um programa. Ele implementa um algoritmo de AM, que, através de um processo de treinamento, aprende a caracterizar uma flor baseado em suas características. Assim como os seres humanos aprendem a diferenciar as variedades de flores observando suas características, o programa de AM também pode aprender a realizar essa tarefa por meio da análise das características.

Existem tarefas descritivas e tarefas preditivas em AM. Nas tarefas descritivas, busca-se o desenvolvimento de algoritmos e modelos (i.e., um algoritmo treinado) que descreve os dados. Entre as tarefas descritivas, uma das principais é o agrupamento de dados [Forgy 1965], que busca separar os dados de maneira que dados semelhantes fiquem em um mesmo grupo. Um exemplo da aplicação de agrupamento de dados é o agrupamento de textos. Nesse caso, o algoritmo procura agrupar textos que abordem o mesmo assunto e separar em grupos diferentes os textos que abordam assuntos diferentes.

As tarefas preditivas podem ser divididas em tarefas de classificação e regressão. Nas tarefas de classificação, busca-se atribuir categorias predefinidas a exemplos de entrada. Por exemplo, um banco pode desenvolver um sistema para a classificação de seus clientes em duas categorias para fornecimento de empréstimo: SIM e NÃO. Por meio do histórico de crédito dos clientes, e também de dados como salário e tempo de emprego, o sistema pode aprender a distinguir os clientes para os quais o banco deve (SIM) ou não deve (NÃO) fornecer um empréstimo. Assim, tem-se um sistema de recomendação de crédito, cujas categorias a serem preditas para um novo cliente são SIM e NÃO (atributo de saída). Nas tarefas de regressão, objetiva-se prever o valor de uma variável numérica (atributo de saída, i.e., variável dependente), dadas outras variáveis (atributos de entrada, i.e., variáveis independentes). Assim, em vez de encontrar uma classe associada, como

[3] https://rasa.com/docs/rasa/nlu-training-data/

na classificação, deve-se encontrar uma função que mapeie um exemplo para um número.

O AM pode ser utilizado em ChatBots para ajudar a diagnosticar doenças. Em [Mathew et al. 2019] é proposto um ChatBot que utiliza um modelo de AM baseado no algoritmo *K Nearest Neighbor* (KNN). Esse modelo é treinado com um conjunto de dados de doenças e sintomas. Durante a conversa entre o ChatBot e o usuário, os sintomas são identificados pelo ChatBot, o qual consegue reconhecer a doença e recomenda o tratamento adequado.

1.3.1. Classificação de Intenções

A classificação de intenções é a categorização de forma automática de dados de texto com base nos objetivos do usuário. Um classificador de intenções analisa os textos e os categoriza em intenções. Isso é útil para entender as intenções por trás das mensagens do usuário, automatizar processos e obter informações úteis. Toda interação com o usuário tem um propósito, objetivo ou intenção. Quer eles queiram, por exemplo, fazer uma compra, solicitar mais informações, ou cancelar a assinatura de internet[Ali 2020].

A classificação de intenção pode usar AM e PLN para fazer a associação de palavras ou expressões de forma automática a uma determinada intenção do usuário. Por exemplo, um modelo de AM pode realizar um agendamento de uma consulta médica através das sentenças, "quero consultar" ou "consulta médica" que estão associadas à intenção "consultas paciente". Para que isso ocorra, o classificador de intenções precisa ser treinado com dados de exemplos, conhecidos como dados de treinamento ou frases de treinamento. Por exemplo, se o ChatBot for destinado a realizar tarefas de vendas de lanches, deve-se escolher sentenças como: "quero comprar lanche" ou "quero lanche". Após a definição das *tags*, pode ser dado início ao processo de treinamento do classificador de intenções, repassando os exemplos de textos ou frases de treinamento para cada sentença. Por exemplo, "Quero comprar um suco de goiaba".

1.3.2. *Deep Learning* e ChatBots

Deep Learning (DL), ou Aprendizado Profundo, é uma sub-área de ML. Mais especificamente, trata de Redes Neurais Artificiais, uma área que busca simular computacionalmente o cérebro enquanto máquina de aprendizado [Pacheco and Pereira 2018]. Atualmente é uma área de pesquisa extremamente ativa, que pode ser aplicada em diversas áreas, tais como reconhecimento de fala, visão computacional, e também ChatBots. Atualmente existem ChatBots baseados em DL que podem ser divididos em duas categorias: baseados no modelo de recuperação, e gerativo [Csaky 2019]. Essa subseção será focada em ChatBots no modelo gerativo.

O ChatBot gerativo é uma abordagem mais inovadora, e foi possibilitada pelo surgimento do DL. Os modelos gerativos foram desenvolvidos para resolver problemas em que o ChatBot não consegue responder uma resposta predefinida. ChatBots gerativos podem trabalhar com casos novos de mensagens de usuários porque não precisam contar com respostas predefinidas. Para que isso acorra, o modelo cria sua própria resposta, através da transformação dos dados de entrada do usuário utilizando a multiplicação de matrizes e funções não lineares que podem conter milhões de parâmetros. Os modelos gerativos baseados em DL conseguem dar aos usuários a sensação de es-

tar conversando com um ser humano real, por não haver respostas predefinidas. Esses modelos precisam de uma grande base de dados para aprenderem a construir suas respostas [Nguyen and Shcherbakov 2018]. O modelo *Sec2seq* [Csaky 2019] é um dos principais modelos de DL utilizados na abordagem de construção de ChatBots gerativos. A rede *Seq2seq* é um modelo de rede neural recorrente (do inglês, *Recurrent Neural Network* - RNNs) que tem sido bastante utilizada em diversas tarefas de PLN, como a tradução e a sumarização de textos.

1.4. Exemplos de ChatBots

1.4.1. SERMO

O SERMO [Denecke et al. 2021] é um ChatBot com funcionalidades que realizam intervenções baseadas em Terapia Cognitivo Comportamental (TCC). O SERMO se destaca dos demais ChatBots por utilizar técnicas de PLN e métodos de análise de emoções dos usuários a partir do diálogo. O SERMO fornece quatro módulos:

- Módulo de interação, que realiza o diálogo com o usuário para coletar informações sobre alguma situação que impactou o usuário, bem como a emoção associada à situação;
- Módulo de fornecimento de atividades e exercícios que disponibilizam aos usuários uma lista de atividades prazerosas divididas em três categorias: exercícios de *mindfulness*, exercícios de relaxamento e atividades de lazer;
- Módulo diário de eventos com emoções associadas, que registra o humor por meio de um controle deslizante com cinco *smileys* diferentes, de bons a ruins. Além da possibilidade de registrar vários humores em um dia;
- Módulo de fornecimento de informação disponibiliza explicações sobre as funcionalidades do SERMO, os princípios básicos da TCC e as emoções: medo, nojo, raiva, alegria, tristeza, culpa e vergonha.

1.4.2. Jamura

O Chatbot Jamura [Salvi et al. 2019] foi proposto para fornecer a capacidade de monitorar e controlar dispositivos domésticos conectados à Internet e responder a perguntas sobre a casa e o jardim do usuário. As informações da residência (e.g., temperatura, umidade, intensidade de luz, estado operacional das luzes, o estado operacional dos ventiladores em cada cômodo, a umidade do solo do jardim, estado da bomba d'água) podem ser acessadas e controladas por meio de comandos de linguagem natural usando o Jamura. O sistema de automação residencial proposto usa os conceitos da Internet das Coisas (IoT) para tornar os serviços, dispositivos e dados relacionados transparentes e acessíveis. O Jamura foi construído utilizando o Dialogflow e disponibilizado na plataforma Telegram.

1.4.3. EMMA

EMMA [Ghandeharioun et al. 2019] é um Chatbot pessoal de bem-estar emocional inteligente e tem o objetivo de fornecer sugestões de bem-estar, através de atividades relevantes

individuais ou sociais, que se enquadram em uma das categorias de psicoterapia, sendo psicologia positiva, intervenções cognitivo-comportamentais, metacognitivas ou somáticas. Como exemplo de atividades tem-se "Divulgar a alegria ligando para um amigo e transmitindo sua energia positiva" ou "Escrever um comentário positivo para a boa postagem de algum amigo".

1.5. Plataformas de Desenvolvimento de ChatBots

Atualmente existem diversas plataformas de desenvolvimento de ChatBots. Por exemplo, *Wit.ia* que é uma plataforma que implementa uma "Interface de Programação de Aplicações" (API), com instância publica e privada sem limites de requisições e pertence ao "Facebook" [Agarwala et al. 2019]. O *Botpress* que é plataforma de código aberto para criar ChatBots de alta qualidade, considerada entre os desenvolvedores a melhor plataforma para criar ChatBots em português [Sabharwal et al. 2019]. O *Botkit* que é um *framework* de código aberto escrito em *JavaScript*, com ele é possível fazer integração com diversas plataformas de mensagens, como *Slack*, *Messenger*, *Hangouts*, *Twilio*, *Webex* e *Facebook*, fornece também integração com API de PLN[4]. Nesta seção será dado uma visão geral nas plataformas "RASA" e "Dialogflow".

1.5.1. Rasa

O RASA [Mellado-Silva et al. 2020] é uma plataforma de desenvolvimento de ChatBots de código aberto, baseada em PLN e AM. Ela é comumente utilizada na comunidade de pesquisa, embora não ofereça uma infraestrutura em nuvem, como hospedagem gerenciada e escalabilidade. A plataforma possui dois componentes principais (RASA NLU e RASA Core) e ambos podem ser utilizados separadamente.

1.5.1.1. RASA NLU

O RASA NLU [Bocklisch et al. 2017] é responsável pelo PLN, fazendo o reconhecimento da entrada do usuário, categorizando a entidade e classificando a intenção. A plataforma dá flexibilidade para utilizar diferentes analisadores de linguagem natural que são baseados nas bibliotecas spaCy, NLTK, CoreNLP, e BERT.

As *pipelines* definem as ordem e as ações a serem executadas para o ChatBots entenderem corretamente as entradas dos usuários. O RASA, por ser de código aberto, oferece diversas formas de configurar suas *pipelines*, como, por exemplo, adicionando ou removendo etapas de processamento. A plataforma recomenda a utilização de duas *pipelines*: spaCy e Tensorflow [Bocklisch et al. 2017]. Uma das principais diferenças entre elas é o uso de vetores de palavras pré-treinadas pelo spaCy. Por exemplo, os dados de treinamento contêm a palavra "vinho" categorizada como bebida, mas não "champanhe". Caso a entrada do usuário seja "Eu quero champanhe", o spaCy consegue reconhecer que "vinho" e "champanhe" são palavras semelhantes. Isso faz com que o RASA tenha uma maior confiança de que champanhe é uma bebida, tendo como resultado o direcionamento do usuário para a intenção correta.

[4]https://botkit.ai/docs/v4/

1.5.1.2. RASA Core

O RASA Core [Bocklisch et al. 2017] é responsável por receber todas as intenções e entidades do usuário (i.e., saída do RASA NLU ou outra ferramenta de extração de entidades e classificação de intenções), e direciona uma ação a ser realizada através de um modelo de AM. Ele leva em consideração o histórico das conversas e os dados de treinamento. A plataforma atribui uma pontuação a todas as ações possíveis e executa aquela com maior valor.

A Figura 1.2 mostra a arquitetura de alto nível do RASA Core e, em seguida, ela é explicada.

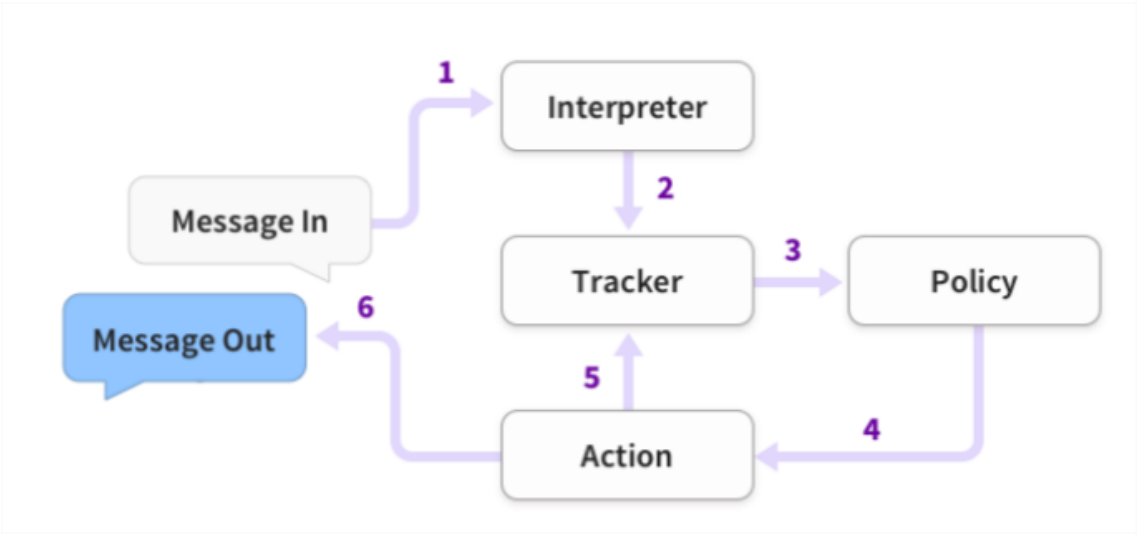


Figura 1.2. Arquitetura de alto nível do RASA Core [Bocklisch et al. 2017].

1. As mensagens de entrada do usuário são recebidas e repassadas para o módulo de interpretação (*Interpreter*), que converte todo o conteúdo em um dicionário constituído por texto original, intenções e entidades. O RASA Core não consegue interpretar as entradas dos usuário, logo é necessário um modelo de NLU (RASA NLU) para analisar as entradas.

2. O *Tracker* gerencia o estado da conversa com um usuário. Ele tem a responsabilidade de receber uma notificação caso uma nova mensagem da conversa chegue, armazenar o estado da conversa com um único usuário, e recuperar a mensagem da conversa armazenada.

3. A *Policy* recebe o estado atual do *Tracker*.

4. A *Policy* define qual a próxima ação a ser tomada nas etapas da conversa.

5. A *Action* escolhida é registrada pelo *Tracker*.

6. Por fim, ocorre o envio da resposta ao usuário.

1.5.2. Dialogflow

O Dialogflow é uma plataforma de PLN que facilita o design e a integração de uma interface do usuário conversacional com aplicações para dispositivos móveis, aplicações web, dispositivos, bots, sistemas interativos de resposta de voz, dentre outras.

O Dialogflow é uma plataforma de desenvolvimento de ChatBots que faz utilização de PLN. A plataforma suporta algumas tecnologias, linguagem de programação e bibliotecas, tais como Android, iOS, Webkit HTML5, JavaScript, Node.js, e Python. Os principais conceitos para o entendimento da plataforma Dialogflow são: agentes, intenções, entidades, contextos e *fullfilment* [Maldonado and Cuadra 2019a].

1.5.3. Agentes

Os agentes são assistentes virtuais que administram conversas com usuários finais. O agente pode ser considerado um módulo de PLN que entende as entradas dos usuários e pode ser integrado a outras aplicações ou canais de atendimento, tais como Facebook, WhatsApp e Telegram [Maldonado and Cuadra 2019a]. O Dialogflow estrutura um conjunto de intenções, as quais são cruciais para a definição do mapeamento de entradas para um conjunto de uma ou mais ações de respostas correspondentes.

1.5.4. Intenções

As intenções [Maldonado and Cuadra 2019b] servem para fazer um mapeamento entre a entrada do usuário e qual ação o ChatBot deve realizar. A intenção é composta por quatro componentes principais [Maldonado and Cuadra 2019b]: frases de treinamento, resposta, *ações* e *parâmetros*.

- **Frases de treinamento.** Para melhorar a forma de reconhecimento das entradas dos usuários, podem ser criadas frases de treinamento pré-definidas. Dessa forma, o ChatBot consegue entender as entradas dos usuários utilizando técnicas de PLN [Muhammad et al. 2020a].

- **Respostas.** São as possíveis saídas que o ChatBot pode retornar para o usuário, podendo ser textos pré-definidos ou ações processadas em serviços externos [Maldonado and Cuadra 2019a], por exemplo, consultar a previsão do tempo ou consultar a cotação do dólar.

- ções e *Parâmetros.* O Dialogflow não consegue trabalhar apenas com a correspondência de palavras no sentido literal fornecidas pelo usuário. É então necessária a utilização de intenções que utilizem entidades, tanto para entrada dos usuários quando na resposta dos ChatBots. Ao definir entidades nas frases de treinamento, a plataforma, automaticamente, transforma-as em *parâmetro* de uma *ação*.

1.5.5. Contextos

Os contextos são strings que representam o contexto atual da conversa do usuário. Eles são úteis para diferenciar frases que podem ser ambíguas e ter significados diferentes dependendo do que foi escrito anteriormente [Muhammad et al. 2020b].

1.5.6. *Fullfilment*

O Dialogflow permite utilizar dois tipos de resposta aos usuários: de forma estática ou processada através do componente chamado *fullfilment*. O *fullfilment* é uma lógica opcional usada pela plataforma para retornar respostas mais apropriadas e inteligentes de acordo com os dados extraídos na intenção do usuário [Muhammad et al. 2020b].

1.6. Desenvolvendo um ChatBot com o Dialogflow

Esta seção apresenta, em formato de tutorial passo a passo, o processo de criação de um ChatBot no Dialogflow para agendamento de consulta. A versão detalhada do desenvolvimento do ChatBot pode ser encontrada em: `https://github.com/josantosc/enucompi2021-minicurso-chatbot`.

1.6.1. Fluxo de Conversa

Para que o ChatBot consiga ter um diálogo eficiente com o usuário, é preciso projetar o fluxo do diálogo cuidadosamente. Para criar um fluxo de conversa, o primeiro passo é determinar o tema geral da conversa, depois determinar cada uma de suas cenas, bem como o personagem para ilustrar cada cena. Por último, criar um exemplo de conversa com a resposta esperada [Muhammad et al. 2020b].

O fluxo de conversa (Figura 1.3) é utilizado para desenvolver o ChatBot de agendamento de consulta. As intenções (representadas na cor cinza) utilizadas no fluxo são baseadas em correspondência de padrões. Assim, é disponibilizado a opção de utilizar um menu. Os menus estão disponíveis no momento que o usuário fornece uma entrada (representada na cor verde) durante o decorrer da conversa. O menu disponibilizado ao usuário tem o formato numérico para a seleção dos conteúdos desejados. Embora as entradas do usuário esperem uma resposta numérica, também existe a utilização de textos.

1.6.2. Acesso ao Dialogflow

O Dialogflow fornece ao usuário uma interface web chamada *Console Dialogflow*. Este console é utilizado para criar agentes, intenções, entidades e contextos. Para criar uma conta no Dialogflow é requirido uma conta do Google. Portanto, para realizar login na plataforma, caso não tenha uma conta, é necessário criar. Siga os seguintes passos para acessar a plataforma:

1. Acesse *https://Dialogflow.cloud.google.com/login*;
2. Clicar no botão *Sign-in with Google*;
3. Selecione a conta de acesso.

1.6.3. Criando o Primeiro Agente

Para criar o agente, é preciso seguir os passos abaixo:

1. Acessando o Console do Dialogflow, clique em “Criar Agente” (Figura 1.4) no menu à esquerda;

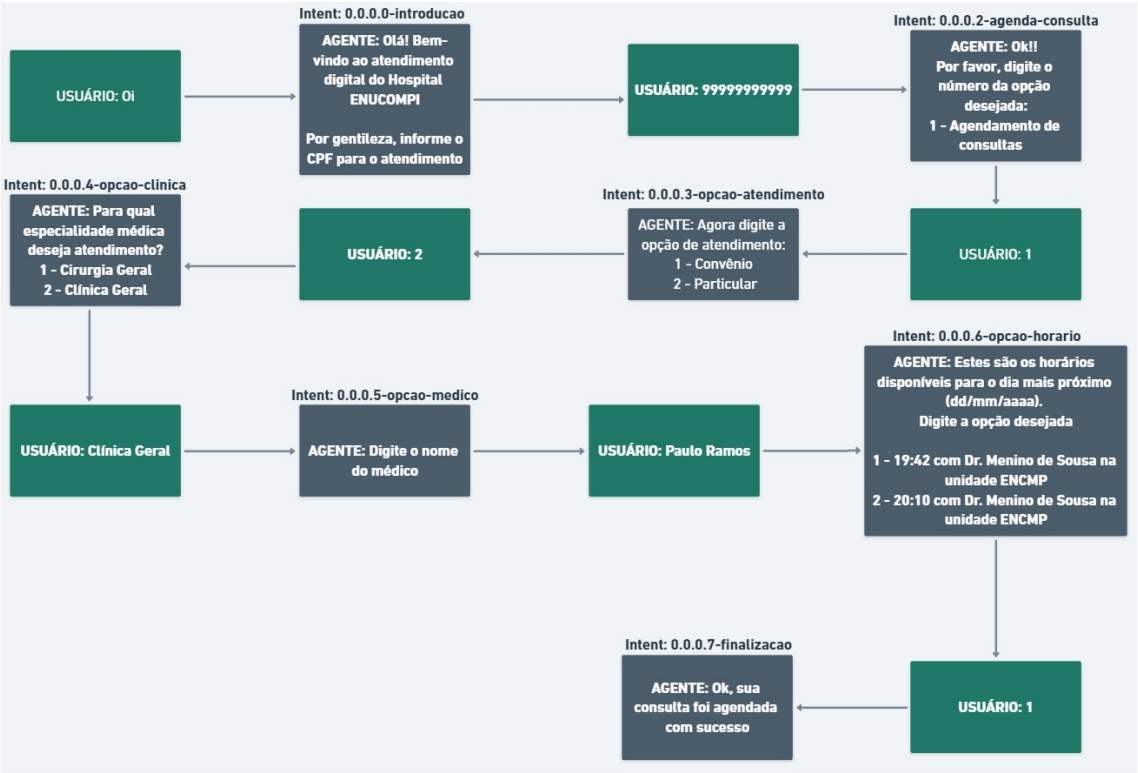


Figura 1.3. Fluxo de conversa de agendamento de consulta.

2. Digite o nome do agente, o idioma e fuso horário padrão;
3. Depois clique no botão Criar (Figura 1.5). É possível alterar a edição do agente após a criação.

1.6.4. Criação de Intenções

Nesta etapa, serão criadas as intenções do ChatBot. Para isso, siga os seguintes passos, considerando a Figura 1.6.

1. Clique no botão de adição **+** ao lado de *Intents* no menu da barra lateral à esquerda;
2. Insira um nome para a *intent*. O nome da *intent* precisa representar as expressões de usuário final que ela reconhece. Para o ChatBot que está sendo criado, a primeira intenção será chamada de "0.0.0.0-introducao";

Siga os passos a seguir para adicionar frases de treinamento, considerando a Figura 1.7).

1. Na seção *Training phrases*, clique em *Add user expression*;
2. Digite as frases de treinamento correspondentes a intenção em questão, e pressione "Enter" após cada entrada.

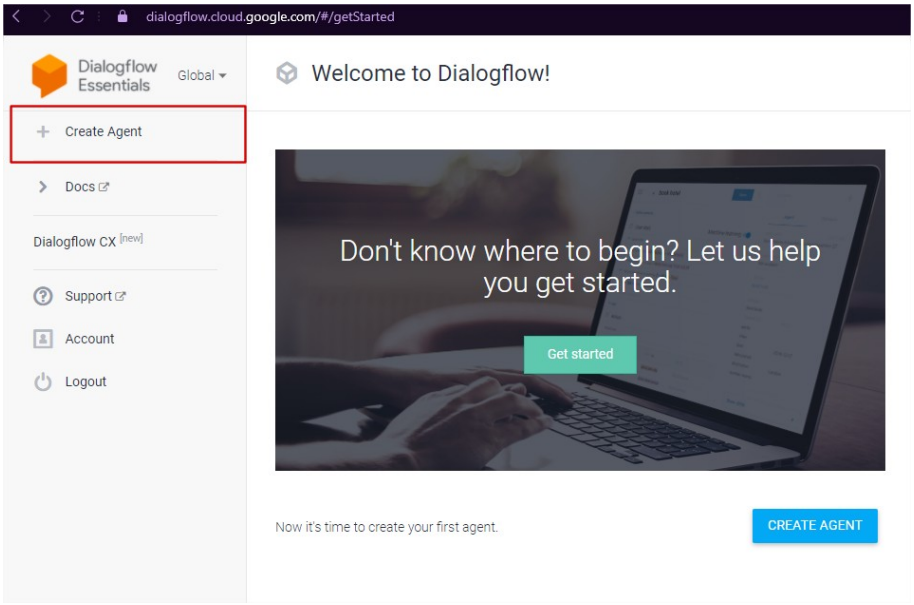


Figura 1.4. Tela inicial do Dialogflow para criar agente.

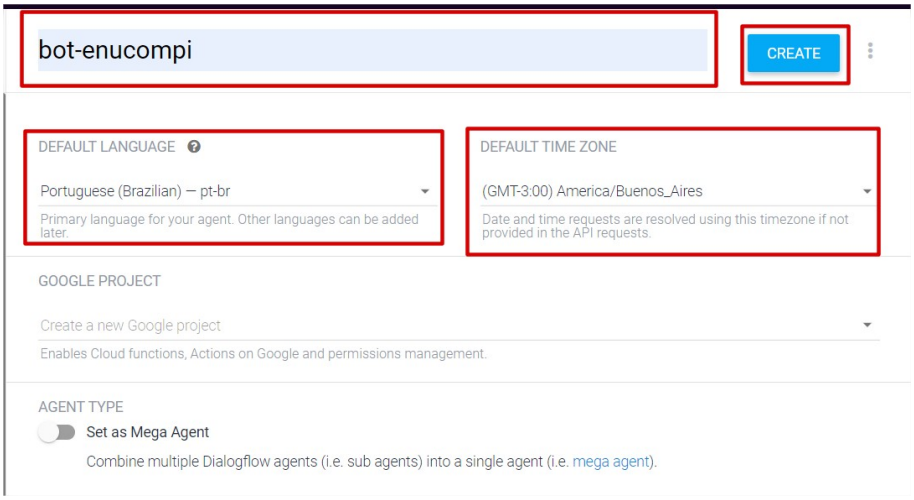


Figura 1.5. Criando agente no Dialogflow.

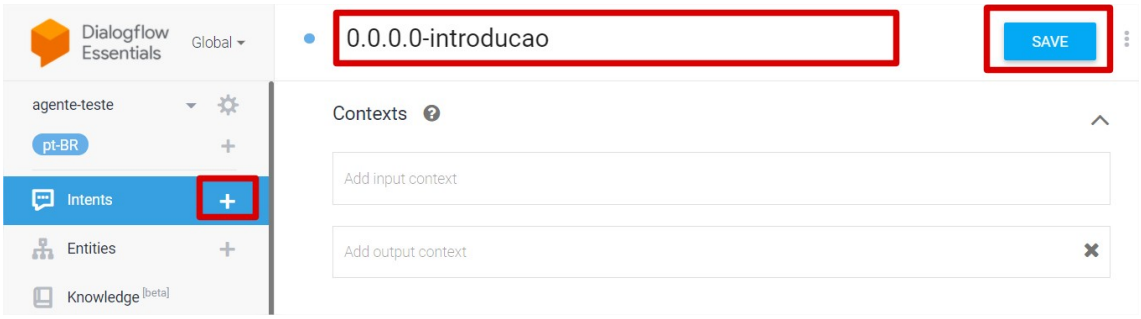


Figura 1.6. Criação de intenção no Dialogflow.

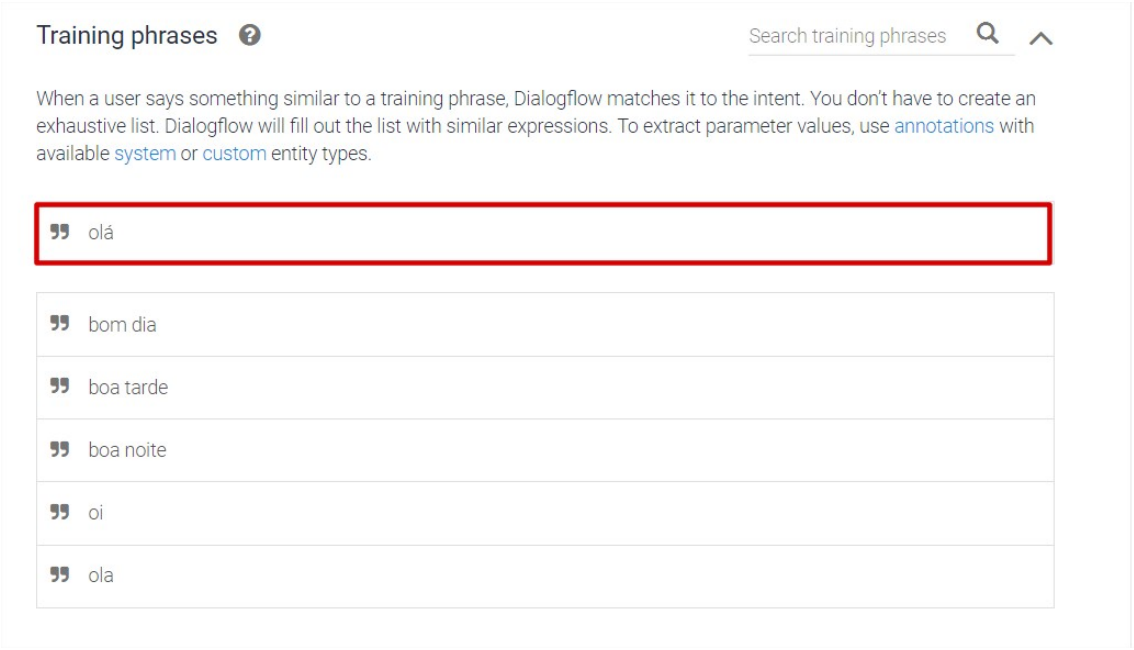


Figura 1.7. Adicionando frases de treinamento no Dialogflow.

A seguir são apresentados os passos para adicionar respostas às intenções, conforme a Figura 1.8.

1. Na seção *Responses*, digite a resposta da intenção na seção *Text Response*;
2. Para adicionar mais de uma resposta, clique no botão *ADD RESPONSES*, e adicione as respostas desejadas;
3. Após o preenchimento das respostas, clique em *Save*.

O processo de criação de intenção precisa ser feito para todo as intenções do fluxo apresentando na Figura 1.3.

1.6.5. Contextos

Esta seção explica como adicionar contextos nas intenções do fluxo para acontecer o encadeamento da conversa. Primeiramente, siga os seguintes passos para adicionar contextos a uma intenção, conforme ilustrado na Figura 1.9.

1. Selecione uma *intent*;
2. Selecione a opção *Contexts*;
3. Clique na opção *ADD CONTEXT*;
4. No campo *Add input context*, adicione o contexto de entrada;
5. No campo *Add output context*, adicione o contexto de saída;

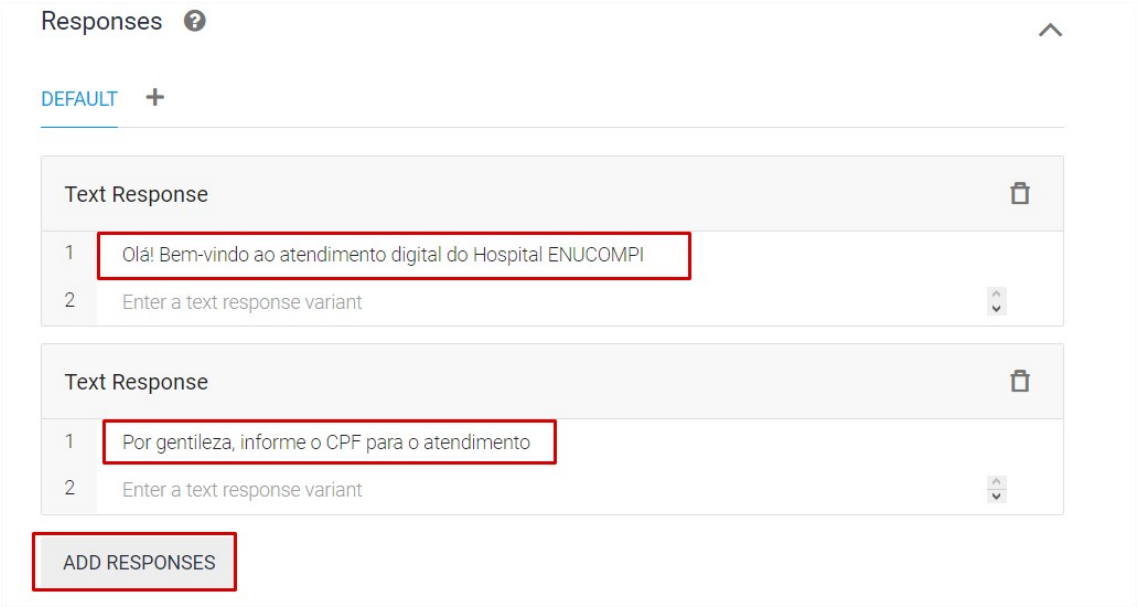


Figura 1.8. Adicionando respostas às intenções no Dialogflow.

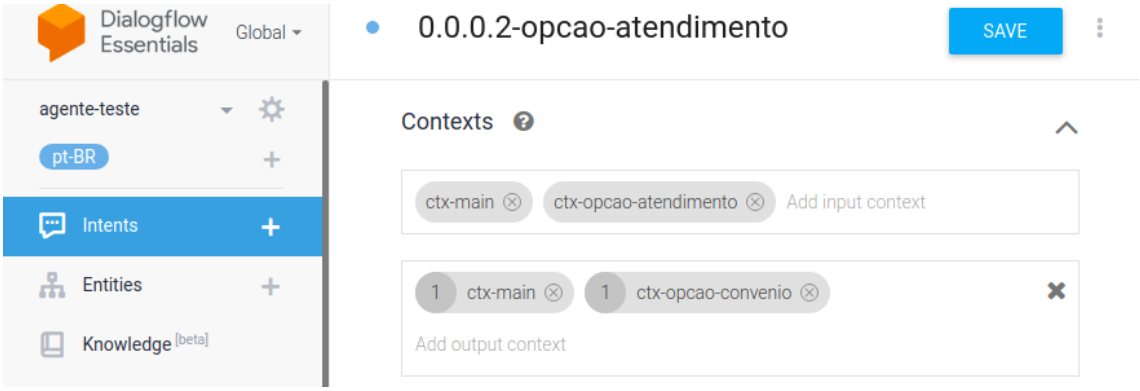


Figura 1.9. Adicionado contextos a uma intenção no Dialogflow.

6. Clique no botão “Salvar”.

Por fim, siga os passos a seguir para realizar o encadeamento de conversa, considerando as Figuras 1.10 e 1.11).

1. Selecione a *intent* “0.0.0.0-introducao”.
   (a) Selecione a opção *Contexts*;
   (b) No campo *Add output context*, adicione os contextos de saída “ctx-introducao” e “ctx-agenda-consulta”;
   (c) Clique no botão “Salvar”.
2. Selecione a *intent* “0.0.0.1-agenda-consulta”.

(a) No campo *Add input context*, adicione os contextos de entrada “ctx-introducao” e “ctx-agenda-consulta”;

(b) No campo *Add output context*, adicione os contextos de saída “ctx-introducao” e “ctx-opcao-atendimento”.

(c) Clique no botão “Salvar”.

O processo de encadeamento de conversa precisa ser configurado para todas as intenções do fluxo.

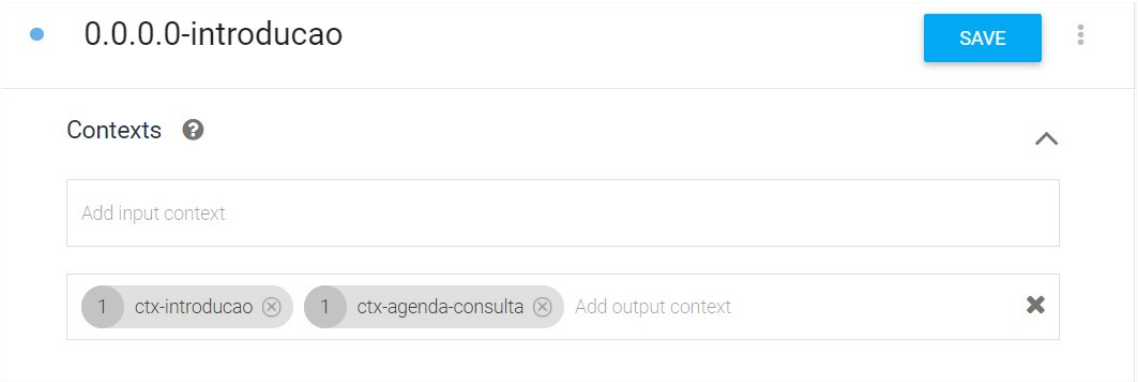


Figura 1.10. Encadeamento de conversa no Dialogflow.

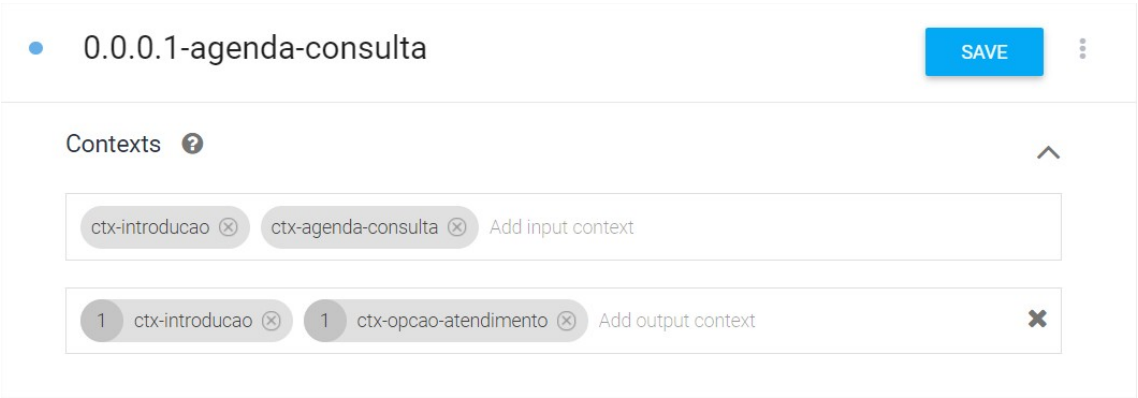


Figura 1.11. Encadeamento de conversa (agendamento).

1.6.6. Testando o ChatBot

Para realizar o teste do ChatBot desenvolvido, siga os passos a seguir, levando em consideração a Figura 1.12.

1. Clique na campo *Try it now*;
2. Digite uma frase de entrada, por exemplo “Oi” e aperte a tecla “Enter”;
3. Como resultado, será apresentado a frase de resposta cadastrada na intenção “0.0.0.0-introducao”(Figura 1.13).

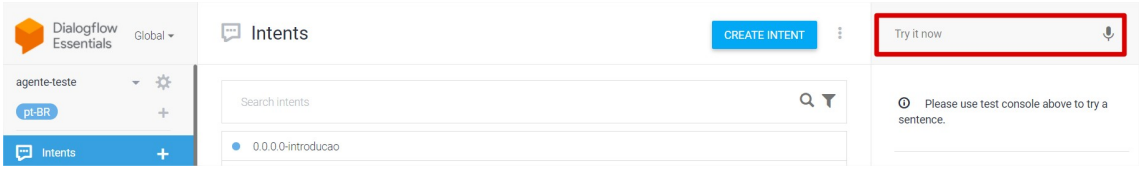


Figura 1.12. Teste do agente no Dialogflow.

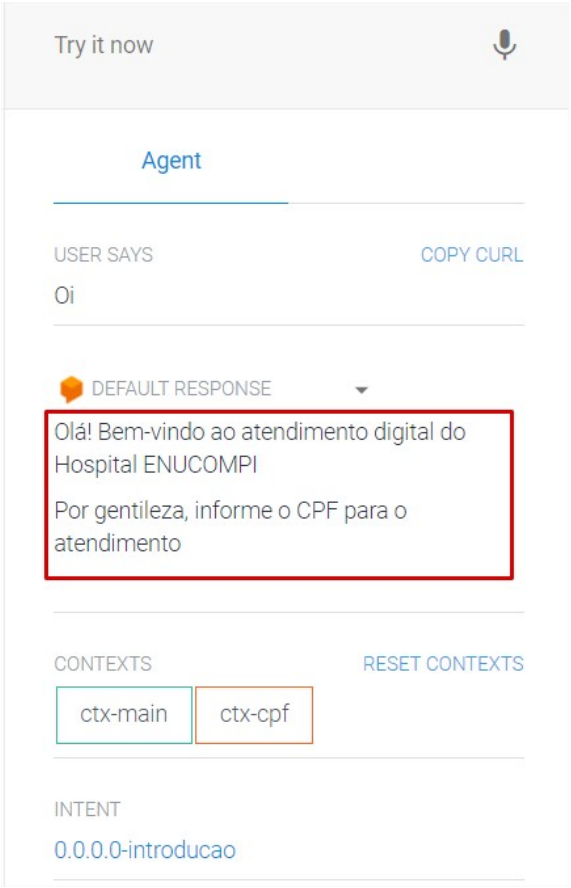


Figura 1.13. Teste do agente (resultado).

Caso deseje, é possível conectar o ChatBot com a plataforma Telegram. A seguir são apresentados os passos para isso.

1. Primeiro será necessário criar um Bot no "Telegram", o processo de criação do Bot pode ser consultado na documentação do "Telegram"[5];
2. Copie o token gerado ao criar o Bot no "Telegram"(Figura 1.14);
3. No Dialogflow, clique no menu *Integrations*;
4. Selecione a opção "Telegram";

[5]https://core.telegram.org/bots#3-how-do-i-create-a-bot

5. Cole o token no campo “Telegram token” (Figura 1.15);

6. Por fim, clique no botão *START*.

Após seguir todo o passo a passo, o ChatBot está pronto para ser utilizado, como ilustrado na Figura 1.16.

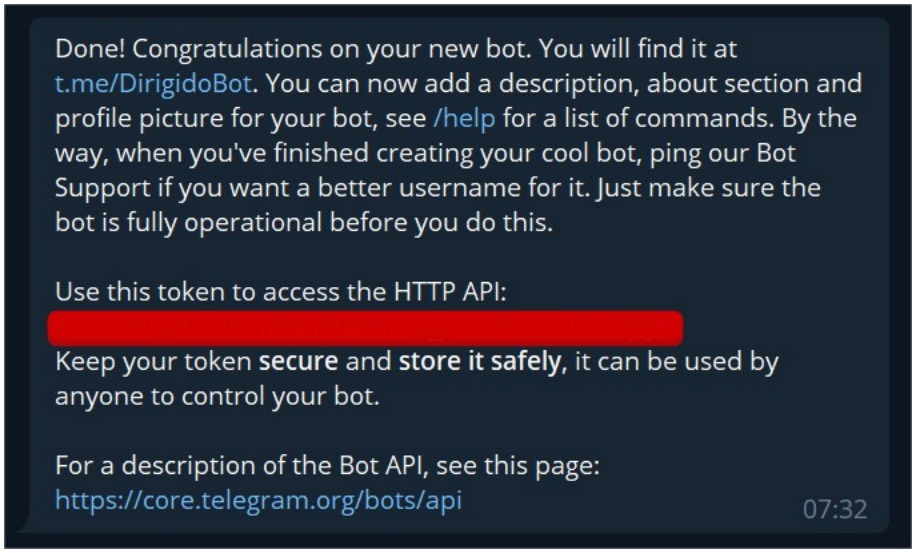


Figura 1.14. Token Telegram.

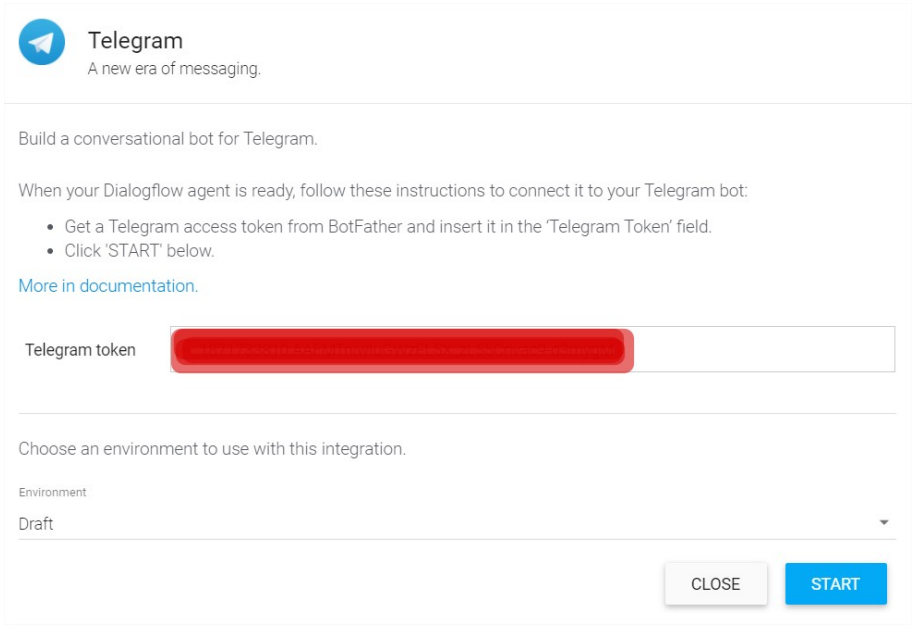


Figura 1.15. Integração com o Telegram no Dialogflow.

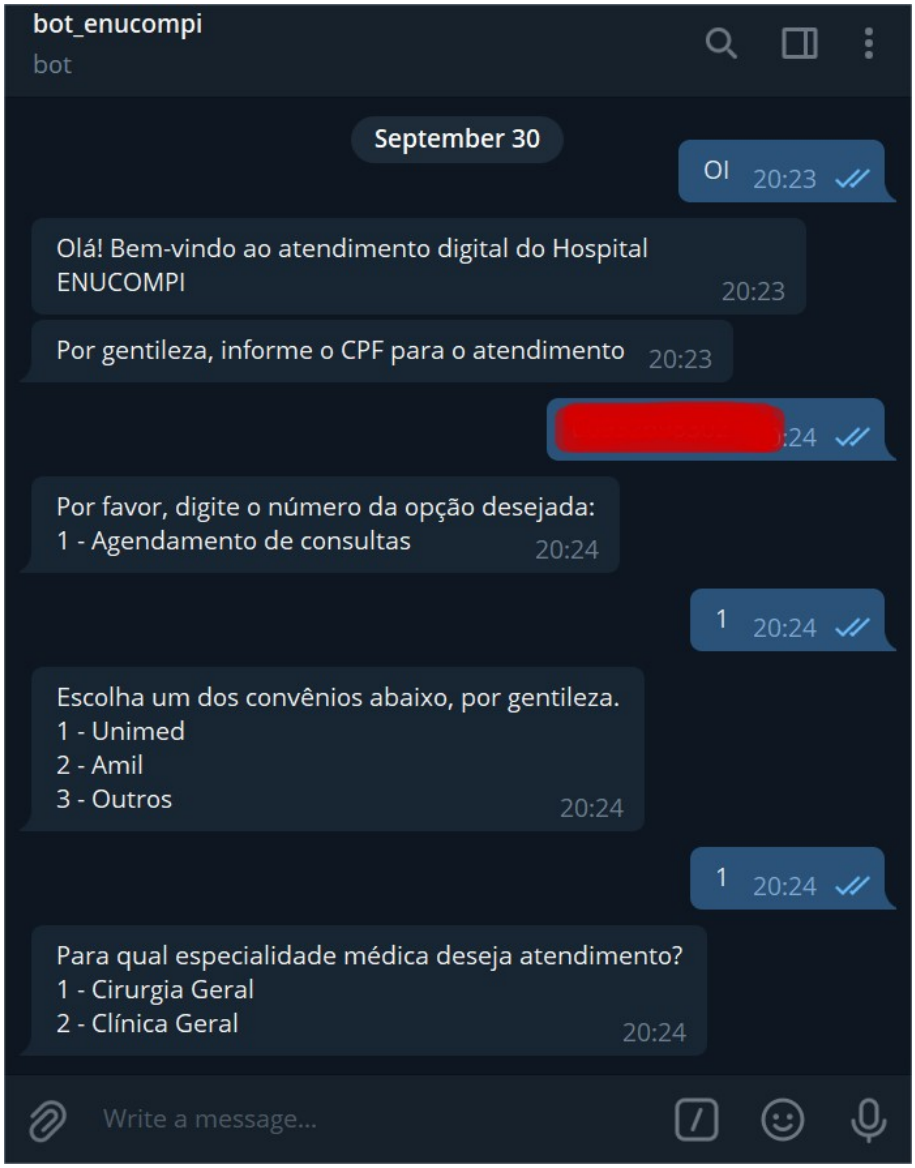


Figura 1.16. Demostração de conversa com o ChatBot criado.

1.7. Considerações Finais

Os ChatBots não são novos no mercado, e suas tecnologias e formas de implementação estão em constante evolução. Isso ocorre na medida em que os avanços nas áreas de processamento de linguagem natural e aprendizado de máquina evoluem. Este capítulo mostrou que as técnicas de PLN e AM disponíveis na plataforma Dialogflow se mostraram eficaz e fáceis de utilização, até mesmo por desenvolvedores não experientes.

No capítulo, foram apresentados os conceitos de ChatBots e as tecnologias utilizadas em seu desenvolvimento. O capítulo deu foco em mostrar como desenvolver um ChatBot para realizar agendamento de consultas médicas com base na plataforma Dialogflow, e integrado ao Telegram. Esse ChatBot pode ser melhorado, como, por exemplo, adicionado integrações com serviços externos (e.g., *Google Calendar*, *API Speech-to-Text*) por

meio de *WebHooks*, e utilizando banco de dados para persistir e consultar informações. Além disso, o Dialogflow fornece suporte para várias integrações, tais como Telegram, Skype, Messeger fom Facebook, Slack, Line, Twilio, Twitter, Viber.

Referências

[Agarwala et al. 2019] Agarwala, H., Becker, R., Fatima, M., and Riediger, L. (2019). Development of an artificial conversational entity (ace) for continuous learning and adaptation to user's preferences and behavior. *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*.

[Ali 2020] Ali, N. (2020). Chatbot: A conversational agent employed with named entity recognition model using artificial neural network. *CoRR*, abs/2007.04248.

[Aranha 2017] Aranha, C. N. (2017). Uma abordagem de pré-processamento automático para mineração de textos em português: Sob o enfoque da inteligência computacional.

[Bocklisch et al. 2017] Bocklisch, T., Faulkner, J., Pawlowski, N., and Nichol, A. (2017). Rasa: Open source language understanding and dialogue management. *CoRR*, abs/1712.05181.

[BULEGON and MORO 2010] BULEGON, H. and MORO, C. M. C. (2010). Mineração de texto e o processamento de linguagem natural em sumários de alta hospitalar.

[Csaky 2019] Csaky, R. (2019). Deep learning based chatbot models. *CoRR*, abs/1908.08835.

[Denecke et al. 2021] Denecke, K., Vaaheesan, S., and Arulnathan, A. (2021). A mental health chatbot for regulating emotions (sermo) - concept and usability test. *IEEE Transactions on Emerging Topics in Computing*, 9(3):1170–1182.

[Forgy 1965] Forgy, E. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–780.

[Ghandeharioun et al. 2019] Ghandeharioun, A., McDuff, D., Czerwinski, M., and Rowan, K. (2019). Emma: An emotion-aware wellbeing chatbot. In *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 1–7.

[Gupta et al. 2021] Gupta, J., Singh, V., and Kumar, I. (2021). Florence- a health care chatbot. In *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, volume 1, pages 504–508.

[Kar and Haldar 2016a] Kar, R. and Haldar, R. (2016a). Applying chatbots to the internet of things: Opportunities and architectural elements. *International Journal of Advanced Computer Science and Applications*, 7(11).

[Kar and Haldar 2016b] Kar, R. and Haldar, R. (2016b). Applying chatbots to the internet of things: Opportunities and architectural elements. *CoRR*, abs/1611.03799.

[Karve et al. 2018] Karve, S., Nagmal, A., Papalkar, S., and Deshpande, S. A. (2018). Context sensitive conversational agent using dnn. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 475–478.

[Kraus 2007] Kraus, H. (2007). Protótipo de um chatterbot para área imobiliária integrado a tecnologia de recionínio baseado emcasos.

[Leptourgos and Corlett 2020] Leptourgos, P. and Corlett, P. R. (2020). Embodied predictions, agence, and psychosis.

[Light 1998] Light, M. (1998). *Journal of Logic, Language, and Information*, 7(1):111–114.

[Maldonado and Cuadra 2019a] Maldonado, J. A. V. and Cuadra, J. A. G. (2019a). Natural language interface to database using the dialogflow voice recognition and text conversion api. In *2019 8th International Conference On Software Process Improvement (CIMPS)*, pages 1–10.

[Maldonado and Cuadra 2019b] Maldonado, J. A. V. and Cuadra, J. A. G. (2019b). Natural language interface to database using the dialogflow voice recognition and text conversion api. In *2019 8th International Conference On Software Process Improvement (CIMPS)*, pages 1–10.

[Mathew et al. 2019] Mathew, R. B., Varghese, S., Joy, S. E., and Alex, S. S. (2019). Chatbot for disease prediction and treatment recommendation using machine learning. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 851–856.

[Mellado-Silva et al. 2020] Mellado-Silva, R., Faúndez-Ugalde, A., and Lobos, M. B. (2020). Learning tax regulations through rules-based chatbots using decision trees: a case study at the time of covid-19. In *2020 39th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–8.

[Morais and Ambrósio 2007] Morais, E. A. M. and Ambrósio, A. P. (2007). Mineração de textos.

[Muhammad et al. 2020a] Muhammad, A. F., Susanto, D., Alimudin, A., Adila, F., Assidiqi, M. H., and Nabhan, S. (2020a). Developing english conversation chatbot using dialogflow. In *International Electronics Symposium (IES)*, pages 468–475.

[Muhammad et al. 2020b] Muhammad, A. F., Susanto, D., Alimudin, A., Adila, F., Assidiqi, M. H., and Nabhan, S. (2020b). Developing english conversation chatbot using dialogflow. In *2020 International Electronics Symposium (IES)*, pages 468–475.

[Nguyen and Shcherbakov 2018] Nguyen, T. and Shcherbakov, M. (2018). A neural network based vietnamese chatbot. In *2018 International Conference on System Modeling Advancement in Research Trends (SMART)*, pages 147–149.

[Pacheco and Pereira 2018] Pacheco, C. and Pereira, N. (2018). Deep learning conceitos e utilização nas diversas Áreas do conhecimento. 2:34–49.

[Pérez-Soler et al. 2021] Pérez-Soler, S., Guerra, E., and de Lara, J. (2021). Creating and migrating chatbots with conga. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 37–40.

[Sabharwal et al. 2019] Sabharwal, N., Barua, S., Anand, N., and Aggarwal, P. (2019). *Developing Cognitive Bots Using the IBM Watson Engine*.

[Salvi et al. 2019] Salvi, S., Geetha, V., and Sowmya Kamath, S. (2019). Jamura: A conversational smart home assistant built on telegram and google dialogflow. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pages 1564–1571.

[Santos et al. 2014] Santos, R. E. S., Neto, J. S. C., Souza, E. P. R., Magalhães, C. V. C., and Vilar, G. (2014). Técnicas de processamento de linguagem natural aplicadas ao processamento de mineração de textos: Resultados preliminares de mapeamento sistemático. revista de sistemas e computação.

[Softić et al. 2021] Softić, A., Husić, J. B., Softić, A., and Baraković, S. (2021). Health chatbot: Design, implementation, acceptance and usage motivation. In *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6.

[TURING 1950] TURING, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460.

[von Rueden et al. 2021] von Rueden, L., Mayer, S., Beckh, K., Georgiev, B., Giesselbach, S., Heese, R., Kirsch, B., Walczak, M., Pfrommer, J., Pick, A., Ramamurthy, R., Garcke, J., Bauckhage, C., and Schuecker, J. (2021). Informed machine learning - a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1.

[Weizenbaum 1966] Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45.

Capítulo 2

Detecção de Máscara em Python Usando OpenCV e Deep Learning

Vitória de Carvalho Brito, Patrick Ryan Sales dos Santos e Antonio Oseas de Carvalho Filho

Abstract

In view of the pandemic scenario we are experiencing, we propose a minicourse aimed at the application of Computer Vision techniques that can be useful to support the authorities in the control of restrictive measures. The minicourse aims to explain the concepts of techniques such as Convolutional Neural Network, Multi-task Cascaded Convolutional Networks, FaceNet and Multilayer Perceptron, practically applying such techniques in a Computer Vision system to detect whether people in a given environment are or are not using mask. This chapter describes the technologies used in the mini-course, as well as the steps involved in implementation.

Resumo

Diante do cenário de pandemia ao qual estamos vivenciando, propomos um minicurso voltado à aplicação de técnicas de Visão Computacional que podem ser úteis para apoiar as autoridades no controle das medidas restritivas. O minicurso tem como objetivo explicar os conceitos de técnicas como Rede Neural Convolucional, Multi-task Cascaded Convolutional Networks, FaceNet e Multilayer Perceptron, aplicando de forma prática tais técnicas em um sistema de Visão Computacional para detectar se pessoas em um determinado ambiente estão ou não utilizando máscara. Este capítulo descreve as tecnologias utilizadas no minicurso, bem como os passos envolvidos na implementação.

2.1. Introdução

Desde a descoberta do novo coronavírus (COVID-19) na China no final de 2019, a doença tornou-se uma preocupação mundial, principalmente devido à sua rápida disseminação. Em setembro de 2021, o número de casos confirmados notificados à Organização Mundial

da Saúde (OMS) já ultrapassava 223 milhões, enquanto o número de óbitos já ultrapassava 4 milhões [WHO 2021].

Em virtude do cenário de pandemia, diversas soluções computacionais têm sido propostas para apoiar as autoridades no controle das medidas restritivas, como por exemplo a detecção de máscara em tempo real. Os algoritmos de detecção de máscara podem ser utilizados em sistemas embarcados presentes em câmeras de supermercados, shoppings, aeroportos, hospitais, escolas e outros ambientes que acomodam um grande número de pessoas.

O propósito deste projeto é desenvolver um algoritmo para detecção de máscaras utilizando técnicas de *Deep Learning* para a caracterização e classificação das faces e a biblioteca OpenCV para a captura de vídeo em tempo real. Ao final, espera-se instigar o leitor quanto a utilização dos métodos de detecção facial e proporcionar um projeto que pode agregar o portfólio dos participantes.

2.1.1. Objetivos

Este capítulo visa construir uma aplicação de Visão Computacional e apresentar os conceitos envolvidos em cada etapa do sistema, bem como as tecnologias utilizadas. De maneira específica, pretende-se:

- Abordar, de maneira sucinta, os principais conceitos de Visão Computacional;
- Explanar os conceitos de Redes Neurais Convolucionais;
- Introduzir sobre a *Multi-task Cascaded Convolutional Networks*;
- Introduzir sobre a *FaceNet*;
- Introduzir sobre o classificador *MLP*;
- Aplicar, de maneira prática, os conceitos explanados.

2.1.2. Organização do Capítulo

Este capítulo está organizado da seguinte maneira: a Seção 2.2 apresenta os conceitos das técnicas utilizadas no desenvolvimento da aplicação; a Seção 2.3 detalha os passos envolvidos na execução do método, bem como os algoritmos criados; por fim, a Seção 2.4 mostra as considerações finais do trabalho, destacando as contribuições e sugestões de melhoria.

2.2. Referencial Teórico

Esta seção fornece um relato da literatura sobre os principais temas abordados na proposta deste trabalho, contribuindo para o entendimento das etapas desenvolvidas.

2.2.1. Visão Computacional

Visão computacional é a ciência responsável pela visão de uma máquina, pela forma como um computador enxerga o meio à sua volta, extraindo informações significativas

a partir de imagens capturadas por câmeras de vídeo, sensores, scanners, entre outros dispositivos. Estas informações permitem reconhecer, manipular e pensar sobre os objetos que compõem uma imagem [Ballard e Brown 1982].

O olho humano consegue perceber e interpretar objetos em uma imagem de forma muito rápida. Isso acontece no córtex visual do cérebro, uma das partes mais complexas no sistema de processamento do cérebro. Alguns cientistas concentraram seus estudos na tentativa de entender o funcionamento dessa parte do cérebro, para então trazer tais ideias para a Visão Computacional. É o que pesquisadores do MIT definem como "ensinar computadores a enxergarem como humano"[de Milano e Honorato 2014].

Dessa forma, a visão computacional fornece ao computador uma infinidade de informações precisas a partir de imagens e vídeos, de forma que o computador consiga executar tarefas inteligentes, simulando e aproximando-se da inteligência humana.

Em geral, um sistema de Visão Computacional é composto pelas seguintes etapas:

- **Aquisição:** nesta etapa as imagens são capturadas e representadas de forma computacional para serem interpretadas pela etapa seguinte.
- **Processamento de imagens:** o objetivo deste estágio é adequar e otimizar os dados visuais adquiridos. Para isso, podem ser aplicadas algumas técnicas como retirada de ruídos, rotação da imagem, aplicação de filtros, etc.;
- **Segmentação:** consiste em dividir a imagem em objeto(s) e fundo. Em outras palavras, essa etapa consiste em técnicas que de alguma maneira consigam formar padrões de agrupamento, gerando sub-regiões que possuem entre si alguma similaridade;
- **Extração de características:** tem como objetivo representar, através de valores, uma imagem ou partes dela. Estes valores são características fundamentais que representam propriedades contidas nas imagens;
- **Reconhecimento de padrões:** neste ponto as imagens são classificadas em função de suas características similares.

2.2.2. *Deep Learning*

O conceito de algoritmos de *Deep Learning* foi introduzido no final do século XX, permitindo que modelos computacionais compostos por várias camadas de processamento aprendam representações de dados com vários níveis de abstração. Em contraste com as abordagens tradicionais de *Machine Learning*, as tecnologias de *Deep Learning* estão progredindo recentemente em aplicações para reconhecimento de fala, processamento de linguagem natural, recuperação de informações, visão computacional e análise de imagens [Liu et al. 2017].

As Redes Neurais Convolucionais (*CNNs*, do inglês *Convolutional Neural Networks*) fazem parte de uma categoria de algoritmos de *Deep Learning* que tornou-se o novo padrão na área de Visão Computacional. As *CNNs* são baseadas no processamento de dados visuais, capaz de aplicar filtros nesses dados, mantendo a relação de vizinhança entre os

pixels da imagem ao longo do processamento da rede [LeCun et al. 1998]. Essa operação é conhecida como convolução, onde ocorre a somatória do produto ponto a ponto entre os valores de um filtro e cada posição da vizinhança do pixel de entrada.

As camadas convolucionais são formadas por um conjunto de filtros que são iniciados com um arranjo 3D, muitas das vezes chamado de volume. Cada filtro tem uma dimensão reduzida, porém estende-se por toda a profundidade do volume de entrada. No processo de treinamento da rede, esses filtros são ajustados automaticamente para que possam extrair características relevantes [Karpathy 2014].

O objetivo da camada de *pooling* é reduzir a dimensionalidade do volume de entrada, de modo geral esse progresso ocorre logo após uma camada convolucional, diminuindo consideravelmente o custo computacional da rede e o tempo de processamento. Por consequência, a camada de *pooling* tende a evitar o *overfitting*. No processo de *pooling*, cada valor da nova matriz é referente ao resultado de alguma métrica aplicada a uma região do mapa de convoluções. Diversas métricas podem ser aplicadas no *pooling*, como o máximo valor da região, o mínimo ou a média, dependendo do problema abordado.

A imagem de entrada fornece para a camada convolucional e de *pooling* a possibilidade de extrair as características relevantes acerca dessa determinada imagem. O objetivo da camada totalmente conectada é utilizar essas características para classificar a imagem em um rótulo pré-determinado. As camadas responsáveis pela classificação das características extraídas das camadas convolucionais são exatamente como uma rede artificial convencional [Haykin et al. 2009].

2.2.2.1. Multi-task Cascaded Convolutional Network

A *Multi-task Cascaded Convolutional Network* (*MTCNN*) é um método de detecção facial baseado em *CNN*, proposto por Zhang et al. [Zhang et al. 2016]. Como mostra a Figura 2.1, o modelo divide-se em três estágios capazes de reconhecer faces e locais de referência, como olhos, nariz e boca.

O primeiro estágio é uma *Fully Convolutional Network* (*FCN*). A diferença entre uma *CNN* e uma *FCN* é que a *FCN* não usa a camada densa como parte de sua arquitetura. A *FCN* usada é denominada *Proposal Network* (*P-Net*), ela é responsável por obter janelas candidatas e seus vetores de regressão de caixas delimitadoras. A regressão de caixa delimitadora é uma técnica popular para prever a localização de regiões quando o objetivo é detectar um objeto de alguma classe predefinida, neste caso faces. Depois de obter os vetores da caixa delimitadora, algum refinamento é feito para combinar regiões sobrepostas. A saída final desse estágio são todas as janelas candidatas após o refinamento para reduzir o volume de candidatas [Gradilla 2020].

Todas as regiões candidatas da *P-Net* são alimentadas na *Refine Network* (*R-Net*). Esta rede é uma *CNN*, não uma *FCN* como a anterior, pois há uma camada densa no último estágio da arquitetura da rede. A *R-Net* reduz ainda mais o número de regiões candidatas, realiza a calibração com a regressão das caixas delimitadoras e emprega uma supressão não máxima para mesclar as caixas candidatas sobrepostas. A saída da *R-Net* prevê se a entrada é uma face ou não, um vetor de 4 elementos (caixa delimitadora para a

face) e um vetor de 10 elementos para a localização do ponto de referência facial.

Por fim, a *MTCNN* possui a *Output Network* (*O-Net*) para operar o terceiro estágio. Este estágio é semelhante ao da *R-Net*, mas a *O-Net* visa descrever o rosto com mais detalhes e produzir as cinco posições dos marcos faciais para olhos, nariz e boca. De acordo com Edwin et. al. [Jose et al. 2019], a *MTCNN* supera de forma consistente os métodos convencionais sofisticados em relação à confiabilidade do desempenho em tempo real. Este desempenho em tempo real é de grande importância em um sistema de vigilância.

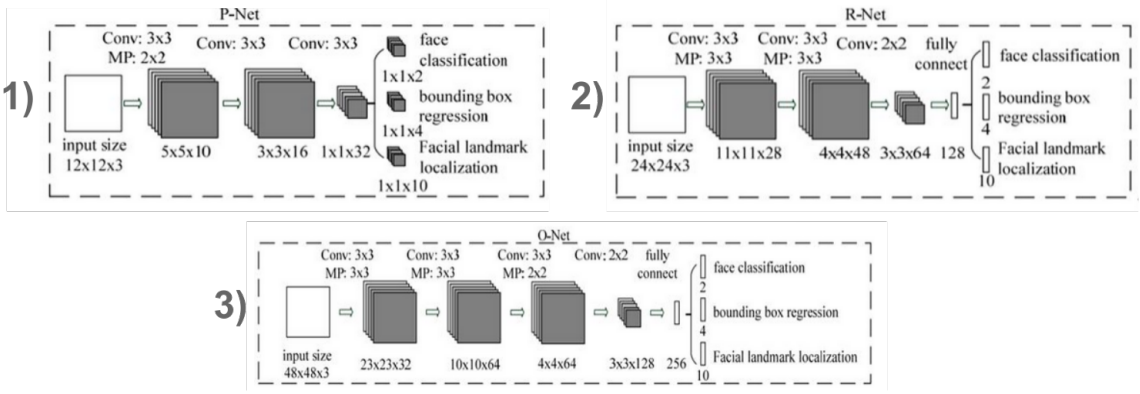


Figura 2.1: Arquitetura da *MTCNN*. Fonte: [Zhang et al. 2016]

2.2.2.2. *FaceNet*

A *FaceNet* é um modelo de reconhecimento facial proposto por Florian Schroff, et al. [Schroff et al. 2015], no Google. A *FaceNet* aprende diretamente um mapeamento a partir de imagens de rosto para um espaço euclidiano compacto onde as distâncias correspondem diretamente a uma medida de semelhança facial. Uma vez que este espaço foi produzido, tarefas como reconhecimento facial, verificação e agrupamento podem ser facilmente implementadas usando técnicas padrão com *embeddings FaceNet* como vetores de características.

O modelo é uma *CNN* profunda treinada por meio de uma função *triplet loss*. Esse treinamento é feito de forma que a distância quadrada L2 entre os *embeddings* corresponda à semelhança de faces: faces da mesma pessoa têm pequenas distâncias e faces de pessoas distintas têm grandes distâncias [Schroff et al. 2015]. A *FaceNet* representa cada face através de um vetor de 128 características.

Para calcular a *triplet loss* são necessárias 3 imagens, chamadas de âncora, positiva e negativa. A intuição por trás da função *triplet loss* é que a imagem âncora (imagem de uma pessoa A específica) esteja mais próxima das imagens positivas (todas as imagens da pessoa A) em comparação com as imagens negativas (todas as outras imagens).

Em outras palavras, o objetivo é que as distâncias entre o *embedding* da imagem âncora e o *embedding* das imagens positivas sejam menores em comparação com as distâncias entre o *embedding* da imagem âncora e o *embedding* das imagens negativas. As Figuras 2.2 e 2.3 ilustram o funcionamento da *FaceNet*.

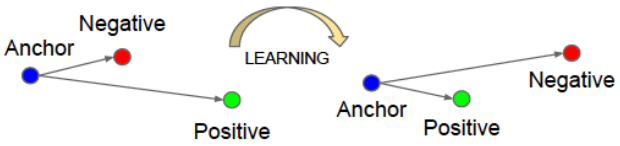


Figura 2.2: Triplet Loss. Fonte: [Kumar 2021].

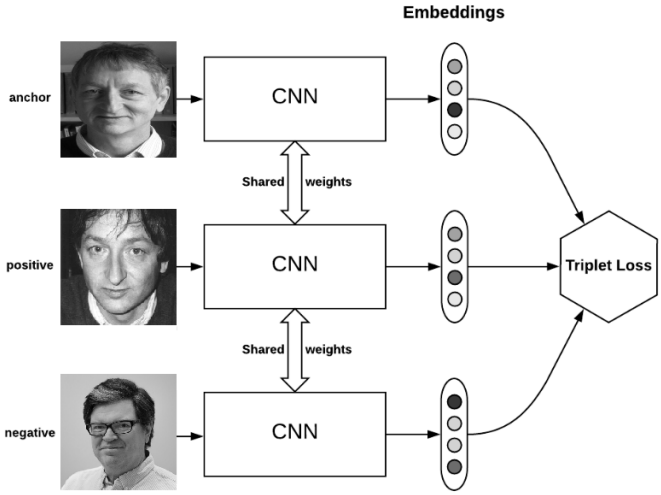


Figura 2.3: Ilustração do funcionamento da *FaceNet*. Fonte: [Kumar 2021].

2.2.2.3. *Multilayer Perceptron*

As Redes Neurais Artificiais tentam modelar o funcionamento do cérebro humano. O cérebro humano, por exemplo, consiste em bilhões de células individuais chamadas neurônios. Dado que o cérebro humano consiste em um grande número de neurônios, a quantidade e a natureza das conexões entre os neurônios são, nos níveis atuais de compreensão, quase impossíveis de avaliar [Ramchoun et al. 2016].

O *Multilayer Perceptron* (*MLP*) é o modelo mais utilizado em redes neurais usando o algoritmo de treinamento de retropropagação. A definição de arquitetura em redes *MLP* é um ponto muito relevante, pois a falta de conexões pode tornar a rede incapaz de resolver o problema de parâmetros ajustáveis insuficientes, enquanto um excesso de conexões pode causar um sobreajuste dos dados de treinamento [Ramchoun et al. 2017], especialmente quando usamos um grande número de camadas e neurônios.

O processo de aprendizado no *MLP* ocorre pela adaptação das conexões pesos a fim de obter uma diferença mínima entre a saída da rede e a saída desejada. Normalmente, utiliza-se o algoritmo de retropropagação baseada em técnicas de gradiente descendente [Ramchoun et al. 2016].

O *MLP* consiste em pelo menos três camadas de nós: uma camada de entrada, uma camada oculta e uma camada de saída. Exceto para os nós de entrada, cada nó é um neurônio que usa uma função de ativação não linear. Suas múltiplas camadas e ativação não linear distinguem o *MLP* de um perceptron linear. Ele pode distinguir dados que não

são linearmente separáveis [Ramchoun et al. 2016].

2.3. Metodologia

Esta seção descreve os passos envolvidos na construção da aplicação. Cada etapa será explicada e os algoritmos de implementação serão mostrados. Toda a implementação deste projeto está disponível no GitHub, bem como a base de imagens e as instruções para executar os códigos. A Figura 2.4 ilustra as etapas desenvolvidas.

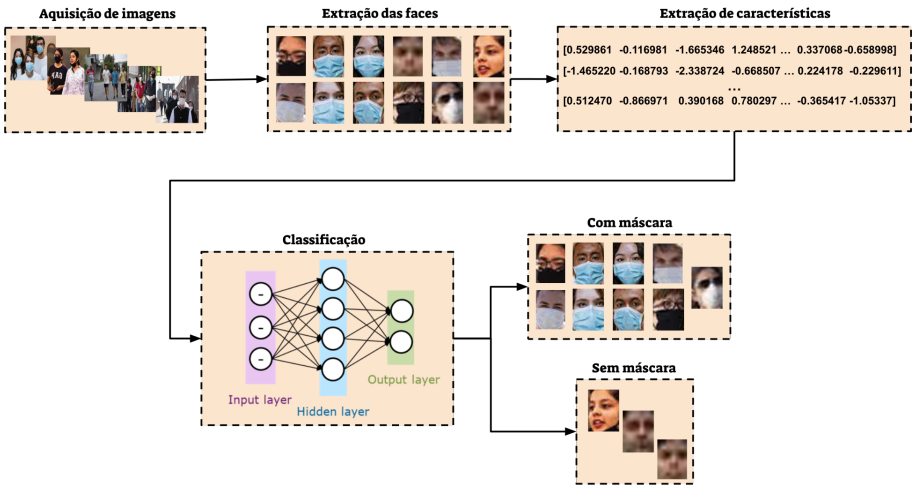


Figura 2.4: Fluxo da aplicação.

2.3.1. Aquisição de Imagens e Pré-processamento

A base de imagens utilizada no projeto foi adquirida, em sua maioria, através do material do professor Sandeco, em seu vídeo sobre detecção de máscaras. A base do professor Sandeco possui 898 imagens de pessoas com máscara e 958 de pessoas sem máscara. Para melhorar essa base, retiramos algumas imagens que não eram necessárias, como por exemplo imagens de pessoas em desenho/animação/não reais. Além disso, adicionamos algumas imagens de pessoas com óculos e máscaras, já que haviam poucas imagens desse tipo na base.

Ao final deste processamento, obtivemos um conjunto de dados que resultou em 1087 imagens de pessoas com máscara e 1000 de pessoas sem máscara, um conjunto relativamente balanceado. Tanto as imagens da base original, quanto da base resultante foram extraídas da internet através da pesquisa de imagens por palavras-chave relacionadas ao tema e métodos para download das imagens em lote. Após a aquisição, as imagens passaram por uma etapa de pré-processamento, onde as faces foram extraídas e redimensionadas para o tamanho 160x160, uma vez que este é o formato de entrada da *FaceNet*.

Após as importações necessárias para os códigos, podemos carregar a base de imagens como no exemplo abaixo.

```
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, cohen_kappa_score,
    confusion_matrix
```

```
from sklearn.neural_network import MLPClassifier
from skimage.io import imread, imshow
import numpy as np
from glob import glob
from tqdm.notebook import tqdm
import pandas as pd
import pickle

maskon = glob('./new_dataset/maskon/*.png')
maskoff = glob('./new_dataset/maskoff/*.png')

print(len(maskon), len(maskoff))
```

Código Fonte 2.1: Importações necessárias e carregamento das imagens.

2.3.2. Extração de Características

A extração de características das faces foi feita por meio do modelo *FaceNet*, que gera um vetor de 128 características para cada entrada, que chamamos de *embeddings*. Neste projeto, usamos o modelo pré-treinado da *FaceNet*, em sua versão implementada no keras. Nós disponibilizamos esse modelo no repositório do projeto no GitHub.

É necessário carregar o modelo treinado e em seguida extrair os *embeddings* de cada imagem através da *FaceNet*.

```
model = keras.models.load_model('./facenet_keras.h5')

data = maskon + maskoff

embeddings = []

for path in tqdm(data):

  try:
    # Lendo e normalizando a imagem
    img = imread(path).astype('float32')/255

    # Aplicando um reshape na imagem para deixar o formato de acordo
    com o input da FaceNet
    input = np.expand_dims(img, axis=0)

    # Extraindo o vetor de embeddings
    embeddings.append(model.predict(input)[0])

  except:
    print(f'Error in {path}')
    continue

labels = pd.DataFrame({
    'label': [1]*len(maskon) + [0]*len(maskoff)
})

df_embeddings = pd.concat([pd.DataFrame(embeddings), labels], axis=1)
```

Código Fonte 2.2: Extração de características com a *FaceNet*.

2.3.3. Classificação

Para classificar os *embeddings* extraídos na etapa anterior, utilizamos o classificador *MLP*. Utilizamos o *MLP* disponível na biblioteca sklearn, que possui vários algoritmos de regressão, classificação e agrupamento. Dividimos o conjunto de dados em treino e teste, com uma proporção de 80% e 20%, respectivamente.

Primeiro separamos o conjunto em treino e teste, em seguida instanciamos, treinamos e testamos o *MLP*.

```
X = np.array(df_embeddings.drop('label', axis=1))
y = np.array(df_embeddings['label'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=0)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

mlp_model = MLPClassifier()

mlp_model.fit(X_train, y_train)

y_pred = mlp_model.predict(X_test)

print('Acurácia: ', accuracy_score(y_test, y_pred))
print('Kappa: ', cohen_kappa_score(y_test, y_pred))
print('Matriz de confusão:\n', confusion_matrix(y_test, y_pred))
```

Código Fonte 2.3: Classificação dos *embeddings* usando o *MLP*.

A acurácia mede a proporção de acerto do modelo, sem considerar o desbalanceamento de classes. Por este motivo, optamos por avaliar o método através do índice kappa também, que leva em consideração o desbalanceamento, aumentando a confiabilidade dos resultados.

Nossos testes mostraram acurácia de 0,9569 e kappa de 0,9135, mas esses valores podem variar de acordo com a aleatoriedade da divisão dos dados ou pela variação de parâmetros do classificador. Para finalizar a parte 1 deste projeto, treinamos o modelo novamente antes de salvá-lo, mas agora usando todos os dados para treino. Quanto mais exemplos de entrada nosso modelo recebe, melhor será seu aprendizado.

```
# Instanciando novamente
mlp_model = MLPClassifier()

# Treinando com todos os dados
mlp_model.fit(X, y)

# Mostrando a acurácia de treino
print(mlp_model.score(X, y))

# Salvando o modelo como um arquivo pickle
pickle.dump(mlp_model, open('./mlp_model.pkl', 'wb'))
```

Código Fonte 2.4: Treinando e salvando o modelo final de classificação. usando o *MLP*.

2.3.4. Detecção em Tempo Real

Com a execução de todas etapas anteriores, teremos um modelo de classificação treinado e pronto para ser utilizado em um cenário real. Para isso, utilizamos a biblioteca OpenCV, uma biblioteca de funções de programação voltada principalmente para a Visão Computacional em tempo real, exatamente o que precisamos.

No código abaixo, temos as importações e o carregamento dos modelos necessários. Apesar da *MTCNN* ter sido utilizada na aquisição de imagens, para extrair as faces de imagens que não estavam na base original, aqui ela é essencial. A *MTCNN* nos fornece um conjunto de coordenadas para cada face detectada, referentes à posição do olho esquerdo, olho direito, nariz, canto esquerdo da boca e canto direito da boca.

```
import pickle
from PIL import Image
from mtcnn import MTCNN
from tensorflow import keras
import cv2 as cv
import numpy as np

# Carregando modelo da FaceNet
facenet_model = keras.models.load_model('./facenet_keras.h5')

# Carregando modelo da MLP
mlp_model = pickle.load(open('./mlp_model.pkl', 'rb'))

# Carregando modelo da MTCNN
detector = MTCNN()
```

Código Fonte 2.5: Importações e carregamento dos modelos necessários.

Basicamente, a OpenCV nos fornecerá a imagem da webcam, a *MTCNN* detectará as faces existentes nessa imagem, a *FaceNet* extrairá as características das faces detectadas e o *MLP* classificará essas características quanto ao uso de máscara.

Abaixo, demos um nome para cada label numérico, pois será esse nome que nós utilizaremos para exibir em cima do retângulo da face, através da OpenCV. Também demos uma cor para cada label, lembrando que o modelo de cor padrão da OpenCV é BGR (blue, green e red), não RGB.

```
label_description = {
    0: 'NO MASK',
    1: 'MASK'
}

color = {
    0: (0, 0, 255),
    1: (0, 255, 0)
}
```

Código Fonte 2.6: Definindo descrição e cor dos labels.

Em seguida, criamos alguns métodos para facilitar a utilização dos modelos e a manipulação dos frames. O primeiro método chamamos de *get_faces*, ele é responsável

por extrair as faces de uma imagem através do modelo *MTCNN* e retorná-las como uma lista de imagens (faces) do tipo Image (Pillow). Como parâmetros dessa função, temos *image*, uma imagem de qualquer dimensão, e *size*, que refere-se à dimensão que as faces recebem antes de serem retornadas, o padrão é (160, 160).

```
def get_faces(image, size=(160, 160)):

    # Transformando imagem em um array numpy
    img = np.asarray(image)

    # Capturando as faces da imagem através da MTCNN
    results = detector.detect_faces(np.asarray(image))

    # Lista para armazenar as faces
    faces = []

    # Percorrendo a lista de faces detectadas
    for i in range(len(results)):

      try:

        # Caso a face tenha sido detectada com mais de 95% de certeza, essa condição é verdadeira
        if results[i]['confidence'] > 0.95:

          # Extraindo os pontos da face
          # w -> width (largura)
          # h -> height (altura)
          x1, y1, w, h = results[i]['box']
          x2, y2 = x1 + w, y1 + h

          # Extraindo a face da imagem fazendo slice nos pontos identificados pela MTCNN
          face = image[y1:y2, x1:x2]

          # Adicionando a face encontrada e suas informações na lista de faces
          # Cada face é redimensionada para o formato especificado na variável size
          faces.append({
              'x1': x1,
              'y1': y1,
              'x2': x2,
              'y2': y2,
              'face': np.array(Image.fromarray(face).resize(size)),
              'confidence': results[i]['confidence']
          })

      except:
        continue

    return faces
```

Código Fonte 2.7: Método responsável por extrair as faces de uma imagem através do modelo *MTCNN*.

Um ponto importante sobre o código acima é que retornamos uma face apenas se o modelo tiver mais de 95% de certeza de que aqueles pontos realmente são de uma face. Isso melhorou consideravelmente a precisão do método, pois antes o modelo identificava como faces algumas regiões sem relação alguma com o alvo.

O método acima retorna as faces encontradas em uma imagem. Agora precisamos de um método que retorne o vetor de *embeddings* da *FaceNet* para um rosto. Chamamos esse método de *get_embeddings* e atribuímos a ele os parâmetros *face*, que é uma imagem com dimensão 160x160 (referente a uma face), e *facenet_model*, uma instância do modelo *FaceNet* previamente treinado. O retorno é um array numpy com 128 posições, referentes às 128 características da *FaceNet* (*embeddings*).

```
def get_embeddings(face, facenet_model):

  # Convertendo a imagem para numpy e normalizando para o intervalo
    [0..1]
  img = np.array(face).astype('float32')/255

  # Expandindo a dimensão da imagem para adequá-la a entrada da FaceNet
  # O shape deve ficar (1, 160, 160)
  input = np.expand_dims(img, axis=0)

  # Fazendo a predição do modelo para extrair os embeddings da imagem
  embedding = facenet_model.predict(input)[0]

  return embedding
```

Código Fonte 2.8: Método responsável por extrair as características de uma face, usando a *FaceNet*.

Agora precisamos de um método para receber um vetor de características (*embeddings*) e retornar a classificação desses dados quanto ao uso da máscara, utilizando o classificador treinado. Chamamos esse método de *get_label*, passando os parâmetros *embedding*, um vetor de características de uma face, representado por um array numpy com 128 valores, e *mlp_model*, o modelo treinado do classificador *MLP*. O retorno é apenas uma variável label, que apresenta valor 0 (sem máscara) ou 1 (com máscara).

```
def get_label(embedding, mlp_model):

    # Expandindo dimensão do array para adequá-lo a entrada do
    classificador
    # A dimensão deve ficar (1, 128)
    embedding = np.expand_dims(embedding, axis=0)

    # Realizando a predição da probabilidade de acerto para cada classe
    proba = mlp_model.predict_proba(embedding)

    # Extraindo do vetor de probabilidades o label que apresentou o
    maior resultado
    label = np.argmax(proba)

    # Caso o modelo tenha menos de 95% de certeza sobre o maior
    resultado, o label predito receberá o valor inverso
```

```
    # Por exemplo, se o modelo não tiver 95% ou mais de certeza sobre
    classificar como label 1, o retorno será label 0
    if proba[0][label] < 0.95:
        label = abs(label-1)

    return label
```

Código Fonte 2.9: Método responsável por classificar o vetor de características de uma face quanto ao uso de máscara, através do *MLP* treinado.

Analisando o código acima, é possível perceber que também há um limiar de confiança, assim como na detecção de faces. O que fizemos foi uma condição que permite que a saída da classificação seja 1 ou 0 apenas se o modelo tiver mais de 95% de certeza sobre isso. Caso a probabilidade de acerto seja menor que esse limiar, a saída prevista será o contrário do que o modelo previu. Essa condição foi necessária para melhorar a precisão dos resultados, já que algumas vezes o modelo errava a classificação porque tinha uma confiança muito baixa sobre a classe prevista, mas retornava essa previsão por ser a maior entre as probabilidades.

O último método nós chamamos de *mark_points_in_frame*, que recebe um frame (imagem), executa o *get_faces* para extrair as faces dessa imagem, o *get_embeddings* para extrair as características das faces e o *get_label* para classificar as características em 0 (sem máscara) ou 1 (com máscara). O método também faz as marcações no frame através da OpenCV, desenhando um retângulo em torno das faces, colocando o label da face em cima do retângulo e inserindo um contador de pessoas sem máscara.

```
def mark_points_in_frame(frame):

    # Transformando a imagem em array numpy
    img = np.asarray(frame)

    # Extraindo faces da imagem
    faces = get_faces(img)

    # Iniciando contador responsável por marcar quantas pessoas sem má
    scara existem na imagem
    no_mask_count = 0

    # Percorrendo a lista de faces identificadas
    for face in faces:

        # Extraindo os pontos da face
        x1 = face['x1']
        x2 = face['x2']
        y1 = face['y1']
        y2 = face['y2']

        # Extraindo os embeddings da face
        emb = get_embeddings(face['face'], facenet_model)

        # Classificando a face em label 0 (maskon) ou label 1 (maskoff)
        label = get_label(emb, mlp_model)

```

```
        # Caso uma pessoa sem máscara seja identificada, o contador é incrementado
        if not label: no_mask_count += 1

        # Configurando o tipo e o tamanho da fonte para iniciar as marcações na imagem
        font = cv.FONT_HERSHEY_TRIPLEX
        font_scale = 0.5

        # Desenhando um retângulo em torno da face
        img = cv.rectangle(img, (x1, y1), (x2, y2), color[label], 2)

        # Parâmetros do método retangle: imagem, coordenada de início, coordenada de fim, cor e grossura das linhas

        # Escrevendo a classificação MASK ou NO MASK em cima do retângulo desenhado
        # Essa descrição é baseada no label
        cv.putText(img, label_description[label], (x1, y1-10), font, fontScale=font_scale,
                   color=color[label], thickness=1)

        # Escrevendo no topo do frame um informativo sobre quantas pessoas estão sem máscara na imagem
        cv.putText(img, f'People without mask: {no_mask_count}', (15, 15), font, fontScale=0.6,
                   color=(0, 0, 0), thickness=1)

        # Parâmetros do método putText: imagem, string a ser escrita, posição do texto no frame, estilo da fonte, tamanho da fonte, cor e grossura da fonte

    return img
```

Código Fonte 2.10: Método responsável por executar os métodos anteriores e desenhar na imagem a caixa delimitadora de cada face, bem como o label associado.

Finalmente, podemos executar o código que abrirá nossa webcam e realizará o processo de detecção de máscara em tempo real. É normal se os frames estiverem sendo renderizados lentamente, isso depende do poder computacional da sua máquina, pois o custo das predições dos modelos para cada frame influencia no tempo de renderização. Se sua máquina tiver uma GPU, a transmissão será bem mais fluida.

```
print('Iniciando captura...\n')

# Instanciando um objeto VideoCapture, para selecionar sua webcam
# Também é possível renderizar um vídeo pronto, basta você passar o caminho desse arquivo no lugar do parâmetro 0
vid = cv.VideoCapture(0)

print('Captura iniciada!')

# A captura dos frames através da sua webcam ou vídeo será feita até que você pressione a tecla 'q'
while (True):
```

```
    # Lendo a imagem e extraindo o frame
    # O método read() retorna dois resultados. O primeiro diz se a
    captura do frame foi feita com sucesso ou não, enquanto o segundo
    entrega o frame capturado.
    _, frame = vid.read()

    # Fazendo a detecção de máscara e as marcações nas faces
    identificadas no frame
    detec = mark_points_in_frame(frame)

    # Exibindo o frame resultante do método anterior
    cv.imshow('frame', detec)

    # Condição de parada do loop: pressione a tecla 'q'
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

# Fechando o arquivo de vídeo ou dispositivo de captura
vid.release()

# Fechando as janelas abertas
cv.destroyAllWindows()

# Apagando o objeto da memória
del (vid)
```

Código Fonte 2.11: Captura da webcam com a OpenCV e detecção de máscara através dos métodos anteriormente definidos.

A Figura 2.5 mostra exemplos do algoritmo aplicado à imagens com e sem máscara, além de com e sem óculos.

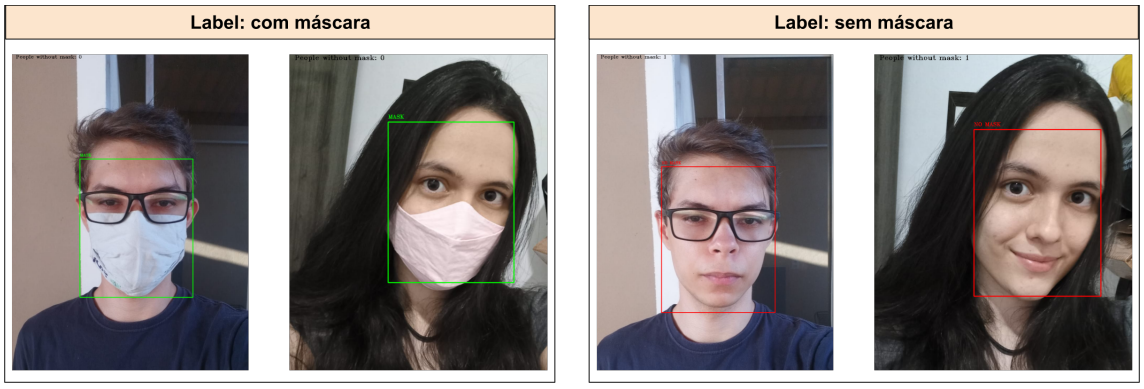


Figura 2.5: Exemplos do método de detecção nos cenários com e sem máscara e com e sem óculos.

2.4. Considerações Finais

Neste capítulo, construímos uma aplicação para detecção de máscaras usando métodos amplamente adotados na literatura para diversas finalidades em Visão Computacional.

No decorrer do capítulo, introduzimos os conceitos de Visão Computacional e *Deep Learning*, adentrando na explicação de cada uma das tecnologias utilizadas na detecção, a saber: *MTCNN*, *FaceNet* e *MLP*.

Embora existam várias melhorias possíveis a serem feitas no projeto para que o mesmo possa integrar um sistema real, os algoritmos de detecção de máscara podem ser utilizados em sistemas embarcados presentes em câmeras de supermercados, shoppings, aeroportos, hospitais, escolas e outros ambientes que acomodam um grande número de pessoas. Além disso, nós apresentamos uma aplicação robusta que pode agregar o portfólio dos alunos e instigar seu interesse quanto a infinidade de aplicações que o ecossistema de Inteligência Artificial nos permite desenvolver.

Destacamos algumas sugestões de melhoria na aplicação:

- Adicionar mais imagens à base de treino, imagens com pessoas em distância maior e imagens capturadas pela própria OpenCV, alternando entre o uso e o não uso de máscara. Com a base de treino incrementada, o classificador deverá ser treinado novamente;
- Utilizar um algoritmo de otimização de hiper parâmetros no classificador, como o Grid Search;
- Variar o limiar de confiança que atribuímos na classificação até encontrar um novo valor ideal;
- Avaliar abordagens alternativas de extração e classificação de características;
- Avaliar abordagens alternativas de detecção facial.

Referências

[Ballard e Brown 1982] Ballard, D. H. e Brown, C. M. (1982). Computer vision. englewood cliffs. *J: Prentice Hall*.

[de Milano e Honorato 2014] de Milano, D. e Honorato, L. B. (2014). Visao computacional.

[Gradilla 2020] Gradilla, R. (2020). Multi-task cascaded convolutional networks (mtcnn) for face detection and facial landmark alignment. Disponível em: https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn-for-face-detection-a Acessado em: 13 de setembro de 2021.

[Haykin et al. 2009] Haykin, S. S., Haykin, S. S., Haykin, S. S., e Haykin, S. S. (2009). *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:.

[Jose et al. 2019] Jose, E., Manikandan, G., T P, M. H., e M H, S. (2019). Face recognition based surveillance system using facenet and mtcnn on jetson tx2.

[Karpathy 2014] Karpathy, A. (2014). Convolutional neural networks. http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/.

[Kumar 2021] Kumar, D. (2021). Introduction to facenet: A unified embedding for face recognition and clustering. Disponível em: https://medium.com/analytics-vidhya/introduction-to-facenet-a-unified-embedding-for-face-recognition-and- Acessado em: 16 de setembro de 2021.

[LeCun et al. 1998] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Liu et al. 2017] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., e Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26.

[Ramchoun et al. 2016] Ramchoun, H., Amine, M., Janati Idrissi, M. A., Ghanou, Y., e Ettaouil, M. (2016). Multilayer perceptron: Architecture optimization and training. *International Journal of Interactive Multimedia and Artificial Inteligence*, 4:26–30.

[Ramchoun et al. 2017] Ramchoun, H., Idrissi, M. A. J., Ghanou, Y., e Ettaouil, M. (2017). Multilayer perceptron: Architecture optimization and training with mixed activation functions. Em *Proceedings of the 2nd International Conference on Big Data, Cloud and Applications*, BDCA'17, New York, NY, USA. Association for Computing Machinery.

[Schroff et al. 2015] Schroff, F., Kalenichenko, D., e Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 815–823.

[WHO 2021] WHO, W. H. O. (2021). Coronavirus disease (covid-19) outbreak situation. Disponível em: https://www.who.int/emergencies/diseases/novel-coronavirus-2019. Acessado em: 12 de setembro de 2021.

[Zhang et al. 2016] Zhang, K., Zhang, Z., Li, Z., e Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503.

Capítulo 3

Emulando Redes de Comunicação para Sistemas de Múltiplos Drones: uma abordagem inicial

Diego S. Pereira, Luís B. P. Nascimento,
Vitor G. Santos, Pablo J. Alsina

Abstract

The popularization of applications that use Unmanned Aerial Vehicles (UAVs) has greatly stimulated research development in this area. Applications with multiple UAVs demand a network capable of enabling communication between the aircraft fleet and the base station on the ground. However, it is not simple to build this type of configuration in real life, making virtual environments an important tool to simulate this scenario with accuracy. In this context, the emulated environment of the Mininet-WiFI integrated with the Robotic Simulation Platform V-REP offers an interesting alternative to represent this scenario. Thus, this mini course is proposed to present how to set up some scenarios for carrying out experiments with multiple drone systems applications.

Resumo

A popularização de aplicações que fazem uso de Veículos Aéreos Não Tripulados (VANTs) tem estimulado o desenvolvimento de pesquisas nessa temática. Aplicações com múltiplos VANTs demandam uma rede capaz de viabilizar a comunicação entre todos os elementos da frota de aeronaves e a estação base em terra. Dessa forma, é preciso fomentar ambientes e estratégias capazes de representar esse cenário de maneira mais próxima da realidade. Nesse contexto, a utilização do ambiente emulado do Mininet-WiFI integrado à Plataforma de Simulação Robótica V-REP apresentam-se com uma alternativa interessante para essa representação. Dessa forma, é proposto um minicurso que visa apresentar essa integração e como montar os primeiros cenários para realização de experimentos com aplicações de sistemas de múltiplos drones.

3.1. Introdução

Os Veículos Aéreos Não Tripulados (VANTs), popularmente conhecidos como drones, vêm sendo utilizados em diversas atividades, tais como inspeção, mapeamento, transporte

de cargas, entre outras [Hong et al. 2021]. Apesar de muitos avanços, existem alguns fatores que limitam a utilização dessas aeronaves em algumas situações, destaca-se a restrição de *payload* (capacidade de transporte de carga útil), tempo de voo (limitada a bateria embarcada) e o alto custo (motores, sensores, atuadores, etc.).

Dessa forma, pesquisas têm investido no uso de múltiplos drones atuando de forma cooperativa para superar tais limitações. Chamados de Sistemas Multi-VANT, eles agregam escalabilidade, flexibilidade, resiliência e autonomia para realização de missões complexas a partir da divisão de tarefas [Fu et al. 2019]. São exemplos interessantes de aplicações que tomam proveito dos benefícios providos por um sistema multi-VANT: o trabalho de [Silva et al. 2019] para realizar varredura grandes regiões em alto mar; o trabalho de [Santos et al. 2019] que tem como objetivo monitorar regiões de proteção ambiental; os autores em [Bai et al. 2021] ofertam serviços de comunicação para usuário em terra; e o trabalho de [Wang et al. 2021] faz uso de um sistema multi-VANT para auxiliar na navegação de grandes embarcações.

Para o funcionamento adequado de um sistema multi-VANT é fundamental a constituição de uma rede de comunicação apta a assegurar a troca de informações entre as aeronaves que integram o sistema. Essa rede de comunicação é denominada Rede *Ad Hoc* Aérea (*Flying Ad Hoc Network* - FANET). Uma FANET é criada de maneira *ad hoc* e deve garantir que todos os nós da rede possam comunicar-se entre si e com a estação base (EB) em terra. Para tal, uma aeronave deve permanecer, obrigatoriamente, no alcance de comunicação de outra e, pelo menos, uma delas necessita ter conectividade com a EB. As FANETs diferem de outros tipos de rede *ad hoc* na manutenção da conectividade, movimentação dos nós, na forma de entrega de dados, descoberta de serviços, entre outras características [Chriki et al. 2019].

O comportamento dinâmico inerente às FANETs, em consequência dos movimentos das aeronaves e da própria natureza da comunicação sem fio, inserem desafios em seu gerenciamento, afinal é imprescindível assegurar um conjunto de requisitos para não inviabilizar a comunicação entre as aeronaves. Nesse cenário, os avanços nos sistemas de telecomunicações, nos protocolos e interfaces de comunicação vêm viabilizando o desenvolvimento das FANETs. Tecnologias importantes providas pelo 4G e 5G, além de avanços com a família IEEE 802.11, são itens importantes para permitir o funcionamento de FANETs. Outras tecnologias como IEEE 802.15.4, DigiMesh e ZigBee também são utilizadas como alternativas viáveis conforme a aplicação na qual a FANET está inserida [Pereira et al. 2020].

Dentro desse contexto, pesquisas relacionadas à temática de múltiplos drones enfrentam alguns desafios para realizar experimentos e avaliar suas proposições, entre eles o alto custo para aquisição de aeronaves, demanda por equipamentos para prover infraestrutura para execução do experimento, além da força de trabalho humano na montagem, durante e finalização dos testes, são exemplos de dificuldades encontradas para experimentação nesse área. Dessa forma, é preciso buscar alternativas que tornem viáveis a execução de ensaios simulados e emulados. Uma opção interessante é a integração do Mininet-WiFi [Fontes et al. 2015] com o CoppeliaSim[1], uma plataforma para simulação de robôs [Rooban et al. 2021].

[1]https://www.coppeliarobotics.com/

A Figura 3.1 apresenta essa integração em funcionamento. É possível ver o terminal (a esquerda) e a representação gráfica dos nós (a direita) do emulador Mininet-WiFi. Enquanto as três aeronaves, do tipo quadrirrotor, no CoppeliaSim (ao centro). Dessa forma, é possível criar cenários de forma rápida e com uso de *drivers* reais, dada a natureza de emulação proposta pelo Mininet-WiFi. Além disso, é possível variar a quantidades de aeronaves e empregar diferentes tipos de tecnologias, permitindo mais flexibilidade ao pesquisador na construção da emulação.

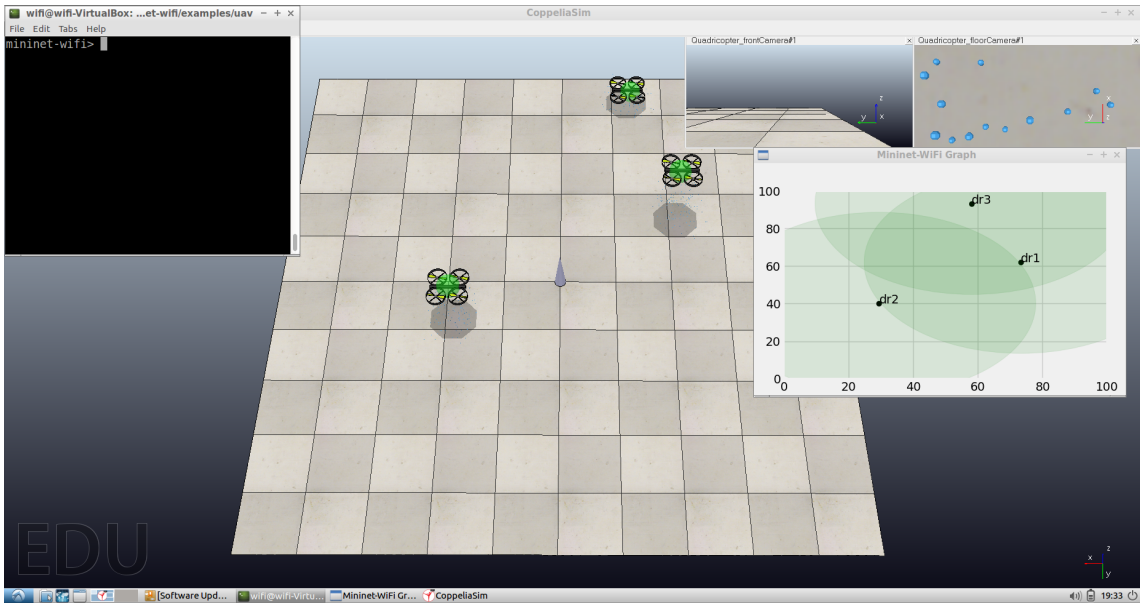


Figura 3.1. Ambiente virtual em execução (Mininet-WiFi e CoppeliaSim).

Portanto, é possível perceber que existem alternativas para investigar as redes de comunicação utilizadas em sistemas de múltiplos drones. Além disso, após a construção do conhecimento básico para criação e utilização dos cenários, agrega-se muito mais valor e motivação a pesquisa, dada a característica visual do CoppeliaSim.

Nessa perspectiva, este capítulo tem como objetivo principal apresentar uma estratégia para emular redes de comunicação para sistemas de múltiplos drones em um ambiente totalmente virtual. Para tal, será feita uma introdução aos principais conceitos referentes a sistemas multi-VANT, além da apresentação de algumas tecnologias de comunicação sem fio utilizadas para viabilizar a comunicação entre as aeronaves desse tipo de sistema. Também serão feitas demonstrações de avaliação de performance com uso de ferramentas abertas(*open source*) a partir de métricas frequentemente utilizadas.

O conteúdo trata-se de uma visão introdutória. Contudo, já permite criar ambientes virtuais customizados para realização de diversas investigações. Ademais, estimular e difundir pesquisas nessa temática, buscando fomentar novos colaboradores. A seguir, será feita uma rápida apresentação sobre redes de comunicação de sistemas de múltiplos drones. Após isso é feito um detalhamento do emulador Mininet-Wifi e da sua integração com o CoppeliaSim. Em seguida, métricas para avaliação de performance de rede são listadas, bem como algumas ferramentas. Por fim, é feito um estudo de caso agregando todo o conteúdo.

3.2. Redes de Comunicação para Sistemas de Múltiplos Drones

Como descrito na introdução do capítulo, um sistema multi-VANT é composto por aeronaves e por, no mínimo, uma estação base. A estação base, normalmente, está em solo e atua como a interface entre o operador e o sistema. Os comados têm como origem a EB e, quando necessário, as aeronaves enviam dados da aplicação para validação do operador. Por exemplo, ao detectar um barco em uma região de navegação proibida, o VANT deve enviar imagens da embarcação para averiguação do operador. Portanto, é necessário garantir a continuidade do enlace de comunicação entre a FANET (composta pelos VANTs) e pela EB.

A Figura 3.2 ilustra o exemplo citado anteriormente. Uma frota de VANTs realizando inspeção de uma região em alto mar. Para tal, as aeronaves são equipadas com uma tecnologia de comunicação sem fio para construção da FANET. Durante toda a operação do sistema multi-VANT ocorre troca de informações entre os VANTs para garantir a execução da missão. Essas informações são oriundas da própria aplicação responsável em controlar o sistema e viabilizar a cooperação entre as aeronaves. Exemplos de funções dessa aplicação são podem ser: manter a comunicação contínua entre as aeronaves e a EB, controlar a posição dos VANTs, ações de recuperação em caso de falhas, entre outras.

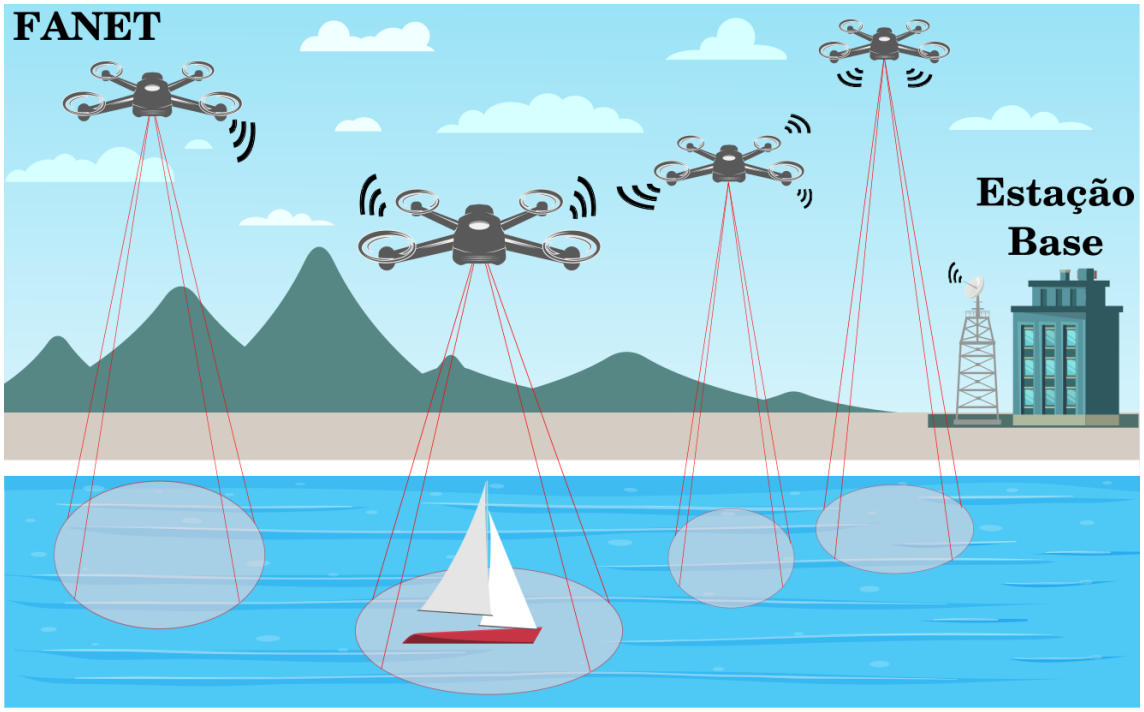


Figura 3.2. Exemplo de sistema multi-VANT executando inspeção em alto mar.

3.3. Emulação de Redes Sem Fio com Mininet-WiFi

A emulação de redes sem fio é uma tarefa árdua e exige um conjunto de softwares integrados que permitam a construção de cenários capazes de reproduzir o comportamento semelhante ao dos dispositivos reais. Nessa perspectiva, o Mininet-WiFi apresenta-

se como uma plataforma para emulação de redes sem fio baseada no módulo mac80211_hwsim nativo do *kernel* do sistema operacional Linux [Fontes et al. 2015]. A seguir, será detalhada a importância da emulação e uma apresentação introdutória do Mininet-WiFi.

3.3.1. Experimentação simulada, emulada ou real?

Existem algumas opções para os pesquisadores testarem suas hipóteses e avaliarem a performance de suas soluções para redes de comunicação de múltiplos drones, as formas mais tradicionais de experimentação são: simulada, emulada e real [Wang et al. 2013]. Todas elas possuem vantagens e desvantagens, portanto, é importante que o pesquisador analise qual a melhor opção para sua pesquisa, ou ainda, quais alternativas ele dispõe para alcançar seus objetivos experimentais.

A experimentação simulada modela as operações e interações entre os dispositivos em um ou mais softwares. Isso permite maior flexibilidade, escalabilidade e controle a um custo muito reduzido. Além de entregar a capacidade de repetição e acessibilidade a outros usuários, que, na maioria dos casos, agregar valor a pesquisa, visto que outros pesquisadores poderão repetir seus experimentos e, também, usar esse ambiente para extrair novos resultados e comparar com anteriores. Contudo, a qualidade dos resultados está relacionada de forma estreita com proximidade entre a modelagem e a realidade, ou seja, quanto mais eficiente a modelagem, mais fidedignos são os resultados obtidos. São exemplos conhecidos de simuladores para redes de computadores o ns-2[2] e ns-3[3].

Diferente da anterior, a experimentação emulada tem como característica principal a utilização de softwares reais, como por exemplo, *kernels* e *drivers*. Isso viabiliza a obtenção de resultados mais próximos da realidade. Outro fator a ser considerado é, na grande maioria dos casos, a portabilidade de código para a experimentação real com a necessidade de poucos ajustes. Um exemplo de software para emulação de redes de computadores é o Mininet[4]. Uma característica interessante é a compatibilidade com ferramentas já desenvolvidas, por exemplo, *iperf* e *ssh*, o que agrega mais possibilidades ou pesquisador na construção e manipulação do cenário. Como na experimentação simulada, também permite maior capacidade de reprodutibilidade e repetibilidade. Destaca-se que soluções emuladas também fazem uso de simulação para conseguir construir seus cenários.

Por fim, a experimentação real é feita com dispositivos de hardware e software reais, normalmente, em um ambiente controlado. Isso implica na construção ou utilização de *testbeds*. A grande vantagem é gerar resultados muito próximos da realidade. Contudo, na maioria dos casos, demanda um custo financeiro elevado, alta complexidade para realização dos experimentos, além da baixa disponibilidade para outros usuários. Logo, as desafios impostos as capacidades de reprodutibilidade e repetibilidade da pesquisa são elevadas. Exemplos de *testbeds* são o Emulab[5] e PlanetLab[6].

[2]http://nsnam.sourceforge.net/wiki/index.php/Main_Page
[3]https://www.nsnam.org/
[4]http://mininet.org/
[5]https://www.emulab.net/portal/frontpage.php
[6]https://planetlab.cs.princeton.edu/

3.3.2. Mininet-WiFi

O Mininet-WiFi foi desenvolvido como uma extensão do Mininet para atuar com redes de comunicação sem fio, sejam elas baseadas no paradigma de Redes Definidas por Software (*Software-Defined Networking* - SDN) ou não. Apesar do termo WiFi fazer referência ao IEEE 802.11 e suas versões(a, b, g, n, ac, etc.), atualmente o emulador suporte diversas tecnologias e protocolos, entre eles IEEE 802.15.4 e LTE. Isso amplia e diversifica a construção de cenários, inclusive viabiliza ambientes de redes heterogêneas. Um exemplo da construção desse tipo de cenário pode ser observado no trabalho de [Selvaraju et al. 2021].

A instalação do Mininet-WiFi é simples, basta utilizar o script *install.sh* disponibilizado no repositório oficial do projeto no GitHub[7]. Para tal, é necessário fazer donwload ou clonar o reposiório para a estação onde deseja realizar a instalação. Além disso, existe a opção de fazer uso da máquina virtual pré-configurada disponível no mesmo repositório. Mais informações sobre instalação e detalhes sobre o Mininet-Wifi podem ser obtidos no livro[8], que possui capítulos gratuitos para comunidade, a página oficial do projeto e publicações que fizeram uso do emulador.

3.3.3. Emulando Redes Sem Fio no Mininet-WiFi

A emulação de cenários básicos de comunicação para redes sem fio no Mininet-WiFi não exige um alto grau de conhecimento prévio. É possível criar cenários através do terminal, com as topologias *single* e *linear*, por exemplo, ou através de scripts *python*, nesse formato é possível uma maior customização. Uma grande variedade de scripts estão disponibilizados no diretório de exemplos e devem ser utilizados como base para cenários mais complexos. A seguir será criado um cenário pelo terminal e alguns comandos serão apresentados. Parte dos comandos são nativos do Mininet-WiFi e outros são ferramentas disponíveis no ambiente Linux.

Prática 01: Criando um cenário básico pelo terminal

OBJETIVO: Criar cenários simples a partir do terminal, aprender alguns comandos do Mininet-WiFi e conhecer algumas ferramentas importantes para monitorar e manipular redes sem fio.

COMANDO: sudo mn –wifi

DESCRIÇÃO: Criar o cenário básico do Mininet-WiFi. Ele é composto por um controlador (c1), um access point(ap1) e duas estações (sta1 e sta2).

```
wifi@wifi-VirtualBox:~$ sudo mn --wifi
*** Adding stations:
sta1 sta2
*** Adding access points:
ap1
```

[7]https://github.com/intrig-unicamp/mininet-wifi

[8]https://mininet-wifi.github.io/book/

```
*** Configuring wifi nodes...
*** Creating network
*** Adding controller
*** Adding hosts:

*** Adding switches:

*** Adding links:
(sta1, ap1) (sta2, ap1)
*** Starting controller(s)
c0
*** Starting L2 nodes
ap1 ...
*** Starting CLI:
mininet-wifi>
```

COMANDO: nodes

DESCRIÇÃO: Listar os nós ativos no cenário em execução.

```
mininet-wifi> nodes
available nodes are:
ap1 c0 sta1 sta2
```

COMANDO: links

DESCRIÇÃO: Listar os links ativos no cenário em execução.

```
mininet-wifi> links
sta1-wlan0<->wifi (OK wifi)
sta2-wlan0<->wifi (OK wifi)
```

COMANDO: iw dev <interface> info

DESCRIÇÃO: Listar informações de uma interface de rede e seu estado. Para mais informações sobre o utilitário *iw* acesse sua documentação[9].

```
mininet-wifi> sta1 iw dev sta1-wlan0 info
Interface sta1-wlan0
        ifindex 46
        wdev 0x1800000002
        addr 02:00:00:00:00:00
        ssid my-ssid
        type managed
        wiphy 24
        channel 1 (2412 MHz), width: 20 MHz (no HT), center1: 2412 MHz
        txpower 14.00 dBm
```

[9] https://wireless.wiki.kernel.org/en/users/documentation/iw

COMANDO: iw dev <interface> link

DESCRIÇÃO: Listar informações do link de comunicação de uma interface de rede, caso esteja conectado.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:02:00 (on sta1-wlan0)
        SSID: my-ssid
        freq: 2412
        RX: 341040 bytes (7802 packets)
        TX: 2353 bytes (27 packets)
        signal: -36 dBm
        tx bitrate: 5.5 MBit/s

        bss flags:        short-slot-time
        dtim period:        2
        beacon int:        100
```

COMANDO: iw dev <interface> scan

DESCRIÇÃO: Listar redes sem fio disponíveis para associação.

```
mininet-wifi> sta1 iw dev sta1-wlan0 scan
BSS 02:00:00:00:02:00(on sta1-wlan0) -- associated
        TSF: 1632158575581557 usec (18890d, 17:22:55)
        freq: 2412
        beacon interval: 100 TUs
        capability: ESS ShortSlotTime (0x0401)
        signal: -36.00 dBm
        last seen: 0 ms ago
        Information elements from Probe Response frame:
        SSID: my-ssid
        Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
        DS Parameter set: channel 1
        ERP: Barker_Preamble_Mode
        Extended supported rates: 24.0 36.0 48.0 54.0
        Supported operating classes:
                * current operating class: 81
        Extended capabilities:
                * Extended Channel Switching
                * SSID List
                * Operating Mode Notification
```

COMANDO: ping <node >

DESCRIÇÃO: Teste de conectividade camada 3. Faz uso do protocolo ICMP. Também calcula a variação entre os tempo de envio e de recebimento de cada pacote. Emite um relatório simples com estatísticas sobre o teste efetuado.

```
mininet-wifi> sta1 ping -c5 sta2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.136 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.142 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.141 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4075ms
rtt min/avg/max/mdev = 0.093/0.127/0.142/0.020 ms
```

COMANDO: iw dev <interface> disconnect

DESCRIÇÃO: Efetuar a desassociação da interface da rede sem fio.

```
mininet-wifi> sta1 iw dev sta1-wlan0 disconnect
mininet-wifi> sta1 iw dev sta1-wlan0 link
Not connected.
mininet-wifi> sta1 ping -c1 sta2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

COMANDO: iw dev <interface> connect <ssid>

DESCRIÇÃO: Efetuar a associação da interface a uma rede sem fio.

```
mininet-wifi> sta1 iw dev sta1-wlan0 connect my-ssid
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:02:00 (on sta1-wlan0)
        SSID: my-ssid
        freq: 2412
        RX: 5915 bytes (134 packets)
        TX: 349 bytes (4 packets)
        signal: -36 dBm
        tx bitrate: 1.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi> sta1 ping -c1 sta2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.350 ms
```

```
--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.350/0.350/0.350/0.000 ms
```

COMANDO: xterm < node >

DESCRIÇÃO: Inicia um terminal emulado para um nó.

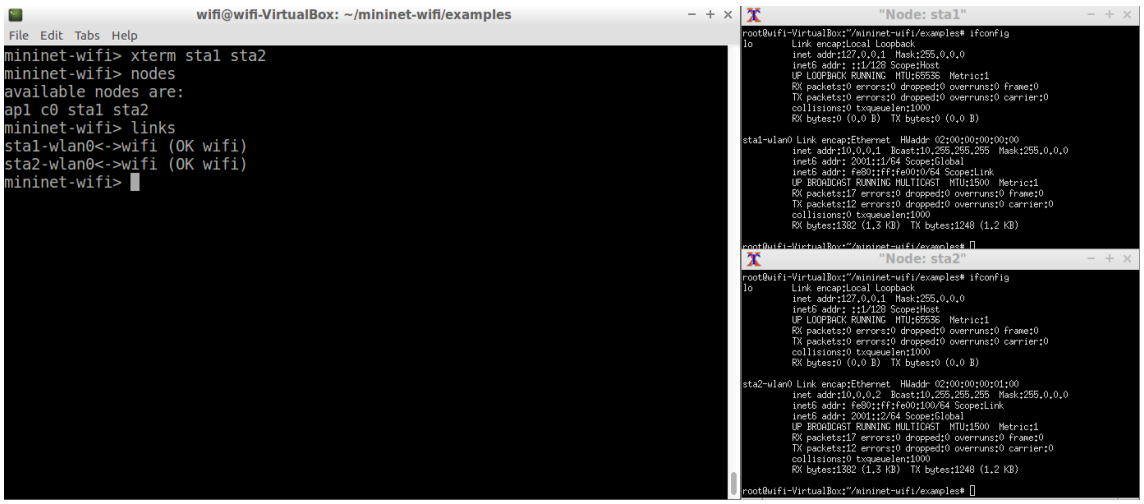


Figura 3.3. Múltiplos terminais com Xterm.

COMANDO: exit

DESCRIÇÃO: Encerra o cenário e o Mininet-WiFi.

```
mininet-wifi> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping switches/access points
ap1
*** Stopping nodes
sta1 sta2

*** Removing WiFi module and Configurations
*** Killing mac80211_hwsim

*** Done
completed in 959.645 seconds
wifi@wifi-VirtualBox:~$
```

Prática 02: Criando um cenário a partir de scripts

OBJETIVO: Criar cenários customizados a partir de scripts python. Considerando as tecnologias utilizadas em FANETs, o foco principal é construir redes ad hoc. Será utilizado como ponto de partida o arquivo *adhoc.py*[10] localizado no diretório *examples* do Mininet-WiFi. Recomenda-se a leitura detalhada do código fonte para melhor entendimento.

COMANDO: sudo python adhoc.py

DESCRIÇÃO: Executar o script adhoc.py.

```
/home/wifi/mininet-wifi/examples$ sudo python adhoc.py
*** Creating nodes
*** Configuring wifi nodes
*** Connecting to wmediumd server /var/run/wmediumd.sock
*** Creating links
*** Starting network
sta1 sta2 sta3
*** Addressing...
*** Running CLI
*** Starting CLI:
mininet-wifi>
```

COMANDO: iw dev <interface> link

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Joined IBSS 02:ca:ff:ee:ba:01 (on sta1-wlan0)
        SSID: adhocNet
        freq: 2432
mininet-wifi> sta2 iw dev sta2-wlan0 link
Joined IBSS 02:ca:ff:ee:ba:01 (on sta2-wlan0)
        SSID: adhocNet
        freq: 2432
mininet-wifi> sta3 iw dev sta3-wlan0 link
Joined IBSS 02:ca:ff:ee:ba:01 (on sta3-wlan0)
        SSID: adhocNet
        freq: 2432
```

EXERCÍCIO 01: Visto que o enlace de comunicação ad hoc está criado, utilize os comandos já apresentados para realizar análises no cenário e verificar se há conectividade entre todos nós.

[10]https://github.com/intrig-unicamp/mininet-wifi/blob/master/examples/adhoc.py

RESPOSTA 01: O cenário é composto por três estações (sta1, sta2 e sta3). Existe o enlace de comunicação ad hoc com ssid *adhocNet*. Ao realizar testes de conectividade entre as três estações percebe-se que a sta1 não consegue enviar/receber pacotes ICMP gerados pelo utilitário *ping* que tenha como destino a sta3.

EXERCÍCIO 02: Feito o diagnóstico, qual a melhor alternativa para viabilizar conectividade entre todos os nós do cenário?

RESPOSTA 02: Para facilitar a busca por soluções, será inserido a linha de código $net.plotGraph(max_x = 100, max_y = 100)$ após a linha $net.configureWifiNodes()$. Para tal, finalize a emulação atual e executa o script *adhoc.py* novamente. Agora será exibido uma representação gráfica do cenário conforme Figura 3.4.

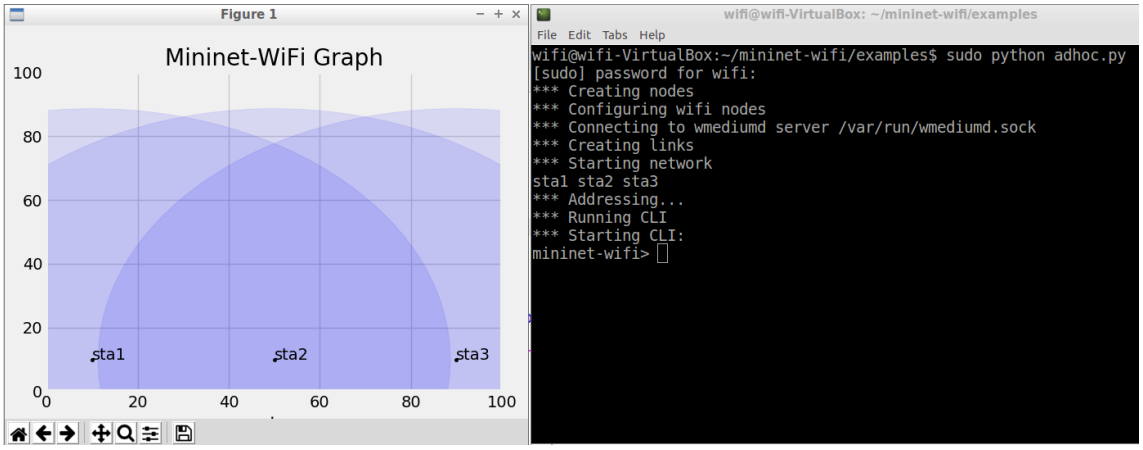


Figura 3.4. Execução do script adhoc.py e visualização do cenário no Mininet-WiFi.

SOLUÇÃO 01: É possível perceber que sta3 não está na área de cobertura da sta1. Apesar de sta2 ter em sua área de cobertura as duas estações, ela não é capaz de viabilizar o encaminhamento de pacotes. Dessa forma, uma alternativa para viabilizar a comunicação entre sta1 e sta3 é alterar a posição de uma das duas estações. A seguir a posição de sta3 será alterada para ingressar na área de cobertura da sta1.

COMANDO: py <node>.setPosition((’x,y,z’)

DESCRIÇÃO: Alterar o posição(x,y,z) de um nó. Para consultar a posição basta usar py <node>.position. É possível encontrar a distância entre dois nós através do comando *distance node1 node2*. Para mais informações sobre comandos no Mininet-WiFI acesse a página oficial no menu comandos[11].

[11] https://mininet-wifi.github.io/commands/

```
mininet-wifi> py sta3.position
[90.0, 10.0, 0.0]
mininet-wifi> py sta3.setPosition('80.0,10.0,0.0')
mininet-wifi> py sta3.position
[80.0, 10.0, 0.0]
mininet-wifi> sta1 ping -c5 sta3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=224 ms

#--- 10.0.0.3 ping statistics ---
5 packets transmitted, 1 received, 80% packet loss, time 4122ms
rtt min/avg/max/mdev = 224.629/224.629/224.629/0.000 ms
```

SOLUÇÃO 02: Conforme observado, sta2 é um elemento intermediário entre sta1 e sta3. Dessa forma, é possível implementar uma solução sem alterar a disposição geográfica dos nós, porém é preciso fazer um de um protocolo de roteamento. O próprio script adhoc.py dispõe de três opções de protocolos, são eles: OLSR, B.A.T.M.A.N e Babel. Não será feito um detalhamento do funcionamento dos protocolos. Durante esse capítulo serão utilizados apenas os dois primeiros.

INSTALAÇÃO: OLSR e B.A.T.M.A.N

DESCRIÇÃO: Para realizar a instalação dos protocolos OLSR e B.A.T.M.A.N, basta utilizar o script *install.sh* dentro do diretório util do Mininet-WiFi utilizando os parâmetro -O e -B, respectivamente.

COMANDO 1: /home/wifi/mininet-wifi$ sudo util/install.sh -O

COMANDO 2: /home/wifi/mininet-wifi$ sudo util/install.sh -B

Após a instalação dos procolos, é necessário apenas executar o script *adhoc.py* com o protocolo desejado como parâmetro. Por exemplo, olsrd ou batman_adv. Por questões de limitação de espaço, só será exibido a execução do OLSR.

```
/home/wifi/mininet-wifi/examples$ sudo python adhoc.py olsrd
*** Creating nodes
*** Configuring wifi nodes
*** Connecting to wmediumd server /var/run/wmediumd.sock
*** Creating links
Starting olsrd in sta1-wlan0...
Starting olsrd in sta2-wlan0...
Starting olsrd in sta3-wlan0...
*** Starting network
```

```
sta1 sta2 sta3
*** Addressing...
*** Running CLI
*** Starting CLI:
mininet-wifi>
```

Executando o novo teste de conectividade.

```
mininet-wifi> sta1 ping -c5 sta3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=63 time=4.28 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=63 time=2.09 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=63 time=2.60 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=63 time=2.71 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=63 time=19.5 ms

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 2.094/6.245/19.535/6.685 ms
```

Observe a seguir a tabela de roteamento da sta1. Para datagramas que tenham como destino o ip 10.0.0.3 deve-se encaminhar para o gateway 10.0.0.2. Portanto, o protocolo OLSR está atualizando a tabela de roteamento da estação.

```
mininet-wifi> sta1 route
Kernel IP routing table
Destination Gateway  Genmask         Flags Metric Iface
10.0.0.0    *        255.0.0.0       U     0      sta1-wlan0
10.0.0.2    10.0.0.2 255.255.255.255 UGH   2      sta1-wlan0
10.0.0.3    10.0.0.2 255.255.255.255 UGH   2      sta1-wlan0
```

3.4. Emulação de Redes de Comunicação para Sistemas de Múltiplos Drones

A construção de um ambiente integrado para emulação de redes de comunicação de sistemas de múltiplos drones é uma demanda crescente entre os pesquisadores. Como elementos motivadores está a explosão de dispositivos capazes de trocar dados, dada a realidade cada vez mais presente da Internet das Coisas (*Internet of Things* - IoT), aliados a popularização dos drones, em especial aqueles com capacidade de realizar missões de forma autônoma. Isso promete revolucionar algumas áreas, entre elas as de logística e de vigilância, por exemplo.

A Figura 3.5 apresenta uma consulta a base de dados Scopus, realizada no dia 23 de setembro de 2021, com a palavra chave ”multi-uav”nos campos título, palavra-chave e resumo. Foi feito um recorte temporal dos últimos dez anos e mostrou o crescimento no número de publicações. Somados, chega-se a um total de 1.414 publicações. Destaque para o ano de 2020 com 304 documentos publicados.

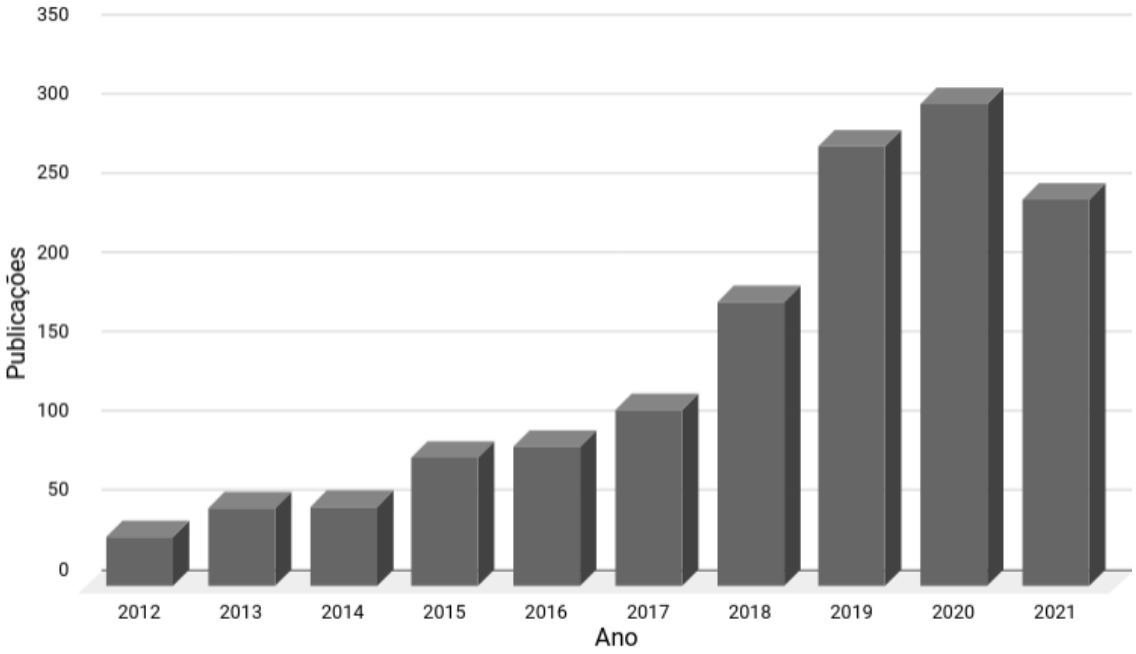


Figura 3.5. Execução do script adhoc.py e visualização do cenário no Mininet-WiFi.

Dentro desse contexto, validar pesquisas nessa temática, independente da estratégia de experimentação, ainda é um grande desafio. Seja pelo custo, hardware e/ou software altamente especializados e caros, ou pela complexidade de integrar diferentes sistemas, tecnologias e até hardware para conseguir um ambiente válido para avaliação de resultados.

Uma alternativa para esse ambiente é a integração do Mininet-WiFi com o simulador robótico CoppeliaSim. Conforme a página oficial do simulador, o CoppeliaSim pode ser usado para desenvolvimento rápido de algoritmos, simulações de automação em fábricas, prototipagem, robótica educacional, monitoramento remoto, verificação dupla de segurança através do gêmeo digital, entre outras. É um simulador de fácil instalação, multiplataforma e disponibiliza uma versão educacional.

3.4.1. Integração Mininet-WiFi e CoppeliaSim

Para realizar a integração entre o Mininet-WiFi e o CoppeliaSim será adotado o sistema operacional Linux. Como informado anteriormente, a página oficial do Mininet-WiFi disponibiliza uma máquia virtual Linux pré-configurada. Para fazer uso dessa máquina é necessário a instalação de um software capaz de executar máquinas virtuais, sugere-se o VirtualBox[12]. Após instalação, basta importar a máquina e iniciá-la.

A integração só é possível devido a disponibilidade de uma API fornecida pelo CoppeliaSim. A API remota permite que aplicações externas ao CoppeliaSim tenham a capacidade de trocas informações com o simulador. Ela é disponibilizada em algumas linguagens de programação, entre elas: C/C++, Java, Python, MatLab, Octave e Lua. Inclusive, Lua é a linguagem de programação nativa do CoppeliaSim. Mais detalhes

[12]https://www.virtualbox.org/

podem ser obtidos no manual do simulador[13].

O suporte a integração entre as plataformas já está disponível no diretório *mininet-wifi/examples/uav/* do Mininet-WiFi. A Figura 3.6 mostra o conteúdo do diretório. Os arquivos sim.py, simConst.py e remoteApi.so compõe a API remota para aplicações python em ambiente Linux. O arquivo simplesTest.py é um exemplo disponibilizado pela CoppeliaSim para verificação do funcionamento da API. O arquivo simulation.ttt é o cenário executado pelo CoppeliaSim, ele contém os drones e todas as informações do ambiente. O arquivo getNodePosition.py captura a posição dos drones no simulador. O arquivo setNodePosition atualiza a posição dos nós no Mininet-WiFI. Por fim, o arquivo uav.py faz a integração entre as plataformas utilizando os arquivos anteriores.

Figura 3.6. Arquivos do diretório *mininet-wifi/examples/uav/*.

Prática 03: Integrando Mininet-WiFI e CoppeliaSim

OBJETIVO: Realizar o setup do ambiente para integração entre o Mininet-WiFi e executar o primeiro cenário utilizando drones do CoopeliaSim. A Figura 3.7 mostra o cenário em execução.

COMANDO: ./examples/uav/install.sh

DESCRIÇÃO: O script install.sh detecta a distribuição Linux, faz o download do CoppeliaSim e descompacta o arquivo no diretório */home/wifi/mininet-wifi/examples/uav/*. De forma opcional, é possível acessar a página oficial do CoppeliaSim, realizar o download conforme a distribuição Linux utilizada e descompactá-lo no diretório *mininet-wifi/examples/uav/*.

COMANDO: sudo python examples/uav/uav.py

DESCRIÇÃO: Executa o script uav.py. O Mininet-WiFi e o Coppelia são iniciados. O cenário é composto por três drones, representados por estações no Mininet-WiFi e quadrirrotores no CoppeliaSim. É feito monitoramento contínuo das posições das aeronaves no CoppeliaSim para serem atualizadas no Mininet-WiFi. Formalmente, os

[13]https://www.coppeliarobotics.com/helpFiles/index.html

software não se comunicam diretamente, apesar da API remota, que é executada no arquivo SimplesText.py.

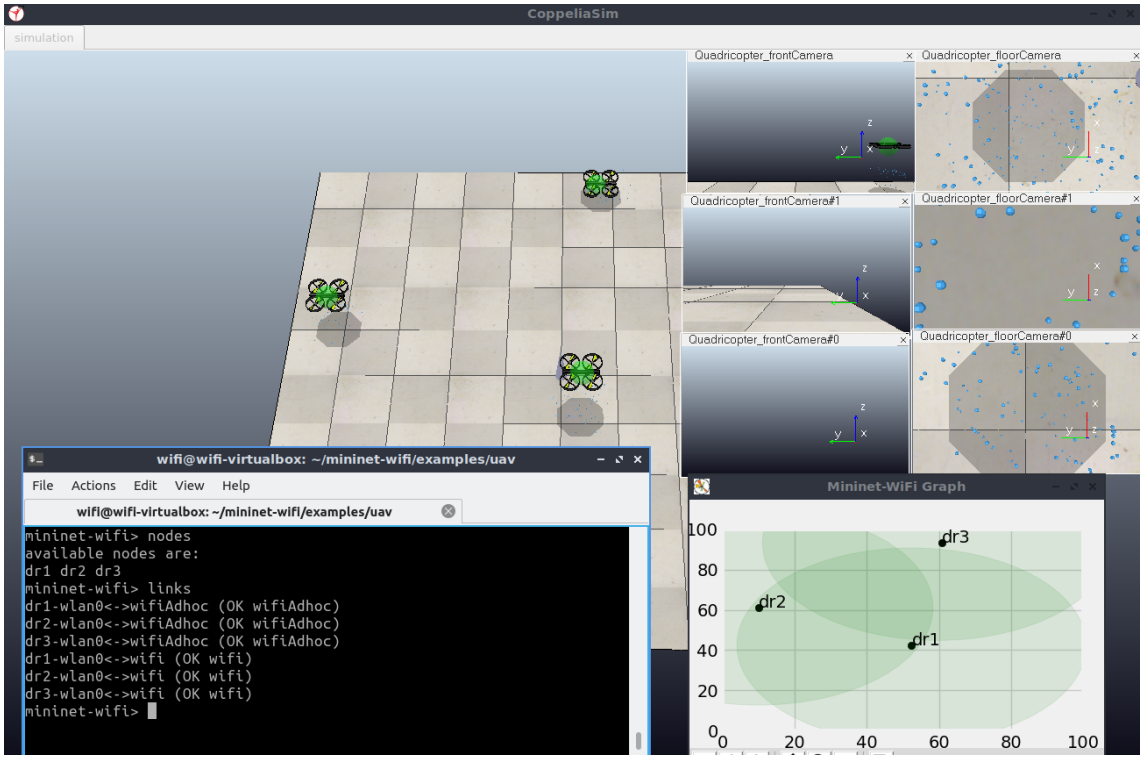


Figura 3.7. Cenário em execução.

Observe os nós ativos e os links. Os nós receberam ”dr”como nome seguido de um número identificador. As demais informações já foram apresentadas nas seções anteriores.

```
mininet-wifi> nodes
available nodes are:
dr1 dr2 dr3
mininet-wifi> links
dr1-wlan0<->wifiAdhoc (OK wifiAdhoc)
dr2-wlan0<->wifiAdhoc (OK wifiAdhoc)
dr3-wlan0<->wifiAdhoc (OK wifiAdhoc)
dr1-wlan0<->wifi (OK wifi)
dr2-wlan0<->wifi (OK wifi)
dr3-wlan0<->wifi (OK wifi)
```

A seguir um teste de conectividade com a ferramente ping entre os drones dr1 e dr2. Destaca-se novamente que toda a emulação relativa a comunicação ocorre no Mininet-WiFi.

```
mininet-wifi> dr1 ping -c5 dr2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
```

```
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=22.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.27 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2.37 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=3.59 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=2.29 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 2.273/6.551/22.230/7.855 ms
```

Devido a movimentação das aeronaves, em algumas momentos da simulação os drones ficam sem conectividade de acordo com a capacidade da área de cobertura provida pela antena embarcada. Logo, garantir que as aeronaves permaneçam sempre na área de outra é um desafio abordado em muitas pesquisas.

```
mininet-wifi> dr1 ping -c5 dr2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4081ms
```

3.4.2. Entendendo o código fonte

A seguir será feito um breve detalhamento de parte do código fonte utilizado na construção do cenário. É importante entender sua estrutura para facilitar a customização de novos cenários.

ARQUIVO: uav.py

DESCRIÇÃO: Inserindo nós do tipo *Station* que serão a representação do drone no ambiente emulado.

```
info("*** Creating nodes\n")
    dr1 = net.addStation('dr1', mac='00:00:00:00:00:01',
                         ip='10.0.0.1/8', position='30,60,0')
    dr2 = net.addStation('dr2', mac='00:00:00:00:00:02',
                         ip='10.0.0.2/8', position='70,30,0')
    dr3 = net.addStation('dr3', mac='00:00:00:00:00:03',
                         ip='10.0.0.3/8', position='10,20,0')
```

DESCRIÇÃO: Inclusão dos links de comunicação do tipo adhoc. No cenário inicial foi utilizado o protocolo de roteamento batman_adv.

```
info("*** Configuring wifi nodes\n")
    net.configureWifiNodes()
```

```
    net.addLink(dr1, cls=adhoc, intf='dr1-wlan0',
                ssid='adhocNet', proto='batman_adv',
                mode='g', channel=5, ht_cap='HT40+')

    net.addLink(dr2, cls=adhoc, intf='dr2-wlan0',
                ssid='adhocNet', proto='batman_adv',
                mode='g', channel=5, ht_cap='HT40+')

    net.addLink(dr3, cls=adhoc, intf='dr3-wlan0',
                ssid='adhocNet', proto='batman_adv',
                mode='g', channel=5, ht_cap='HT40+')
```

DESCRIÇÃO: Executando o script python setNodePosition.py passando como parâmetro os nomes dos drones(objetos stations).

```
info("*** Configure the node position\n")
    setNodePosition = 'python {}/setNodePosition.py '.format(path) +
                sta_drone_send + ' &'
    os.system(setNodePosition)
```

3.5. Performance de Redes de Comunicação para Sistemas de Múltiplos Drones

A avaliação da performance da rede sem fio responsável em viabilizar a comunicação entre as aeronaves (FANET) é fundamental para mitigar falhas na aplicação. Cada aplicação demanda por requisitos específicos de desempenho, portanto a infra-estrutura do sistema multi-VANT deve ser capaz de suprir as suas necessidades. Por exemplo, existem aplicações que demandam latência inferior a 1ms. Elas são classificas como aplicações de baixa latência (*low-latency*) e estão presentes em diversas áreas, desde saúde até transporte, tais como a realização de cirurgias remotas, utilização de carros autônomos, sistemas baseados em realidade aumentada, entre outras [Lema et al. 2017].

Dentro desse contexto, foram estabelecidas métricas de desempenham que são utilizadas para caracterizar a infraestrutura de comunicação. Elas devem ser consideradas desde a etapa de projeto de uma solução de conectividade, afinal o levantamento dos requisitos é uma das primeiras etapas de um projeto. Considerando os conceitos apresentados anteriormente, a configuração, aferição, análise e divulgação de resultados devem seguir padrões para garantir a reprodutibilidade da pesquisa e, além disso, a qualidade dos resultados obtidos [Rehman et al. 2010]. Como sugestão, recomenda-se a leitura da RFC 1242 *Benchmarking Terminology for Network Interconnection Devices* e da RFC 2544 *Benchmarking Methodology for Network Interconnect Devices*, além de suas atualizações.

3.5.1. Métricas de desempenho

Para efeitos de padronização, as métricas de desempenho apresentadas a seguir tiveram como referência a RFC 1242. As métricas apresentadas serão vazão (*throughput*), latência, *jitter* e perda de pacotes. Elas foram selecionadas por estarem alinhas com as ferramentas apresentadas na seção seguinte.

- **Vazão (*Throughput*)**: é a quantidade de dados recebidos corretamente em um intervalo de tempo, normalmente 1 segundo. Para mensurar o comportamento da vazão é importante utilizar tamanhos diferentes de quadros.
- **Latência**: para efeito de simplificação, a latência será considerada o tempo necessário para um pacote acessar o enlace de comunicação e ser completamente recebido no seu destino [Youm and Kim 2013].
- ***Jitter*:** é o intervalo de tempo entre transmissões e recebimentos de pacotes bem-sucedidas. Deve ser aferido de forma contínua. Ele é obtido a partir do desvio padrão da latência [Youm and Kim 2013]. Mais detalhes do cálculo do *Jitter* podem ser encontrados na RFC 3550.
- **Perda de Pacotes**: é uma taxa percentual de pacotes não recebidos ou com falhas.

3.5.2. Ferramentas para Avaliação de Performance

Atualmente, existem diversas ferramentas para avaliação de performance de redes de comunicação. Inclusive, elas estão integradas com sistemas responsáveis por monitorar a infraestrutura, isso inclui também dispositivos de interconexão, como roteadores, pontos de acesso e switches, como também servidores ou qualquer outros dispositivo conectado a rede. O objetivo é manter a alta disponibilidade dos serviços ofertados. Exemplos desse tipo de solução são o Netdata[14] e o Zabbix[15]. Contudo, este capítulo ficara restrito a duas ferramentas simples e bastante eficientes: ping e iperf. Elas são brevemente apresentadas a seguir.

- **Ping:** é uma ferramente utilizada, principalmente, para verificação de conectividade. Faz uso do protocolo ICMP e marcações de tempo para gerar relatórios e estatísticas. Dentre informações que seu relatório fornece, está o *Round Trip Time* (RTT). Nesse contexto, o RTT é a variação de tempo para a requisição ICMP ser respondida.
- **Iperf:** é uma ferramenta para aferição de métricas de performance de redes de comunicação baseadas na pilha de protocolos TCP/IP. Construída na arquitetura cliente/servidor, oferece opções de funcionamento para os protocolos TCP e UDP. Dentre as métricas disponíveis estão: largura de banda, perda de pacotes e jitter. A versão 3 contém mais funcionalidade que seu antecessor. Mais informações estão disponíveis na página oficial da ferramenta[16].

[14]https://www.netdata.cloud/
[15]https://www.zabbix.com/
[16]https://iperf.fr/

Prática 04: Utilizando Ferramentas para Avaliação de Performance de Rede

OBJETIVO: Utilizar as ferramentas apresentadas para aferir métricas de desempenho da rede. A partir disso, realizar alterações de configuração no cenário para efeitos de comparação com resultados obtidos anteriormente.

COMANDO: iperf -s

DESCRIÇÃO: Executar a ferramenta iperf no formato servidor TCP. Após isso, utilize o parâmetro -u para executá-la com o protocolo UDP.

```
root@elefante:~/mininet-wifi/examples/uav# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  6] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 45098
[ ID] Interval       Transfer     Bandwidth
[  6]  0.0-10.9 sec  6.38 MBytes  4.93 Mbits/sec
```

COMANDO: iperf -c <ip do servidor>

DESCRIÇÃO: Executar a ferramenta iperf no formato cliente TCP.

```
root@elefante:~/mininet-wifi/examples/uav# iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.2 port 45098 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.4 sec  6.38 MBytes  5.16 Mbits/sec
```

EXERCÍCIO 03: Como extrair apenas o tempo de RTT médio a partir do relatório emitido pela ferramenta ping?

SOLUÇÃO: Utilizar os comandos tail e cut disponíveis no Linux para tratar a saída. Feito isso, é possível direcionar a saída para um arquivo e plotar gráficos.

```
ping -O -D -c10 10.0.0.1 | tail -1 | cut -d' ' -f4 | cut -d'/' -f2
```

EXERCÍCIO 04: É possível gerar arquivos ”csv”a partir dos relatórios emitidos pela ferramenta iperf?

SOLUÇÃO: Dentre os parâmetros disponíveis para a ferramenta iperf, está opção -y. Ela omite o relatório padrão da ferramenta e gera uma saída ”csv”. A disposição das colunas segue a seguinte sequência, quando emitido pelo servidor UDP: data-hora, ip-local, porta-local, ip-cliente, porta-cliente, id, tempo, total dados transmitidos, bandwith, jitter, pacotes perdidos, pacotes transmitidos, taxa de erro. Diante disso, para obter o jitter, pacotes perdidos e enviados pode-se utilizar o comando a seguir

```
iperf -u -s -yc | cut -d, -f10-12
```

3.6. Estudos de Caso - Inspeção em situações de emergência: incêndios

Devido a alta flexibilidade e rápida implantação dos sistemas multi-VANT, é possível utilizá-los em missões de preservação da vida. Em incêndios, principalmente, florestais, os quais ocorrem em grandes áreas, as aeronaves têm a capacidade de fornecer diversas informações, tais como: direção e intensidade do vento, temperatura, poluentes dispersos no ar, entre outras. Além disso, eles podem fornecer imagens aéreas da região permitindo aos operadores uma melhor visualização da situação. Inclusive, para localizar seres humanos, ou até animais, em situação de risco.

A Figura 3.8 ilustra uma situação de incêndio onde três seres humanos estão em risco. As aeronaves atuam de forma cooperativa para cobrir uma maior área de busca, desse modo é fundamental que eles possam trocas informações entre si. Observa-se que foi criada uma rede ad hoc para viabilizar a rápida implantação do sistema. Um protocolo de roteamento também é utilizado para habilitar a comunicação entre os drones que estão nas extremidade da topologia.

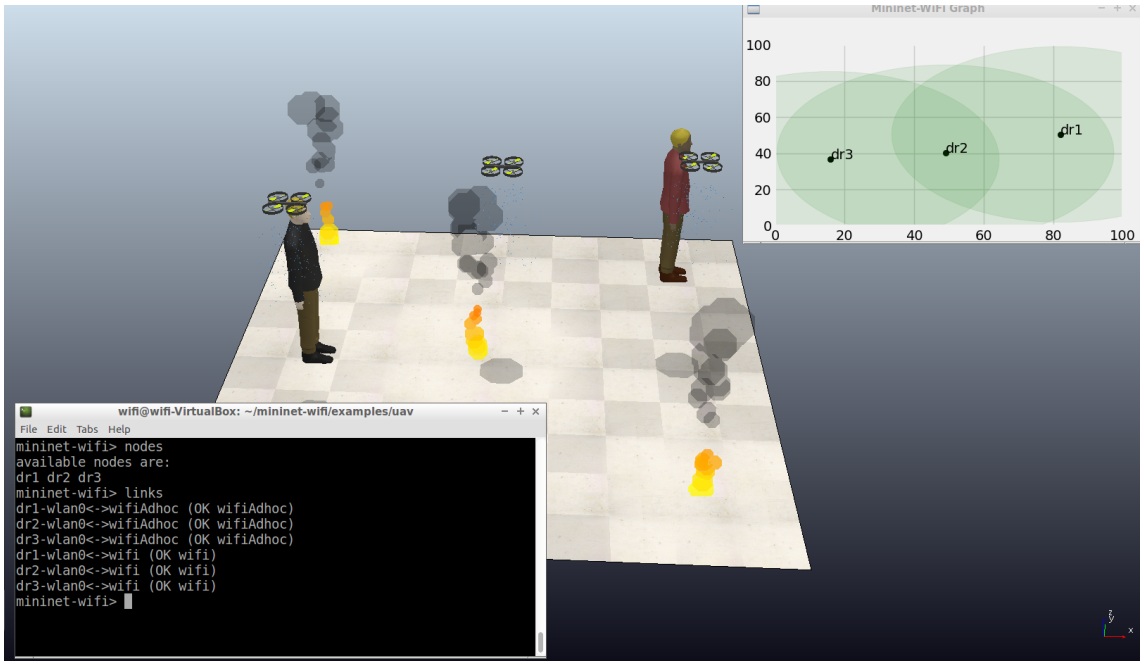


Figura 3.8. Cenário em execução.

Prática 05: Análise da rede de comunicação de um sistema multi-VANT

EXERCÍCIO 05: Faça o estudo da rede de comunicação do sistema multi-UAV. Utilize os comandos e ferramentas apresentados para comparar com o cenário apresentado anteriormente. Observe que no cenário de inspeção a navegação dos drones é controlada por um planejador de caminho.

3.7. Conclusão

Este capítulo apresentou uma estratégia para emular redes de comunicação para sistemas de múltiplos drones. Para tal, foram utilizados o emulador de redes sem fio Mininet-WiFi e o simulador CoppeliaSim. O Mininet-WiFi é responsável por emular o comportamento da rede de comunicação, enquanto o CoppeliaSim a movimentação das aeronaves, além do ambiente virtual que permite a visualização das aeronaves no cenário em execução.

Dentro desse contexto, percebe-se que aplicações que fazem uso de múltiplos drones atuando de forma cooperativa representam uma alternativa interessante para superar alguns problemas, principalmente quando relacionados a missões em locais de alto risco ou de difícil acesso. Contudo, é importante investigar soluções de conectividade para esses sistemas. Para tal, é fundamental apropriar-se de estratégias para construção de cenários e avaliação de performance conforme a aplicação que o sistema está inserido.

O estudo nessa temática e outras linhas aliadas a ela, como, por exemplo, Internet das Coisas, cresceu bastante nos últimos cinco anos. Entende-se que ainda existe um *gap* para poder atender as atuais demandas oriundas da Indústria 4.0. Outros avanços relevantes voltados para a tecnologia embarcada e para inteligência artificial vão posicionar os drones como elementos protagonistas nos próximos anos. Portanto, os sistemas de múltiplos drones devem seguir em forte desenvolvimento e expandindo ainda mais suas capacidades e aplicações.

Currículo dos autores

Diego da Silva Pereira possui graduação em Redes de Computadores pelo Instituto Federal do Rio Grande do Norte (IFRN), Mestrado em Ciência da Computação pela Universidade Estadual do Rio Grande do Norte (UERN) e está em doutoramento no Programa de Pós-graduação em Engenharia Elétrica e de Computação (PPgEEC) pela UFRN. Atualmente é professor no Instituto Federal do Rio Grande do Norte com pesquisas na área de redes de comunicação sem fio aplicadas à sistemas robóticos autônomos e aeroespaciais.

Luís Bruno Pereira do Nascimento possui Bacharelado em Ciência da Computação pela Universidade Estadual do Piauí (UESPI), Mestrado em Engenharia Elétrica e de Computação pela Universidade Federal do Ceará (UFC) e Doutorado em Engenharia Elétrica e de Computação (PPgEEC) pela Universidade Federal do Rio Grande do Norte (UFRN). Atualmente é professor no Instituto Federal do Rio Grande do Norte (IFRN). Suas áreas de interesse incluem robótica, otimização e aprendizado de máquina.

Vitor Gaboardi dos Santos possui graduação em Engenharia Elétrica na Universidade Federal do Rio Grande do Norte (UFRN) com período sanduíche na University of Kansas (USA) e Mestrado em Engenharia Mecatrônica pela UFRN. Atualmente, é professor substituto no Instituto Federal do Rio Grande do Norte (IFRN). Suas áreas de interesse incluem redes neurais, processamento de imagens e visão computacional.

Pablo Javier Alsina possui graduação em Engenharia Elétrica (1987), mestrado na área de controle de motores de indução (1991) e doutorado em Engenharia Elétrica com tema em controle de manipuladores robóticos (1996), pela Universidade Federal da Paraíba. Atualmente é professor titular do Departamento de Engenharia de Computação e Automação (DCA). É professor nos Programas de Pós-Graduação em Engenharia Mecatrônica (PPGEM) e em Engenharia Elétrica e de Computação (PPgEEC) da Universidade Federal do Rio Grande do Norte (UFRN), onde é chefe do Laboratório de Robótica. Desenvolve pesquisas em robótica, nas áreas de controle, planejamento e percepção robótica, com aplicações em robótica assistiva, robótica móvel e Veículos Aéreos Não Tripulados.

Referências

[Bai et al. 2021] Bai, C., Yan, P., Yu, X., and Guo, J. (2021). Learning-based resilience guarantee for multi-uav collaborative qos management. *Pattern Recognition*, page 108166.

[Chriki et al. 2019] Chriki, A., Touati, H., Snoussi, H., and Kamoun, F. (2019). Fanet: Communication, mobility models and security issues. *Computer Networks*, 163:106877.

[Fontes et al. 2015] Fontes, R. R., Afzal, S., Brito, S. H., Santos, M. A., and Rothenberg, C. E. (2015). Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389. IEEE.

[Fu et al. 2019] Fu, Z., Mao, Y., He, D., Yu, J., and Xie, G. (2019). Secure multi-uav collaborative task allocation. *IEEE Access*, 7:35579–35587.

[Hong et al. 2021] Hong, Y., Jung, S., Kim, S., and Cha, J. (2021). Autonomous mission of multi-uav for optimal area coverage. *Sensors*, 21(7):2482.

[Lema et al. 2017] Lema, M. A., Laya, A., Mahmoodi, T., Cuevas, M., Sachs, J., Markendahl, J., and Dohler, M. (2017). Business case and technology analysis for 5g low latency applications. *IEEE Access*, 5:5917–5935.

[Pereira et al. 2020] Pereira, D. S., De Morais, M. R., Nascimento, L. B., Alsina, P. J., Santos, V. G., Fernandes, D. H., and Silva, M. R. (2020). Zigbee protocol-based communication network for multi-unmanned aerial vehicle networks. *IEEE Access*, 8:57762–57771.

[Rehman et al. 2010] Rehman, S. U., Turletti, T., and Dabbous, W. (2010). Benchmarking in wireless networks.

[Rooban et al. 2021] Rooban, S., Suraj, S. D., Vali, S. B., and Dhanush, N. (2021). Coppeliasim: Adaptable modular robot and its different locomotions simulation framework. *Materials Today: Proceedings*.

[Santos et al. 2019] Santos, V. G., Pereira, D. S., Alsina, P., Fernandes, D. H., Nascimento, L. B., Leite, D. L., Morais, M. R., Silva, M. R., and Souza, E. S. (2019). Multi-uav system architecture for environmental protection area monitoring. In *Proc. Anais do Simpsio Brasileiro de Automao Inteligente*, pages 1–6.

[Selvaraju et al. 2021] Selvaraju, S. P., Balador, A., Fotouhi, H., Vahabi, M., and Bjorkman, M. (2021). Network management in heterogeneous iot networks. In *2021 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1581–1586. IEEE.

[Silva et al. 2019] Silva, M. R., Souza, E. S., Alsina, P. J., Leite, D. L., Morais, M. R., Pereira, D. S., Nascimento, L. B., Medeiros, A. A., Junior, F. H. C., Nogueira, M. B., et al. (2019). Performance evaluation of multi-uav network applied to scanning rocket impact area. *Sensors*, 19(22):4895.

[Wang et al. 2013] Wang, S.-Y., Chou, C.-L., and Yang, C.-M. (2013). Estinet openflow network simulator and emulator. *IEEE Communications Magazine*, 51(9):110–117.

[Wang et al. 2021] Wang, X., Yang, L. T., Meng, D., Dong, M., Ota, K., and Wang, H. (2021). Multi-uav cooperative localization for marine targets based on weighted subspace fitting in sagin environment. *IEEE Internet of Things Journal*.

[Youm and Kim 2013] Youm, S. and Kim, E.-J. (2013). Latency and jitter analysis for ieee 802.11 e wireless lans. *Journal of Applied Mathematics*, 2013.

Capítulo 4

Procedimentos adotados na coleta de sinais mioelétricos para construção de próteses da mão

Alberto Monteiro Peixoto, Guilherme de Oliveira Monteiro Peixoto, Roberto Luiz Souza Montteiro, Tereza Kelly Gomes Carneiro

Abstract

This chapter provides information on the precautions and recommendations that need to be taken in collecting the myoelectric signal for use in myoelectric prostheses with multiple movements. For a better understanding of the process, an approach is made about the EMG signal, its treatment, movement identification techniques and the instructions for the acquisition process. A prototype acquisition equipment is also being presented here as an example.

Resumo

Este capítulo traz informações sobre os cuidados e recomendações que precisam ser tomados na coleta do sinal mioelétrico para uso em próteses mioelétricas com vários movimentos. Para melhor compreensão do processo, é feita uma abordagem sobre o sinal EMG, seu tratamento, técnicas de identificação dos movimentos e as instruções para o processo de aquisição. Aqui também está sendo apresentado um protótipo de equipamento de aquisição como exemplo.

4.1. Informações Gerais

Este capítulo tem o objetivo de apresentar o processo de aquisição dos sinais mioelétricos, abordando as condutas que devem ser adotadas para se conseguir maximizar a identificação do número de movimentos realizados pelo membro superior.

A abordagem está organizada de forma a proporcionar, não só o conhecimento das principais condutas que deverão ser adotadas para a identificação dos movimentos, como também a compreensão do processo de captura do sinal. Desta forma, o capítulo está estruturado em cinco subtítulos: Identificação da atividade muscular; Características do sinal mioelétrico; Tratamento dos sinais mioelétricos; Técnicas de identificação dos movimentos; e Instruções para o processo de aquisição.

4.2. Identificação da Atividade Muscular

Uma das condições importante para compreensão e melhor aquisição dos sinais mioelétricos é o entendimento de como os vários músculos contribuem para realização de um determinado movimento, pois o sinal eletromiográfico (EMG), é formado por um conjunto de sinais provenientes de músculos que estão atuando diretamente no movimento investigado, como também de músculos que estão atuando indiretamente.

Portanto, abordaremos a seguir como os músculos contribuem com os movimentos e quais os principais mecanismos que atualmente podem ser utilizados para detectá-los.

4.2.1. Atividade muscular e sua relação com os movimentos

Para entendermos a participação dos músculos em determinados movimentos, utilizaremos a seguir a classificação apresentada por (LIPPERT, 2013) que distingue quatro tipos de músculos:

Agonistas: Este termo é aplicado aos músculos que são os principais responsáveis por um determinado movimento do membro. Se considerarmos, por exemplo, flexão do antebraço sobre o braço, o bíceps braquial seria o agonista.

Antagonistas: Este termo é aplicado aos músculos que agem realizando movimento oposto aos agonistas. Durante a ação de um agonista, os antagonistas geralmente atuam passivamente sem se contrair nem oferecer resistência. Se considerarmos o exemplo da flexão do antebraço sobre o braço, o tríceps braquial seria o antagonista.

Neutralizadores: Este termo é aplicado aos músculos que atuam junto aos agonistas para permitir que um determinado movimento seja executado de forma adequada. Se considerarmos o exemplo da flexão do antebraço sobre o braço, percebemos que o bíceps braquial (agonista) faz a flexão do antebraço sobre o braço, mas produz também o movimento de supinação. Para evitar este segundo movimento, outro músculo atua com o objetivo de neutralizá-lo. Neste caso, o músculo pronador redondo.

Estabilizadores: Este termo é aplicado aos músculos que atuam estabilizando as estruturas do corpo para que determinado movimento seja realizado. Considerando o exemplo da flexão do antebraço sobre o braço, para que este movimento seja feito para realizando um levantamento de peso sem que a articulação do braço sofra deslocamento, o músculo deltoide precisará atuar estabilizando a articulação do ombro (glenoumeral). Portanto, neste caso o deltoide é um musculo estabilizador.

Sendo assim, o sinal mioelétrico capturado na superfície da pele é formado por um complexo conjunto de sinais originados em vários tipos de músculos. Alterações posturais, mudanças na angulação do movimento ou a presença de movimentos paralelos são fatores que podem alterar o padrão do sinal mioelétrico.

Esta informação nos permite entender porque os movimentos precisam ser treinados junto aos processos de aquisição do sinal EMG. Quanto mais treinos, mais bem definido o padrão do sinal capturado e maior a probabilidade de diferenciar um número maior de movimentos.

4.2.2. Mecanismos utilizados para detecção das atividades musculares

Embora o sinal mioelétrico seja a informação mais completa sobre a atividade muscular, existem vários métodos que podem ser utilizados para detectar essa atividade. Abordaremos aqui quatro destes métodos: *eletromiografia; panda ring resonator; optomiografia e mecanomiografia*.

Eletromiografia: Esta é uma técnica utilizada para captura dos sinais elétricos produzidos pelos músculos. A eletromiografia (EMG) pode ser realizada de duas formas: utilizando eletrodos de agulha (IEMG) ou utilizando eletrodos de superfície (SEMG). A forma mais utilizada desta técnica para o uso em próteses é a SEMG, visto que tem a vantagem de ser não invasiva (HARGROVE; ENGLEHART; HUDGINS, 2007).

A SEMG utiliza eletrodos de superfície para fazer a captura dos sinais EMG e a escolha do seu formato, assim como a composição vão depender do objetivo da coleta. A Figura 4.1 apresenta alguns formatos destes eletrodos. Em (a) é apresentado tipo bipolar com barras de Ag/AgCl, sendo um formato utilizado de forma fixa em algumas próteses; em (b) é apresentado um tipo também bipolar com pinos mais afastados; em (c) está apresentado o tipo descartável monopolar, bastante utilizado por ser de baixo preço e de fácil uso, no entanto, o cabo é conectado ao eletrodo através de um conector, o que acaba produzindo ruídos causados por artefatos de movimento; e em (d) temos também um modelo monopolar, sendo diferente do anterior por ter o cabo fixado no eletrodo, evitando os artefatos de movimento.

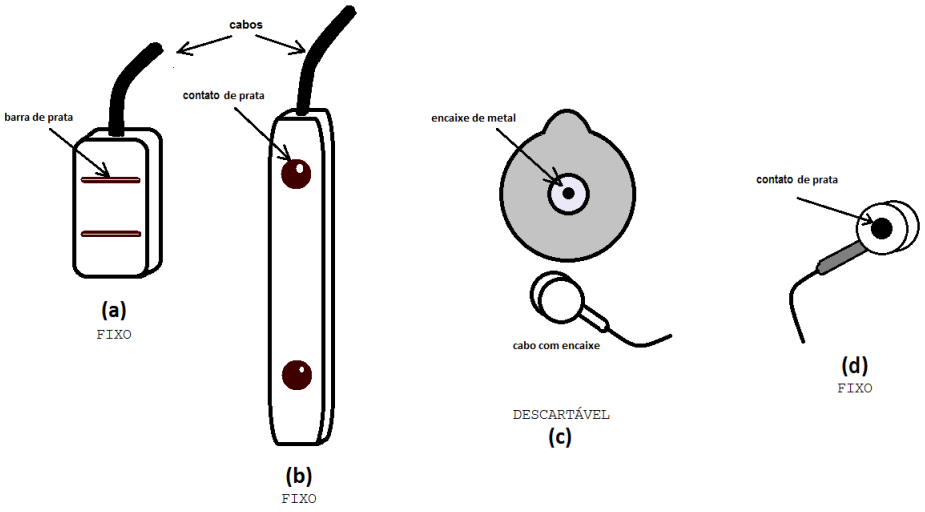


Figure 4.1. Visão geral de alguns tipos de eletrodos utilizados na SEMG.

Estes eletrodos são componentes muito importantes na aquisição dos sinais, visto que são eles que permitem o contato entre a pele e o sistema de captura do sinal.

O contato eletrodo-pele pode ser um gargalo na obtenção do sinal mioelétrico devido à impedância apresentada nesta conexão. O aumento da impedância pode ser produzido tanto por uso de um eletrodo inadequado, quanto por fatores biológicos inerentes à pele.

Os eletrodos mais utilizados para captura dos sinais mioelétricos com o objetivo de

uso em próteses, são os que utilizam os contatos de Ag/AgCl, tanto para os eletrodos fixos quanto para os descartáveis. No caso dos descartáveis, são utilizados os mesmos eletrodos indicados para a aquisição dos sinais cardíacos (ECG). Estes eletrodos utilizam, além dos contatos de Ag/AgCl, gel eletrolítico. Outros procedimentos também são adotados para baixar a impedância da interface eletrodo-pele, que são a retirada de pelos da pele, e a limpeza da superfície para retirada de cremes, suor e tecidos mortos.

O uso da eletromiografia é bastante ampla, perpassando por diagnóstico de doenças; estudo dos movimentos musculares; construções de equipamentos de biofeedback; e para a construção de próteses mioelétricas.

Panda Ring Resonator: Trata-se de um sensor óptico que tem potencial para ser empregado na detecção de movimentos musculares graças a variação da fase e comprimento de onda ocasionados pela perturbação no caminho óptico. Ele é constituído por três ressonadores de micro-anel estruturados como unidade de referência (RL), unidade de detecção (RR) e o terceiro é usado para formar sinais de interferência.

Com esse sistema óptico de detecção muscular somos capazes de identificar facilmente quando ocorre ou não uma perturbação no sistema. Quando não ocorre perturbação, a simetria entre os picos do sinal de referência e o sinal de detecção não mudam. Caso ocorra perturbação, o sinal da unidade de detecção (RR) que foi perturbada pela contração muscular, possuirá um comprimento de onda levemente diferente da unidade de referência.

A relação entre a mudança no comprimento de onda e a perturbação no comprimento do caminho óptico causada pela contração muscular é dada de forma proporcional. Como mostra a Equação (1).

$$\frac{\Delta\lambda m}{\lambda m} = \frac{\Delta n}{n} + \frac{\Delta L}{L} \tag{1}$$

Em que L é o comprimento do caminho óptico, ou seja, a circunferência do ring resonator, n é o índice de refração. Já a relação entre a força aplicada e a variação do comprimento do caminho óptico depende também do módulo de Young e da área da secção transversal. Como mostra a Equação (2).

$$F = \left(\frac{Y_0 A_0}{L_0}\right) \cdot \Delta L \tag{2}$$

Em que F é a força aplicada, Y_0 é o módulo de Young, A_0 é a área inicial da secção transversal, L_0 é o comprimento inicial e ΔL é a variação no comprimento óptico. Em [1] e [2] identificamos uma relação linear entre a força aplicada e a variação do comprimento de onda. Essa relação linear torna viável o uso do sistema óptico de detecção muscular (YOTHAPAKDEE; P.YUPAPIN; TAMEE, 2016).

Assim como em outros sensores de deformação, o ring resonator se baseia também no efeito óptico de tensão (strain-optical effect) que consiste na variação do índice de refração induzido por qualquer deformação aplicada à fibra óptica (TAMEE et al., 2013).

Optomiografia: A optomiografia é uma técnica que pode ser utilizada para iden-

tificar a atividade muscular. O sensor eletro-óptico para detecção de contração muscular apresentado por Chianura e Giardini (2010) se embasa na observação de que as células musculares são acomodadas como fibras alongadas, alinhadas ao longo do eixo principal do músculo. No tecido muscular, espera-se que a luz infravermelha, seja retroespalhada anisotropicamente conforme o músculo se contrai.

O sensor é colocado diretamente na pele, sobre o músculo. Um LED localizado no centro do sensor emite luz através da pele e dois fotodiodos não adjacentes a coletam na direção das fibras, enquanto outros dois fotodiodos coletam a luz espalhada perpendicularmente a essa direção.

Este tipo de sensor óptico parece promissor como uma alternativa ao EMG de superfície, permitindo uma detecção não invasiva do sinal de contração muscular, com vantagens na não captação de ruído eletromagnético e com a capacidade de distinguir entre contrações isométricas e isotônicas.

Mecanomiografia:

Refere-se a uma técnica não-invasiva utilizada para identificar contração muscular. Distintos instrumentos podem ser utilizados como: ondas sonoras e vibrações. Keidel e Keidel (1989) empregou o termo vibromiografia para os detectores de vibração, e Orizio, Perini e Veicsteinas (1989) empregou o termo sonomiografia para os detectores sonores. Este mesmo autor empregou o termo Mecanomiografia em 1993 que passou a englobar todas as técnicas não-invasivas de detecção da atividade muscular através de vibrações e sons medidos por transdutores.

Outro exemplo de instrumento pode ser visto no trabalho de Krueger et al. (2014) que utiliza acelerômetros com três eixos para medir as vibrações de deslocamento geral da fibra muscular nas três direções ortogonais (X, Y e Z). Já no trabalho de Watakabe et al. (2001) é mostrado a diferença entre sinais mecanomiograma (MMG) capturados utilizando um microfone e um acelerômetro, evidenciando que as características do sinal MMG dependem do tipo do sensor utilizado.

Por fim, embora várias técnicas para detecção da atividade motora tenha surgido nos últimos anos, a técnica mais utilizada e que tem se mostrado a mais adequada para identificação dos movimentos relacionados à contração muscular, ainda tem sido a eletromiografia.

4.3. Sinal Mioelétrico

O sinal EMG é formado pelo agrupamento de sinais provenientes dos músculos em atividade, sendo estes músculos os responsáveis por determinado movimento. A compreensão da sua origem, suas características e como o sinal EMG varia em função da participação de cada músculo, é fundamental para escolha e construção da técnica de captura. Sendo assim, abordaremos mais detalhes sobre estes pontos.

4.3.1. Origem e características do sinal mioelétrico

A origem do sinal mioelétrico encontra-se na estrutura formada por um neurônio motor (motoneurônio) e um conjunto de células musculares (fibras musculares) inervadas por ele. A Figura 4.2 ilustra um exemplo de unidade motora (UM). O sinal é gerado a partir

da passagem de íons entre os lados interno e externo das células. Essa onda se propaga pelo tecido e recebe o nome de potencial de ação da unidade motora (MUAP).

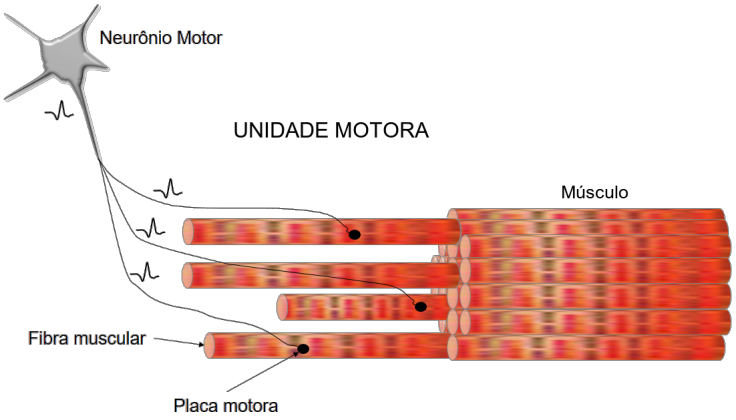


Figure 4.2. Ilustração de uma unidade motora. Fonte: próprio autor

Cada músculo envolvido no movimento dos membros, é responsável por um sinal composto pela somatória de centenas de MUAPs. Considerando que a mobilização de um determinado membro exige a participação de vários músculos, o sinal capturado na superfície apresenta-se bastante complexo. Portanto, podemos afirmar que o sinal EMG é o somatório de milhares de MUAPs. A Figura 4.3 ilustra a captura dos sinais utilizando um amplificador diferencial.

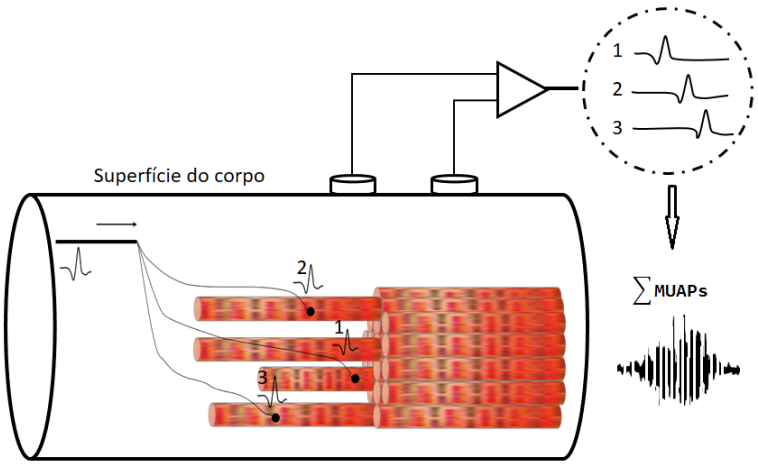


Figure 4.3. Ilustração da captura do sinal EMG formado pelos vários MUAPs. Fonte: próprio autor

A aparência do sinal capturado é semelhante ao que observamos na Figura 4.4 (A). Trata-se de um sinal praticamente aleatório, mas durante a captura em uma fase de contração e relaxamento do músculo, ele apresenta um padrão Gaussiano. Em (B), está apresentado o espectro de frequência do sinal. Embora varie de 1hz a 500hz, sua faixa mais importante encontra-se entre 6hz e 20hz, sendo esta a faixa de frequência onde a maioria das fibras musculares está atuando. A intensidade do sinal pode chegar aos 10 milivolts na superfície da pele quando medido em atletas (MERLETTI; PARKER, 2004).

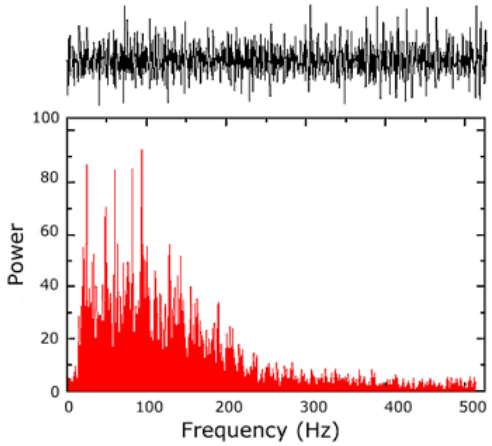


Figure 4.4. (A) sinal EMG bruto; (B) espectro de frequência do sinal. Fonte: (DE LUCA, 2002)

4.3.2. Comportamento dos sinais na presença de movimentos

O padrão do sinal EMG pode refletir um determinado movimento executado por um membro através de um padrão, no entanto, vários fatores podem alterar esse padrão. Por tanto, vamos abordar os fatores que consideramos mais importantes de serem controlados para que sejam aumentadas as chances de acertar quais os movimentos estão sendo executados a partir da identificação do padrão dos sinais EMG.

Eletrodos: O projeto SENIAM [1] faz algumas recomendações em relação aos eletrodos para que a aquisição seja feita de forma adequada (STEGEMAN; HERMENS, 2007):

- Tamanho dos eletrodos: recomenda-se que sejam utilizados tamanhos que variam de $1mm^2$ a alguns cm^2. Os eletrodos maiores, principalmente seguindo o sentido das fibras musculares apresentam maior ganho e apresentam comportamento de filtro *passa-baixa*;

- Distância entre os eletrodos: segundo o projeto, a distância mais adequada é de 20 mm entre os eletrodos, com a observação de que em músculos pequenos, essa distância deve estar limitada a $\frac{1}{4}$ do tamanho do músculo.

- Posicionamento dos eletrodos: o posicionamento indicado é na região da pele sobre os ventres musculares dos principais músculos que estão participando do movimento.

- Materiais dos eletrodos: como já foi mencionado, é recomendo os eletrodos com terminais com AgCl, Ag ou Au.

Local da coleta: O local é algo que depende do número de eletrodos a serem utilizados e, embora sejam recomendadas a aplicação sobre os ventres musculares (regiões

[1] Projeto que padronizar a forma de aquisição do sinal mioelétrico de superfície (www.seniam.org)

de maior volume do músculo), a escolha de quais deles serão utilizados é algo que depende de testes. O mais importante é saber que uma vez determinado o local da coleta, não poderá haver variações na colocação, visto que isto implica na alteração do padrão do sinal.

Padronização do movimento: Outro fator importante é a padronização do movimento, ou seja, o movimento precisa ser executado sempre com a mesma força, amplitude e angulações para que o padrão seja estável. Nestas condições é possível identificar o movimento a partir dos sinais capturados.

Treinamento: Chamamos de treinamento, a repetição de determinado movimento com o objetivo de conseguir um padrão de sinal que possa ser associado exclusivamente ele.

4.4. Tratamento dos sinais mioelétricos

Uma das condutas mais importantes quando se procura utilizar os sinais EMG para uso em próteses é o tratamento destes sinais. As várias interferências a que estão submetidos dificultam sua classificação. Assim, apresentaremos alguns dos fatores que alteram o padrão do sinal EMG e que devem ser controlados.

4.4.1. Artefatos e ruídos presentes na aquisição dos sinais

Os artefatos e ruídos são interferências que produzem distorções nos sinais biológicos, e que não são de interesse para um determinado estudo (FERREIRA, 2007). Condutas importantes para diminuir estas interferências são: manter os eletrodos bem fixados à pele, e os cabos bem fixados aos eletrodos para evitar movimentos; evitar exposições a campos eletromagnéticos intensos; utilizar amplificadores com alta relação sinal/ruído.

Os amplificadores são muito importantes nesse processo, visto que seus ganhos são muito altos. Valores em torno de 10 mil vezes são utilizados, visto que os sinais são da ordem de 0,3 a 10 mV, por isso a relação sinal/ruído é um fator muito importante.

4.4.2. Alguns filtros utilizados na melhoria da qualidade dos sinais

A filtragem dos sinais está relacionada ao uso destes sinais. Quando o objetivo é diagnóstico, a filtragem é bastante complexa e vários filtros são aplicados tanto, utilizando componentes físicos quanto utilizando filtros digitais. Nestes casos, a filtragem deve procurar eliminar inclusive os chamados cross-talk, que são os sinais oriundos dos músculos próximos ao que está sendo avaliado.

No caso do uso para próteses, a filtragem é menos exigente podendo o cross-talk, por exemplo, fazer parte do padrão do sinal EMG. Isto vai depender ainda se o sinal EMG será utilizado apenas para classificar o movimento que deu origem ao sinal, ou se será usado para medir a força e os movimentos finos. Os filtros mais utilizados no primeiro caso são o *passa-baixa* para evitar a produção de *aliasing*, e o filtro de 60Hz.

4.4.3. Parte do sinal a ser considerada, dependendo do objetivo

Durante o processo de aquisição, dependendo do objetivo (classificação do movimento ou identificação de início meio e fim do movimento) parte da aquisição pode ser descartada.

No segundo caso, deve ser incrementado algoritmo de leitura do espectro de frequência para medir a força muscular e outros que permitissem identificar o início e o fim do movimento. Quando o uso é classificador, a parte inicial e final do sinal é desconsiderada.

4.5. Técnicas de Identificação dos Movimentos

É muito importante, antes de procurar identificar a técnica utilizada para identificação do movimento associado ao padrão do sinal EMG, determinar qual o objetivo da coleta. Algumas próteses com baixas funcionalidades necessitam apenas dos estados dos membros (flexão, extensão, abdução, adução etc.). Diante dessa informação, as próteses posicionam-se mimetizando o estado identificado.

Outras próteses com altas funcionalidades necessitam de informações mais detalhadas, como o momento em que o movimento foi iniciado e finalizado, assim como a variação de força durante o processo de contração e relaxamento. Para estas identificações é necessário medir não só a amplitude do sinal, como também o espectro de frequência do sinal EMG e analisar quais as frequências mais importantes em tempo de execução. Aqui, podem ser utilizadas algumas técnicas complementares, sejam elas no domínio do tempo, no domínio da frequência, ou no domínio tempo-frequencia. Podemos citar a Transformada Rápida de Fourier (FFT); Transformada Wavelet (WT) e Transformada Wavelet Packet (WPT).

Neste capítulo iremos abordar de forma resumida as técnicas para próteses simples, portanto, iremos falar das técnicas de classificação dos movimentos. Considerando que existem varias técnicas com este objetivo, citaremos apenas três delas.

4.5.1. Redes Neurais

Quando se escolhe as Redes neurais para fazer classificação de movimento através dos sinais EMG, não há necessidade de hardware robusto se estivermos falando de apenas dois movimentos, exemplo flexão e extensão. Podemos ver isto na construção da Mão de São Carlos (CUNHA, 2002). Já para a prótese mimetizar vários movimentos, se faz necessário o uso de Deep Learning e isto exige muito mais esforço de processamento.

A unidade funcional que deu origem ao nome da Rede Neural é o neurônio artificial (HAYKIN, 2001). A Figura 4.5 apresenta um modelo deste neurônio onde X_1 a X_n correspondem aos sinais de entrada; W representam um determinado peso que correspondem às sinapses dos neurônios biológicos; k_n é cada um dos neurônios utilizados.

Os sinais provenientes dos neurônios são aplicados em um somador ($\sum$) que soma todas as multiplicações W_{kn} resultantes e direciona o resultado U_k para uma função de ativação (φ) que determina a saída Y_k. No modelo apresentado existe um threshold que regula esse limiar de ativação.

Matematicamente, o sinal de saída pode ser expresso conforme as equações 1 e 2. Sendo U_k o valor de saída do somador e Y_k a saída da função de ativação do neurônio.

As Redes Neurais podem ser aplicadas em várias áreas, entre elas podemos citar o diagnóstico de doenças musculares, a Interação Humano Computador (IHC) e as próteses. Nesta última, pode atuar na classificação de sinais EMG (AHSAN; IBRAHIMY; KHALIFA, 2012).

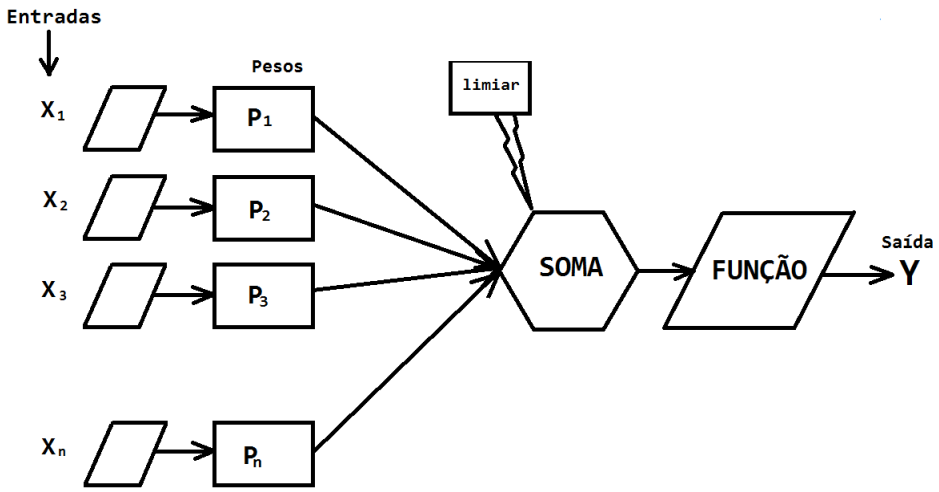


Figure 4.5. Modelo de um neurônio artificial não linear utilizado em Redes Neurais. Fonte: Modificado de (HAYKIN, 2001, p. 36)

4.5.2. Sistema de Inferência fuzzy - FIS

Desenvolvido por Zadeh (1965), é um método capaz de imitar a tomada decisão humana de forma mais semelhante, do que outros classificadores. A natureza não estacionária do sinal EMG tem sido um dos fatores que dificultam sua classificação, além de não ter um padrão de repetição. Isto dificulta para os classificadores estatísticos, mas o método fussy apresenta-se adaptável no reconhecimento do padrão (KHEZRI; JAHED, 2011).

Este método tem sido usado como classificador dos sinais EMG associado a outros métodos, como o feedback visual. Um projeto que fez uso da inferência fussy para controle de prótese multifuncional foi o de Ajiboye e Weir (2005).

4.5.3. Random Forest

O *Random Forest* é um método de aprendizagem supervisionado. Seu funcionamento assemelha-se a um fluxograma contendo nós com ramificações. Em cada nó é feito um teste para tomada de decisão, e isso tem um formato de árvores. A construção das árvores é mais rápida do que outros métodos utilizados para construção dos classificadores, sendo também tão preciso, ou mais, do que estes outros (SHARMA; KUMAR, 2016).

Vários algoritmos de *Random Forest* estão disponíveis. Citamos aqui seis deles: CHID, CART, ID3, C4.5, C5.0, Sec5 e Hunt's Algorithm. Quatro destes podem ser visto com mais detalhe na tabela 4.1. O termo Pruning faz referência à técnica de aprendizagem de máquina que otimiza o tamanho das árvores de decisão, atuando na remoção das seções que fornecem pouco poder de classificação. Já o termo Boosting faz referência ao algoritmo capaz de fazer a conversão de aprendizagem fraca em forte (PATEL; RANA, 2014).

Por fim, o *Random Forest* é um conjunto de árvores de decisão aleatória. Neste método cada árvore indica uma previsão da classificação, aquela que for mais indicada na floresta é eleita a previsão do método.

Table 4.1. Comparação entre algoritmos de Árvore de Desisão.

Parâmetros	ID3	C4.5	C5.0	CART
Tipo de dado	Categoria	Contínuo e Categoria	Contínuo, Categoria, Data, Tempo	Contínuo, Atributos, Data
Velocidade	Baixa	Maior que ID3	Alta	Média
Pruning	Não	Pré-Pruning	Pré-Pruning	Post-Pruning
Boosting	Não suportado	Não suportado	Não suportado	Não suportado
Ausência de Valores	Não lidar	Não lidar	Pode lidar	Pode lidar
Fórmula	Usa entropia e ganho de informação	Informação dividida e taxa de ganho	Mesmo C4.5	índice de diversidade Gini

4.6. Instruções para o processo de aquisição

As instruções para uma aquisição adequada não se limita a condutas. Trata-se de um processo personalizado devido às variações anatômicas e fisiológicas entre os indivíduos. Assim, entendemos que se inicia na seleção ou construção própria de um equipamento de aquisição, na construção de uma interface de treinamento e das recomendações apresentadas no projeto SENIAM.

4.6.1. Equipamento de aquisição

A escolha ou construção do equipamento adequado depende de vários fatores como: quantos canais serão utilizados, e isto depende da quantidade de movimentos que se pretende identificar, além da quantidade de músculos envolvidos; custos, software embutido e dos filtros necessários. Equipamentos comerciais de eletromiografia geralmente não é uma boa opção pelo custo, tamanho e por apresentarem recursos dispensáveis aos projetos de próteses.

Apresentaremos aqui um exemplo de construção de um equipamento de aquisição construído pelo programa de pós-graduação em modelagem computacional e tecnologia industrial do SENAI CIMATEC em parceria com a UNCISAL e apoio da FAPESB (PEIXOTO, 2021)

Trata-se de um equipamento em fase de protótipo que contém cinco canais de aquisição. Segundo o autor, com apenas três canais é possível identificar pelo menos cinco movimentos da mão e punho. Para sua confecção foram utilizadas cinco placas comerciais de aquisição de sinal EMG modelo *Myo WareTM Muscle Sensor (AT-04-001)*. Cada placa tem 1 canal de entrada e foram arranjada em paralelo para a composição dos 5 canais.

Além das placas de aquisição, foi utilizada uma placa Arduíno® para realizar a conversão A/D (analógico digital). Também compõem o equipamento: uma bateria interna, um regulador de tensão e os cabos com os eletrodos. O protótipo pode ser visto na Figura 4.6 cuja composição apresenta-se dividida em três partes.

4.6.2. Algoritmo classificador

O algoritmo utilizado neste projeto foi o Random Forest. O objetivo foi apenas de identificar as posições em que a mão e punho se encontravam e classificá-los conforme um grupo de posições pré-estabelecidas.

A utilização do Random Forest proporcionou um acerto de cinco posicionamentos da mão e punho que variou de 72% a 99% utilizando 3 canais. Um trabalho realizado na China utilizando a braçadeira *Myo armband 8 canais* para identificação de 10 movimentos da mão teve um percentual de acerto entre 73,7% a 100%.

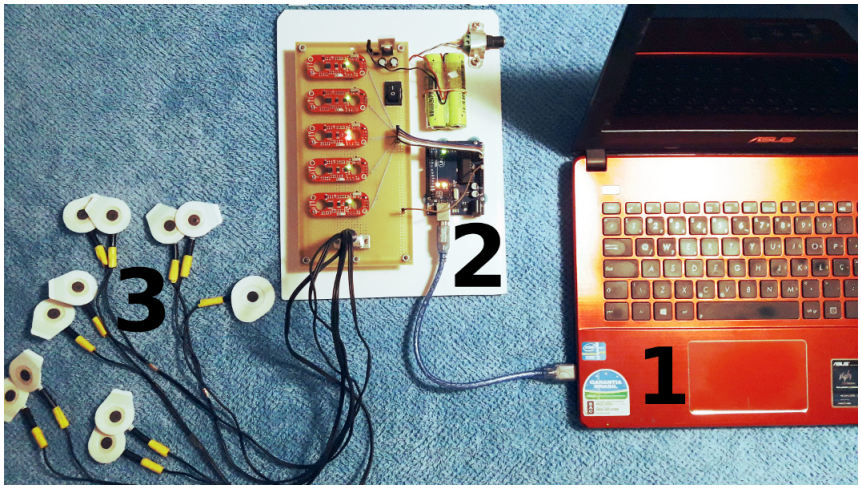


Figure 4.6. Equipamento de aquisição de sinais EMG. (1) computador, (2) placa de aquisição, (3) conjunto de canais. Fonte: (PEIXOTO, 2021)

4.6.3. Procedimentos recomendados

As recomendações apresentadas aqui estão em conformidade com o projeto SENIAM e em maior parte obtidas do trabalho de Peixoto (2021). A lista a seguir apresenta estas recomendações divididas em quatro partes.

Cuidados gerais

- Limpeza da pele com água e sabão para retirada do suor e produtos químicos
- Retirada de pelos
- Evitar proximidade com aparelhos que emitem campos eletromagnéticos intensos

Características do sistema de aquisição

- Utilizar eletrodos descartáveis que tenham contato de Ag/AgCl e gel sólido, e não reutilizar estes eletrodos
- Em caso de uso prolongado do eletrodo, substituir o descartável pelo fixo
- Utilizar bateria nos equipamentos de aquisição em vez de ligar direto à rede elétrica

Posicionamento dos eletrodos

- Manter a distância de 2 a 2,5cm entre os eletrodos de cada canal
- Prender os eletrodos de forma que não fiquem instáveis
- As recomendações para a localização dos eletrodos é que sejam colocados sobre os principais ventres musculares relacionados aos movimentos realizados.

Aquisição do sinal

- Antes das aquisições para treinamento dos algoritmos, deve-se realizar treinos com feedback para padronização dos sinais
- Não há um número ideal de aquisições previamente determinado, deve-se testar qual o número mais adequado para a aprendizagem do algoritmo. O número de aquisições e o treinamento são dois componentes relacionados e seus aumentos tendem a aumentar a eficiência do algoritmo
- Em caso do sinal ser utilizado para outros fins que não a prótese mioelétrica, escolher preferencialmente o membro predominante, de maior uso da pessoa

References

AHSAN, M. R.; IBRAHIMY, M. I.; KHALIFA, O. O. The Use of Artificial Neural Network in the Classification of EMG Signals. In: *2012 Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing*. IEEE, 2012. p. 225–229. ISBN 978-1-4673-1956-0. Disponível em: <http://ieeexplore.ieee.org/document/6305853/>.

AJIBOYE, A.; WEIR, R. A Heuristic Fuzzy Logic Approach to EMG Pattern Recognition for Multifunctional Prosthesis Control. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, v. 13, n. 3, p. 280–291, sep 2005. ISSN 1534-4320. Disponível em: <http://ieeexplore.ieee.org/document/1506815/>.

CHIANURA, A.; GIARDINI, M. E. An electrooptical muscle contraction sensor. *Medical & Biological Engineering & Computing*, v. 48, n. 7, p. 731–734, jul 2010. ISSN 0140-0118. Disponível em: <http://link.springer.com/10.1007/s11517-010-0626-x>.

CUNHA, F. L. da. *Mão de São Carlos, uma prótese multifunção para membros superiores: um estudo dos mecanismos, atuadores e sensores*. Tese (Doutorado) — Universidade de São Paulo, São Carlos, apr 2002. Disponível em: <http://www.teses.usp.br/teses/disponiveis/18/18133/tde-13032006-124951/>.

DE LUCA, C. J. *Surface Electromyography: detection and recording*. DELSYS Incorporated, 2002. 1–10 p. Disponível em: <https://www.delsys.com/downloads/TUTORIAL/semg-detection-and-recording.pdf>.

FERREIRA, É. L. C. *Análise da interferência de ruídos e artefatos no processo de aquisiçao e processamento digital de um sinal biológico*. Tese (Doutorado) — Univap, 2007.

HARGROVE, L.; ENGLEHART, K.; HUDGINS, B. A Comparison of Surface and Intramuscular Myoelectric Signal Classification. *IEEE Transactions on Biomedical Engineering*, v. 54, n. 5, p. 847–853, may 2007. ISSN 0018-9294. Disponível em: <http://ieeexplore.ieee.org/document/4154997/>.

HAYKIN, S. *Redes neurais: princípios e prática*. 2. ed. Porto Alegre: Bookman, 2001. 902 p. ISBN 978-85-7307-718-6.

KARMEN, G.; GABRIEL, D. A. *Essentials of Electromyography*. United States: Human Kinetics, 2010. 256 p. ISBN 0-7360-8550-5.

KEIDEL, M.; KEIDEL, W.-D. The Computer-Vibromyography as a Biometric Progress in Studying Muscle Function - Die Computer-Vibromyographie - Ein biometrischer Ansatz zur Messung mechanischer Muskelaktivität. *Biomedizinische Technik/Biomedical Engineering*, v. 34, n. 5, p. 107–116, 1989. ISSN 0013-5585. Disponível em: <https://www.degruyter.com/document/doi/10.1515/bmte.1989.34.5.107/html>.

KHEZRI, M.; JAHED, M. A NeuroFuzzy Inference System for sEMG-Based Identification of Hand Motion Commands. *IEEE Transactions on Industrial Electronics*, v. 58, n. 5, p. 1952–1960, may 2011. ISSN 0278-0046. Disponível em: <http://ieeexplore.ieee.org/document/5491165/>.

KRUEGER, E.; SCHEEREN, E. M.; NOGUEIRA-NETO, G. N.; BUTTON, V. L. d. S. N.; NOHAMA, P. Advances and perspectives of mechanomyography. *Revista Brasileira de Engenharia Biomédica*, v. 30, n. 4, p. 384–401, dec 2014. ISSN 1517-3151. Disponível em: <http://www.scielo.br/scielo.php?script=sci{_}arttext{\&}pid=S1517-31512014000400009{\&}lng=e>.

LIPPERT, L. S. *Cinesiologia Clínica e Anatomia*. 5. ed. Rio de Janeiro: Gua, 2013. 348 p. ISBN 978-85-277-2190-5.

MERLETTI, R.; PARKER, P. *Electromyography Physiology, Engineering, and Non-Invasive Applications*. New Jersey: John Wiley & Sons, Inc., Hoboken, 2004. 493 p. ISBN ISBN 0-471-67580-6.

ORIZIO, C.; PERINI, R.; VEICSTEINAS, A. Muscular sound and force relationship during isometric contraction in man. *European Journal of Applied Physiology and Occupational Physiology*, v. 58, n. 5, p. 528–533, mar 1989. ISSN 0301-5548. Disponível em: <http://link.springer.com/10.1007/BF02330708>.

PATEL, B. R.; RANA, K. K. A Survey on Decision Tree Algorithm For Classification. *IJEDR*, v. 2, n. 1, p. 2321–9939, 2014. Disponível em: <https://www.ijedr.org/papers/IJEDR1401001.pdf>.

PEIXOTO, A. M. *Aquisição de sinais EMG da região do antebraço para uso em róteses mioelétricas: contexto, experimentos e definição de um protocolo*. 105 p. Tese (Doutorado) — SENAI CIMATEC, 2021.

POZZO, M. Electromyography (EMG), Electrodes and Equipment for. In: *Wiley Encyclopedia of Biomedical Engineering*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2006. Disponível em: <http://doi.wiley.com/10.1002/9780471740360.ebs1425>.

SHARMA, H.; KUMAR, S. A Survey on Decision Tree Algorithms of Classification in Data Mining. *International Journal of Science and Research (IJSR)*, v. 5, 2016. Disponível em: <https://www.researchgate.net/publication/324941161>.

STEGEMAN, D. F.; HERMENS, H. J. Standards for surface electromyography: the european project surface emg for non-invasive assessment of muscles (seniam). p. 108–112, 2007.

TAMEE, K.; CHAIWONG, K.; YOTHAPAKDEE, K.; YUPAPIN, P. P. Muscle sensor model using small scale optical device for pattern recognitions.(Research Article). *The Scientific World Journal*, v. 13, 2013. ISSN 1537-744X.

WATAKABE, M.; MITA, K.; AKATAKI, K.; ITOH, Y. Mechanical behaviour of condenser microphone in mechanomyography. *Medical & Biological Engineering & Computing*, v. 39, n. 2, p. 195–201, mar 2001. ISSN 0140-0118. Disponível em: <http://link.springer.com/10.1007/BF02344804>.

YOTHAPAKDEE, K.; P.YUPAPIN, P.; TAMEE, K. Facial Gesture Measurement Using Optical Muscle Sensing System. *Nano Biomedicine and Engineering*, v. 7, n. 4, jan 2016. ISSN 2150-5578. Disponível em: <http://nanobe.org/Data/View/268?type=100>.

ZADEH, L. Fuzzy sets. *Information and Control*, v. 8, n. 3, p. 338–353, jun 1965. ISSN 00199958. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S001999586590241X>.

Capítulo 5

Introdução à Análise de Dados Geoespaciais com Python

Gesiel Rios Lopes, Alexandre C. B. Delbem, Joélcio Braga de Sousa

Resumo

A análise espacial, ou apenas análise geoespacial, é uma abordagem para aplicar a análise estatística e outras técnicas analíticas a dados que possuem um aspecto geográfico ou espacial. Essa análise normalmente é feita utilizando técnicas de renderização de mapas a partir do processamento de dados espaciais e a aplicação de métodos analíticos a conjuntos de dados terrestres ou geográficos. Este minicurso é uma introdução à análise de dados geoespaciais com Python, com foco em dados vetoriais tabulares usando GeoPandas. O conteúdo concentra-se em apresentar as diferentes bibliotecas para trabalhar com dados geoespaciais e as relações no espaço. Isso inclui a importação de dados em diferentes formatos (por exemplo, `shapefile`, `GeoJSON`*), visualizando, combinando e organizando-os para análise, fazendo o uso de bibliotecas como* `pandas`, `geopandas`, `shapely`, `pyproj`, `matplotlib`, `cartopy`, *dentre outras.*

5.1. Introdução

A compreensão da distribuição espacial a partir de dados originados de fenômenos ocorridos no espaço constitui um grande desafio para diversas áreas do conhecimento, seja em saúde, em geologia, em agronomia, computação entre tantas outras. Tais estudos vem se tornando cada vez mais comum, devido a crescente disponibilidade de dados espaciais, além do crescimento vertiginoso das tecnologias de Sistemas de Informação Geográficas - SIG (do inglês, *Geographic Information System – GIS*) aliada procedimentos computacionais e recursos humanos [Monteiro et al. 2004].

Entender a distribuição dos dados geoespaciais, ou seja, levando em conta a localização espacial do fenômeno em estudo de forma explícita, e traduzi-los em padrões considerando propriedades mensuráveis e relacionadas faz parte da Análise Espacial de Dados Geográficos [Monteiro et al. 2004].

A partir dos dados espaciais, é possível descobrir não apenas a localização, mas também o comprimento, tamanho, área ou forma de qualquer objeto. Os dados geo-

espaciais têm um grande número de aplicações em nossa vida cotidiana, indo desde o entendimento e modelagem do comportamento urbano de pessoas, veículos e outros objetos móveis e utilizando esse entendimento na construção e aperfeiçoamento de modelos de contágio, auxiliando no desenvolvimento de ações e medidas preventivas no controle de epidemias [Zheng et al. 2014, Domingues et al. 2020].

Neste minicurso será apresentada uma introdução à análise de dados geoespaciais com Python, com foco em dados vetoriais tabulares a partir de diferentes bibliotecas para trabalhar com dados geoespaciais e as relações no espaço, como `pandas`, `geopandas`, `shapely`, `pyproj`, `matplotlib`, dentre outras.

5.2. Introdução aos dados vetoriais

Como já mencionado, quando falamos sobre dados geoespaciais, estamos falando sobre os dados que representam objetos/características na superfície da Terra e sua localização específica no globo, sendo portanto disponibilizados em vários formatos. Um dos formatos mais utilizados em aplicações de análise geoespaciais é o de **Dados Vetoriais** [Monteiro et al. 2004, Domingues et al. 2020, Lawhead 2015].

O formato dos dados vetoriais baseia-se na utilização de pontos ou vértices sequenciais para definir a localização e os limites de um objeto, ou seja, conectar esses pontos forma uma linha e conectar essas linhas acabará por criar um polígono. Este polígono certamente envolverá uma área. Podemos facilmente usar isso para representar um objeto ou uma região na superfície superficial da terra. Portanto, podemos dizer que os vetores são mais bem usados para apresentar generalizações de objetos ou características na superfície da Terra [Domingues et al. 2020].

5.2.1. Formatos para criar e compartilhar o conjunto de dados espaciais

Shapefile: O `shapefile` é um formato não-topológico para bases de dados geoespaciais e vetoriais. É considerado um formato aberto, apesar de proprietário. Por ser aberto, o formato recebe suporte de diversos aplicativos de processamento de mapas gratuitos e de código livre. Apesar de ser um termo no singular, o formato `shapefile` consiste numa coleção de arquivos de mesmo nome e terminações diferentes, armazenados no mesmo diretório. Existem três arquivos obrigatórios para o funcionamento correto de um `shapefile`: `.shp`, `.shx` e `.dbf`. O arquivo `shapefile` propriamente dito é o `.shp`, mas se distribuído sozinho não será capaz de exibir os dados armazenados. A distribuição deve ser feita juntamente com os outros dois arquivos.

- `.shp` — formato shape; as características da geometria propriamente dita;
- `.shx` — formato índice de shape, ou seja, um índice com as características das geometrias para permitir buscas mais rápidas;
- `.dbf` — formato de atributos; atributos apresentados em colunas para cada "`shape`"

GeoJSONs: O `GeoJSON` é um formato de intercâmbio de dados geoespaciais de padrão aberto que representa características geográficas simples e seus atributos não

espaciais, ou seja, informações não espaciais sobre uma característica geográfica. A ideia central é fornecer uma especificação para codificação de dados geoespaciais, permanecendo decodificáveis por qualquer decodificador `JSON`. Sendo um subconjunto do `JSON` imensamente popular, o suporte de análise está em um nível diferente do !Shapefile!. Além disso, para obter suporte da maioria dos SIGs. Os recursos suportados pelo `GeoJSON` são pontos, `MultiPoint`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString` e `MultiPolygon`.

GeoPackage: O GeoPackage foi desenvolvido pela *Open Geospatial Consortium* (OGC), tornando-se a alternativa oficial para o `Shapefile`. É um subconjunto do SQLite, que por sua vez é uma implementação SQL leve projetada para bancos de dados autônomos. Semelhante ao `GeoJSON`, isso torna o `GeoPackage` altamente compatível por design e acessível a qualquer SIG.

5.2.2. Ferramentas e pacotes python para começar com os dados geoespaciais

Os pacotes python que podem ser utilizados para começar a explorar dados geoespaciais são os seguintes:

- **Pandas:** Pandas[1] é uma biblioteca licenciada com código aberto que oferece estruturas de dados de alto desempenho e de fácil utilização voltado a análise de dados para a linguagem de programação Python [Coelho 2017].
- **Matplotlib:** O Matplotlib[2] é uma biblioteca de plotagem 2D do Python que produz números de qualidade de publicação em vários formatos de cópia impressa e ambientes interativos entre plataformas. O Matplotlib pode ser usado em *scripts* Python, nos *shell* Python e IPython, *notebooks* Jupyter e em servidores *web*.
- **GDAL:** GDAL[3] é uma biblioteca de tradução para formatos de dados geoespaciais vetoriais e raster que é lançada sob uma Licença de Código Aberto do estilo X/MIT pela Open Source Geospatial Foundation. Como uma biblioteca, ela apresenta um único modelo de dados abstratos de rasterização e um único modelo de dados abstratos de vetor para o aplicativo de chamada para todos os formatos suportados.
- **Shapely:** Shapely[4] é um pacote Python para análise teórica de conjunto e manipulação de recursos planares usando (via módulo ctypes do Python) funções da biblioteca GEOS[5] bem conhecida e amplamente implantada.
- **GeoPandas:** GeoPandas[6] é um projeto de código aberto para facilitar o trabalho com dados geoespaciais em python. GeoPandas estende os tipos de dados usados pelos pandas para permitir operações espaciais em tipos geométricos. As operações geométricas são realizadas pelo shapely.

[1]Disponível em `https://pandas.pydata.org/`

[2]Disponível em `https://matplotlib.org/`

[3]Disponível em `https://gdal.org/`

[4]Disponível em `https://github.com/Toblerity/Shapely`

[5]GEOS, uma porta do Java Topology Suite (JTS), é o mecanismo de geometria da extensão espacial PostGIS para o PostgreSQL RDBMS

[6]Disponível em `https://geopandas.org/`

- **Seaborn:** Seaborn[7] é uma biblioteca de visualização de dados Python baseada no *matplotlib*. Ela fornece uma interface de alto nível para desenhar gráficos estatísticos atraentes e informativos.

Para o desenvolvimento deste minicurso, será utilizado o ambiente do Jupyter Notebook, uma interface de programação literária interativa[8] muito interessante para criar seus modelos e compartilhar com quem quiser, disponibilizado pelo *Anaconda*[9], conforme Figura 5.1 [Shen 2014, Pimentel et al. 2021]. Em [Pimentel et al. 2021] e [Lopes et al. 2019] pode ser encontrado um vasto material de como utilizar o Jupyter Notebook.

Figura 5.1. Diagrama das ferramentas do ambiente de trabalho utilizados.

Um alternativa similar ao Jupyter Notebook e que não requer configuração para ser usada é uma ferramenta desenvolida pelo Google chamada Colaboratory [10]. Para usar este ambiente, é necessário apenas ter uma conta google. No Google Colaboratory o código do seu notebook é executado em uma máquina virtual dedicada à sua conta. As máquinas virtuais são recicladas após um determinado tempo ocioso, ou caso a janela seja fechada. Ao restaurar um notebook com manipulação de arquivos, talvez seja necessário refazer o *upload* dos arquivos utilizados e executar as opções "*Runtime*" e "*Restart and run all*".

5.2.3. Introdução ao GeoPandas

GeoPandas, como já visto, estende as estruturas de dados do Pandas, uma das mais populares bibliotecas de ciência de dados, adicionando suporte para dados geoespaciais.

A estrutura de dados central do GeoPandas é `geopandas.GeoDataFrame`, uma subclasse do `pandas.DataFrame` capaz de armazenar colunas geométricas e realizar operações espaciais. As geometrias são tratadas como `geopandas.GeoSeries`, uma subclasse de `pandas.Series`. Portanto, seu GeoDataFrame é uma combinação de Series com seus dados (numéricos, booleanos, texto etc.) e GeoSeries com geometrias (pontos, polígonos e etc.), A Figura 5.2 é apresentada uma visão geral de um GeoDataFrame.

O primeiro passo, antes de ler alguns dados geoespaciais, é declarar o uso da biblioteca GeoPandas (Figira 5.3). Primeiramente, usaremos `\%matplotlib inline`, linha 1, uma configuração para permitir que os mapas apareçam diretamente no nosso no-

[7]Disponível em `https://seaborn.pydata.org/`

[8]O paradigma de programação literária busca ajudar na comunicação de programas através da alternância de texto em linguagem natural formatada, pedaços de código executáveis, e resultados de computações [Pimentel et al. 2021]

[9]Anaconda, disponível em `https://docs.anaconda.com/anaconda/install/`

[10]Disponível em: `https://colab.research.google.com/`

Figura 5.2. Estrutura de um GeoDataFrame do GeoPandas.

tebook, ao invés de serem exibidos em uma janela diferente. Além disso, importaremos o GeoPandas com o alias `gpd`, linha 2.

```
1 %matplotlib inline
2 import geopandas as gpd
```

Figura 5.3. Declaração do GeoPandas.

Assumindo que temos um arquivo contendo dados e geometria (por exemplo, GeoPackage, GeoJSON, Shapefile), podemos lê-lo facilmente usando a função *gpd.read_file*, que detecta automaticamente o tipo de arquivo e cria um GeoDataFrame. Para criar nosso primeiro mapa, utilizaremos a malha de setores censitários do estado do Piauí que provêm do Instituto Brasileiro de Geografia e Estatística (IBGE) e que pode ser baixado no seguinte endereço: `https://shorturl.at/klAF7`.

A Figura 5.4 apresenta os primeiros cinco registros do GeoDataFrame da malha de setores censitários do estado do Piauí.

```
1 setores_censitarios_pi = gpd.read_file(url_setores_censitarios_pi_ibge)
2 setores_censitarios_pi.head()
```

	CD_SETOR	CD_SIT	NM_SIT	CD_UF	NM_UF	SIGLA_UF	CD_MUN	NM_MUN	CD_DIST	NM_DIST	CD_SUBDIST	NM_SUBDIST	geometry
0	220005305000001	1	Área Urbana de Alta Densidade de Edificações	22	Piauí	PI	2200053	Acauã	220005305	Acauã	22000530500	None	POLYGON ((-41.08058 -8.21775, -41.08105 -8.218...
1	220005305000002	8	Área Rural (exclusive aglomerados)	22	Piauí	PI	2200053	Acauã	220005305	Acauã	22000530500	None	POLYGON ((-40.85118 -8.17031, -40.85080 -8.170...
2	220005305000004	8	Área Rural (exclusive aglomerados)	22	Piauí	PI	2200053	Acauã	220005305	Acauã	22000530500	None	POLYGON ((-40.78822 -8.22331, -40.78802 -8.223...
3	220005305000005	8	Área Rural (exclusive aglomerados)	22	Piauí	PI	2200053	Acauã	220005305	Acauã	22000530500	None	POLYGON ((-40.88414 -8.23197, -40.88462 -8.231...
4	220005305000006	8	Área Rural (exclusive aglomerados)	22	Piauí	PI	2200053	Acauã	220005305	Acauã	22000530500	None	POLYGON ((-40.96740 -8.31242, -40.96741 -8.312...

Figura 5.4. Malha de setores censitários do estado do Piauí.

Na Figura 5.5, é apresentado o mapa com a malha de setores censitários do Piauí, criada a partir da chamada da função `plot()`, uma das funcionalidades do Matplotlib embutidas no GeoPandas. Cada GeoDataFrame contém uma coluna especial de geometria, coluna "*geometry*", que contém todos os objetos geométricos que são exibidos quando chamamos a função `plot()`.

É possível manipular o GeoDataFrame da mesma forma que manipulamos um DataFrame no pandas e para exemplificar, iremos selecionar apenas a malha de setores censitários urbanos de Teresina, capital do Piauí, através da função *query* do pandas, linhas 4 e 5 da Figura 5.6.

```
1 setores_censitarios_pi.plot(figsize=(15,6));
```

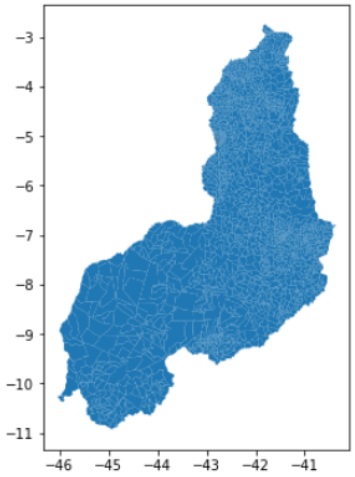


Figura 5.5. Mapa com a malha de setores censitários do Piauí.

5.2.4. Geometrias: pontos, linhas e polígonos

Como visto na seção **??**, os dados vetoriais espaciais podem consistir em diferentes tipos, e os 3 tipos fundamentais são Figura 5.7:

- **Point**: representa um único ponto no espaço.
- **Line** ("LineString"): representa uma sequência de pontos que formam uma linha.
- **Polygon**: representa uma área preenchida.

Na Figura 5.8 exemplos de criação de figuras básicas do shapely. Na linha 1 é feito o import do GeoPandas para plotar as geometrias que serão criadas, já na linha 2 é feita a importação da representação dos dados vetoriais do shapely, em seguida são criados três polígonos, linhas de 4 a 6, um polígono formados de LineString, linha 8 e um ponto, linha 10. Em seguida é criado uma GeoSeries para imprimir as geometrias criadas. Para que as linhas fiquem mais visíveis, alteraremos a paleta de cores para `tab10`.

5.2.5. Sistemas de Coordenadas

O campo de estudo que mede a forma e o tamanho da Terra é a geodésia. Segundo [Bolstad 2016], para que seja utilizado de maneira efetiva dados geoespaciais e os sistemas que derivam deles, é necessário estabelecer um entendimento claro de como os sistemas de coordenadas são definidos para a Terra, como essas coordenadas são medidas sobre a superfície curva da mesma, e por fim como são convertidas em diversas projeções para seu uso, seja manual ou digital.

Um sistema de coordenadas geográficas que é definido por coordenadas bidimensionais com base na superfície da Terra tem coordenadas X, que é a longitude e tinha as

```
setores_censitarios_urbano_teresina = (
    setores_censitarios_pi
    .query(
        "NM_MUN == 'Teresina' and \
         NM_SIT.str.contains('Urban')",
         engine='python'
    )
)
setores_censitarios_urbano_teresina.plot(figsize=(15,6));
```

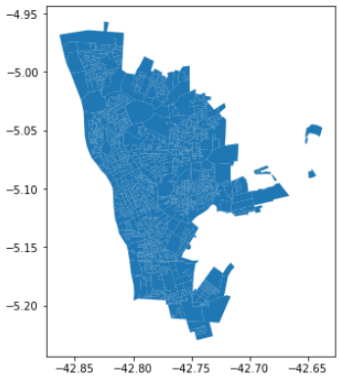


Figura 5.6. Mapa dos Setores Censitários Urbanos de Teresina - PI.

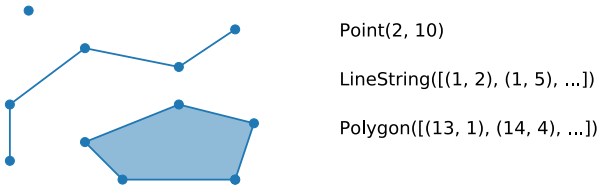


Figura 5.7. Exemplos de dados vetoriais espaciais da biblioteca shapely.

coordenadas Y, que é a latitude. Mas a terna (X,Y,Z) também contêm um valor de altura para elevação. O valor Z geralmente se refere à elevação naquele local do ponto.

As linhas de longitude têm coordenadas X entre -180 e +180 graus. O Meridiano de Greenwich (ou meridiano principal) é uma linha zero de longitude a partir da qual medimos o leste e o oeste. Longitudes positivas estão a leste do meridiano principal e as negativas estão a oeste. Na verdade, a linha zero passa pelo Observatório Real de Greenwich, na Inglaterra. Em um sistema de coordenadas geográficas, o meridiano principal é a linha que tem 0° de longitude. O Meridiano de Greenwich separa o leste do oeste da mesma forma que o Equador separa o norte do sul.

As linhas de latitudes têm valores Y que estão entre -90 e +90 graus. O equador é onde medimos o norte e o sul. Tudo ao norte do equador tem valores de latitude positivos. Considerando que, tudo ao sul do equador tem valores de latitude negativos. A Figura 5.2.5 a divisão do globo terrestre a partir do Meridiano de Greenwich e da Linha do Equador.

```
import geopandas as gpd
from shapely.geometry import Polygon, Point, LineString

p1 = Polygon([(0, 0),(1, 0),(1, 1),(0, 1)])
p2 = Polygon([(0, 0),(1, 0),(1, 1)])

p3 = Polygon([(2, 0),(3, 0),(3, 1),(2, 1)])

p4 = LineString([(0,1),(3,0),(1,1)])

p5 = Point(0.5, 0.5)

g = gpd.GeoSeries([p1, p2, p3, p4, p5])
g.plot(cmap="tab10");
```

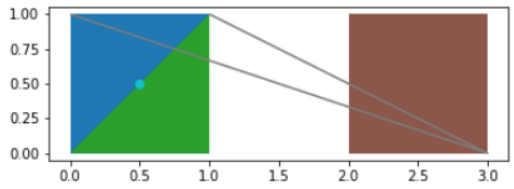


Figura 5.8. Código com exemplos de dados vetoriais espaciais da biblioteca shapely.

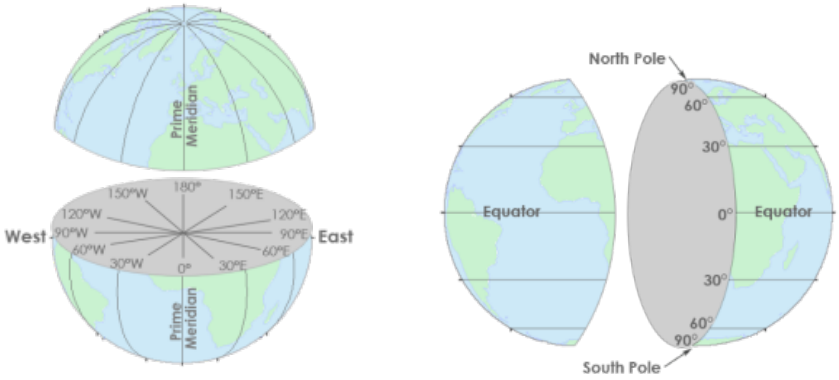
5.2.6. Projeções Espaciais

Devido à complexidade de se trabalhar com a forma real da Terra, é comumente feita uma aproximação da superfície para um modelo do do globo terrestre. Existem diversas projeções diferentes para o globo terrestre e, apesar de terem como fator comum a representação em uma superfície plana, elas variam quanto ao tipo e quanto às suas propriedades. O tipo da projeção se refere à forma geométrica utilizada para converter o globo em uma superfície plana. Comumente são utilizados três modelos para representa a superfície terrestre (Figura 5.10): projeção plana, formato esférico e formato elíptico [Bolstad 2016, Rosa and BRITO 2013, Domingues et al. 2020].

A escolha de uma projeção deve se basear na precisão desejada, no impacto sobre o que se pretende analisar e no tipo de dado disponível. A Tabela 5.1, adaptada de [Rosa and BRITO 2013] e [Domingues et al. 2020], apresenta uma análise comparativa de algumas projeções.

Para relacionar os pontos projetados a um local específico do globo terrestre é necessário adotar um Sistema de Referência de Coordenas (CRS do inglês, *Coordinate Reference Systems*). O CRS é uma forma padronizada de escrever as localizações no globo terrestre. Existe mais de um CRS, e sua escolha depende de um conjunto de fatores, como a abrangência geográfica ou mesmo em que época os dados foram coletados. Quando queremos comparar conjuntos de dados com CRSs diferentes, é importante estabelecer um CRS em comum entre eles para torná-los comparáveis.

Existem três parâmetros que, geralmente, aparecem na maioria das configurações de CRS, a projeção (`proj`), para definir a representação bidimensional do globo terrestre; elipses (`ellps`), utilizado para definir a forma da Terra; e DATUM (`datum`), que é responsável por ancorar as coordenadas da Terra, determinando as origens e a direção dos eixos.



(a) (b)

Figura 5.9. Sistema de coordenadas geográficas usado para localizar objetos sobre a Terra. (a) As linhas de longitude têm coordenadas X entre -180 e +180 graus. (b) As linhas de latitudes têm valores Y que estão entre -90 e +90 graus, adaptado de [Maling 2013].

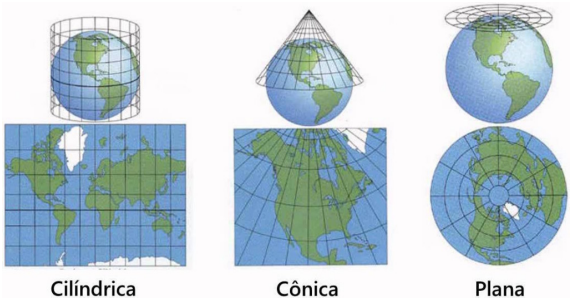


Figura 5.10. Tipos de projeções quanto a geometria de conversão [Domingues et al. 2020].

Para este minicurso, utilizaremos `latlong` e `utm` para projeções; `WGS84` e `GRS80` para elipses; e `WGS84`, utilizado pelos sistemas de GPS e `SIRGAS2000`, oficialmente utilizado pelo Brasil (pois é a melhor representação da América Latina) para o DATUM. Na Figura 5.11 temos um exemplo de configuração para CRS. Para obter uma explicação detalhada, consulte [crs].

```
crs = {
  'proj':'latlong',
  'ellps':'WGS84',
  'datum':'WGS84'
}
```

Figura 5.11. Exemplo de definição de um CRS.

Na Figura 5.12 é apresentado o CRS dos setores censitários urbanos de Teresina/PI (Figura 5.6) definidos originalmente pelo IBGE. Como é possível observar, o CRS definido é o SIRGAS 2000, pois como já mencionado, é o CRS oficialmente utilizado pelo Brasil e, portanto, é o utilizado pelo IBGE por padrão.

Já na Figura 5.13 temos um exemplo de realizar a reprojeção do CRS dos se-

Tabela 5.1. Análise comparativa das projeções.

Projeção	Classificação	Aplicações	Características da projeção
Albers	Cônica Equivalente	-cartas gerais e geográficas	-preserva áreas -garante precisão de escala -substitui com vantagens todas as outras cônicas equivalentes
Cilíndrica Conforme Equidistante Mercator	Cilíndrica Equidistante Cilíndrica Conforme	-mapas mundi -mapas em escala pequena -cartas náuticas -cartas geológicas e magnéticas -mapas mundi	-altera áreas -altera os ângulos -preserva os ângulos -mantém a forma de áreas pequenas celestes/meteorológicas
UTM	Cilíndrica Conforme	-mapeamento básico em escalas médias e grandes -cartas topográcas	-preserva ângulos -altera áreas (porém as distorções não ultrapassam 0.5%)
Gauss	Cilíndrica Conforme	-cartas topográcas -mapeamento básico em escala média e grande	-altera áreas (porém as distorções não ultrapassam 0.5%) -preserva os ângulos -similar à UTM com defasagem de 3^o de longitude entre os meridianos centrais
Estereográfica Polar	Plana Conforme	-mapeamento das regiões polares -mapeamento da Lua, Marte e Mercúrio	-preserva ângulos -preserva forma de pequenas áreas -oferece distorção de escalas

tores censitários urbanos de Teresina/PI para a projeção Mercator (`http://epsg.io/3395`), e isso é feito através da função to_crs do GeoDataFrame, linha 2. No site `http://www.spatialreference.org` é possível encontrar diversos CRSs e em vários formatos.

Uma das projeções mais utilizadas em pesquisas científicas é a projeção UTM (do inglês, *Universal Transverse Mercator*), pois possui alguns atributos que tornam a estimativa das distâncias mais precisa. A projeção UTM divide a Terra em 60 fusos ou zonas de 6^o de longitude. Para cada fuso, adota-se como superfície de projeção um cilindro transverso com eixo perpendicular ao seu meridiano central, que assume ainda o papel de longitude de origem.

Na Figura 5.14 temos uma projeção das zonas em que o Brasil está localizado (da 18 até a 25). Quando declararmos um CRS utilizando o parâmetro UTM, teremos que configurar alguns parâmetros adicionais, como a zona, o hemisfério e as unidades de

```
setores_censitarios_urbano_teresina.crs
```

```
<Geographic 2D CRS: EPSG:4674>
Name: SIRGAS 2000
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: Latin America - Central America and South America - onshore and offshore. Brazil - onshore and offshore.
- bounds: (-122.19, -59.87, -25.28, 32.72)
Datum: Sistema de Referencia Geocentrico para las AmericaS 2000
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

Figura 5.12. CRS dos setores censitários de Teresina no Piauí definidos pelo IBGE.

```
setores_censitarios_urbano_teresina = (
    setores_censitarios_urbano_teresina.to_crs("EPSG:3395")
)
ax = setores_censitarios_urbano_teresina.plot(figsize=(15,6))
ax.set_title(setores_censitarios_urbano_teresina.crs.name);
```

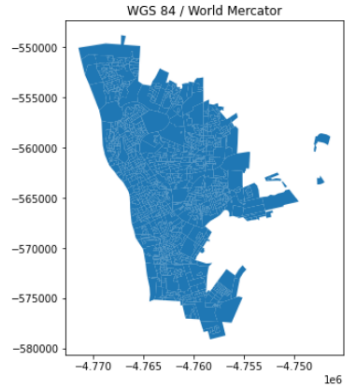


Figura 5.13. Reprojeção do CRS dos setores censitários de Teresina no Piauí para projeção Mercator.

medida que estão sendo trabalhadas. A Figura 5.15 uma possível configuração para o CRS com UTM para a cidade de Teresina/PI. Note que, para o parâmetro *zone*, escolhemos o valor 23 - que é exatamente a projeção em que está localizada a Teresina/PI. Também utilizamos o parâmetro *south*, definindo que estamos trabalhando com o hemisfério sul, e a unidade (*units*) como *m*, de "metros".

5.3. Relações e operações espaciais

Um aspecto importante dos dados geoespaciais é que podemos olhar para as relações espaciais: como dois objetos espaciais se relacionam entre si (se eles se sobrepõem, se

Figura 5.14. Representação das zonas em que o Brasil está localizado.

```
{
  'proj':'utm',
  'zone':23,
  'south':True,
  'ellps': 'GRS80',
  'units':'m',
  'no_defs': True
}
```

Figura 5.15. Exemplo, em forma de dicionário, de configuração UTM para a cidade de Teresina/PI.

cruzam, se contêm, e etc.).

As relações topológicas e teóricas de conjuntos em GIS são normalmente baseadas no modelo DE-9IM. O modelo DE-9IM expressa importantes relações espaciais que são invariantes às transformações de rotação, translação e escala. Para quaisquer dois objetos espaciais a e b, que podem ser pontos, linhas e/ou áreas poligonais, existem 9 relações derivadas de DE-9IM [Strobl 2008]:

CONTAINS: Verifica se uma representação contém completamente a outra. Inválida para a combinação ponto-linha, pois uma linha não pode estar completamente contida dentro de um ponto; porém a combinação inversa é válida.

CROSSES: Analisa se as representações se sobrepõem em algum lugar, ou seja, se as geometrias possuem pontos interiores em comum, mas não todos (uma não está contida na outra). Vale ressaltar que esta operação pode ser usada para representações com quantidade de dimensões diferentes, por exemplo uma linha e um polígono.

DISJOINT: Verifica se as representações utilizadas são disjuntas, ou seja, não compartilham nenhum ponto em comum.

EQUALS: Verifica se as duas geometrias são iguais.

INTERSECTS: Analisa se as geometrias se interceptam em algum ponto, ou seja, compartilham qualquer porção de espaço. Retorna FALSO se as geometrias forem disjuntas.

OVERLAPS: Analisa se representações de mesma dimensão se sobrepõem, mas uma não está contida na outra.

RELATE: Verifica de forma mais geral se duas representações se relacionam através de interseções nos limites, interiores ou exteriores desta, mas não são disjuntas. Esta operação é útil para verificar de uma só vez se há interseção ou se as geometrias se cruzam ou se tocam, por exemplo.

TOUCHES: Analisa se há interseção entre os limites das geometrias, mas seus interiores não se intersectam.

WITHIN: Verifica se uma geometria está dentro da outra. Representa a relação inversa de CONTAINS.

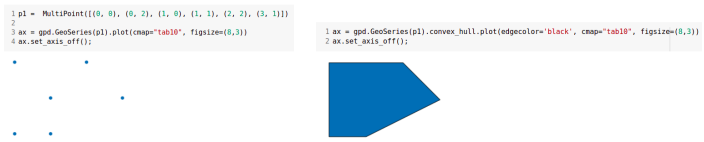
Existem também outras operações que não analisam apenas a relação entre duas geometrias, retornando VERDADEIRO ou FALSO, mas realizam operações espaciais, retornando valores ou novas geometrias como saída. Tais operações são:

BUFFER: Dada uma distância especificada pelo usuário, a operação irá gerar e retornar uma nova geometria resultante da adição de uma silhueta à geometria original (Figura 5.3).

(a) (b)

Figura 5.16. Operação BUFFER. (a) Geometria Base. (b) Geometria Resultante.

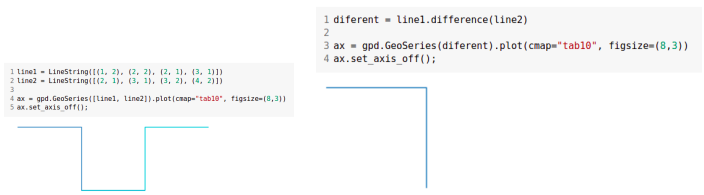
CONVEXHULL: Retorna o envoltório convexo da geometria especificada (Figura 5.3).



(a) (b)

Figura 5.17. Operação CONVEXHULL. (a) Geometria Base. (b) Geometria Resultante.

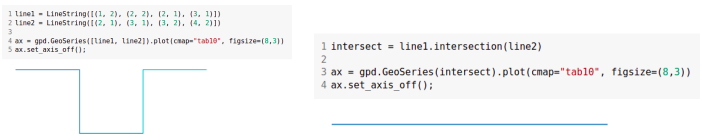
DIFFERENCE: Retorna uma geometria que contém todos os pontos que estão na representação de base mas não na geometria de comparação (Figura 5.3).



(a) (b)

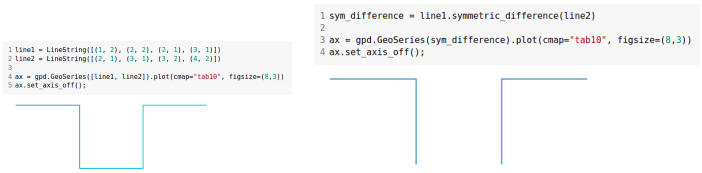
Figura 5.18. Operação DIFFERENCE. (a) Geometria Base. (b) Geometria Resultante.

INTERSECTION: Retorna a geometria que pode ser observada em ambas as representações utilizadas (Figura 5.3).



(a) (b)

Figura 5.19. Operação INTERSECTION. (a) Geometria Base. (b) Geometria Resultante.

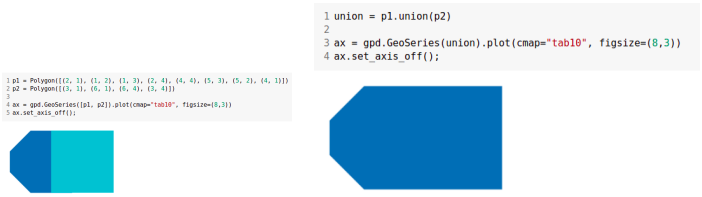


(a) (b)

Figura 5.20. Operação SYMDIFFERENCE. (a) Geometria Base. (b) Geometria Resultante.

SYMDIFFERENCE: Retorna a geometria que contém todas aquelas que não se intersectam nas representações utilizadas (Figura 5.3).

UNION: Retorna a geometria obtida com a união de todas aquelas presentes nas duas representações (Figura 5.3).



(a) (b)

Figura 5.21. Operação UNION. (a) Geometria Base. (b) Geometria Resultante.

5.4. Visualização de dados geoespaciais

A visualização geoespacial está se tornando uma maneira progressiva e sofisticada para analistas de dados ou cientistas transmitirem suas análises de forma eficiente e como já visto na Figura 5.4, o GeoPandas também pode traçar mapas, para que possamos verificar como nossas geometrias se parecem no espaço. O método principal responsável por isso é o `plot()`, uma interface de alto nível para a biblioteca *matplotlib* para construir mapas. Vale ressaltar que, em geral, todas as opções que podem ser passadas para `pyplot` do *matplotlib*, como opções de estilo por exemplo, podem ser passadas para o método `plot()`.

Vamos agora determinar os centroides dos setores censitários urbanos de Teresina/PI, criado na Figura 5.6, linhas de 1 a 3 da Figura 5.22 e em seguida criar duas

camadas, uma para os setores censitários, linha 4 da Figura 5.22, e outra para os centroides, plotando-os de preto para destacar, linha 5 da Figura 5.22. O resultado dessa operação pode ser observado na Figura 5.23

```
1 setores_censitarios_urbano_teresina['centroid'] = (
2     setores_censitarios_urbano_teresina['geometry'].centroid
3 )
4 ax = setores_censitarios_urbano_teresina['geometry'].plot(figsize=(23,15))
5 setores_censitarios_urbano_teresina['centroid'].plot(ax=ax, color="black");
```

Figura 5.22. Cálculo dos centroides dos setores censitários urbanos de Teresina/PI.

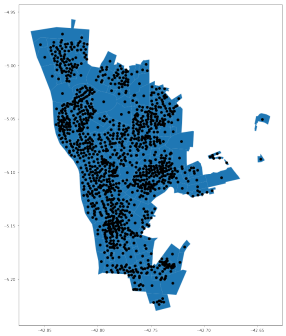


Figura 5.23. Mapa com os setores censitários urbanos de Teresina/PI com seus respectivos centroides.

Agora, vamos calcular a área de cada setor censitário urbano de Teresina/PI, linhas de 1 a 3 da Figura 5.24, e em seguida plotar essa área calculado com uma legenda, linha 4 da Figura 5.24. O resultado dessa operação pode ser observado na Figura 5.25

```
1 setores_censitarios_urbano_teresina['area'] = (
2     setores_censitarios_urbano_teresina['geometry'].area
3 )
4 setores_censitarios_urbano_teresina.plot('area', legend=True, figsize=(23,15))
```

Figura 5.24. Cálculo da área dos dos setores censitários urbanos de Teresina/PI.

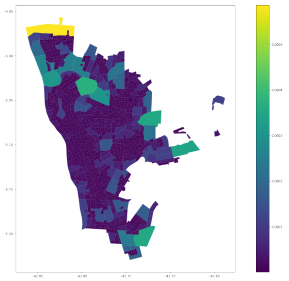


Figura 5.25. Mapa dos setores censitários urbanos de Teresina/PI graduados pela área.

Outra forma de visualização de dados geoespaciais é através de bibliotecas de visualizações de mapas interativos baseadas na web. Existem várias bibliotecas com essa finalidade, segue alguns pacotes com um exemplo para cada um:

- **Bokeh:** `https://bokeh.pydata.org/en/latest/docs/gallery/texas.html`
- **GeoViews** (outra interface para Bokeh/Matplotlib): `http://geo.holoviews.org`
- **Altair:** `https://altair-viz.github.io/gallery/choropleth.html`
- **Plotly:** `https://plot.ly/python/#maps`

Para criação dos nossos mapas interativos, utilizaremos a biblioteca Folium[11], uma biblioteca Python que possibilita a visualização geográfica interativa de dados espaciais através do *Leaflet.js*[12].

Para criar um mapa interativo com o Folium, basta importá-lo, linha 1 da Figura 5.26, e em seguida chamar o método `Map()` passando a localização em termos de latitude e longitude como parâmetro, linha 3 da Figura 5.26. No exemplo da Figura 5.26, foi utilizado às coordenadas de Teresina/PI[13] e o resultado pode ser observado na Figura 5.27.

```
1 import folium
2
3 mapa_teresina = folium.Map(location=[-5.088889, -42.801944], zoom_start = 12)
4 mapa_teresina
```

Figura 5.26. Código para criação de um mapa interativo com o Folium em Python.

Figura 5.27. Mapa interativo de Teresina/PI, plotado com o Folium em Python.

Um parâmetro interessante de mudar é o tipo de gráfico, através do parâmetro *tiles*. Vale ressaltar que um *tileset* é uma coleção de dados *raster* e vetoriais divididos em uma grade uniforme de ladrilhos quadrados. Cada *tileset* tem uma maneira diferente de

[11] Disponível em `http://python-visualization.github.io/folium/`

[12] Leaflet.js é uma biblioteca JavaScript de código aberto usada para construir mapas interativos e *mobile-friendly*. Ela utiliza dados do *OpenStreetMaps* para construir a projeção de mapas detalhados contendo informações de vias e demarcações de locais e transportes públicos [Domingues et al. 2020, Agafonkin 2014].

[13] Coordenadas de Teresina/PI disponível em `https://geohack.toolforge.org/geohack.php?pagename=Teresina&params=5_05_20_S_42_48_07_W_type:city_region:BR`

representar dados no mapa. O Folium nos permite criar mapas com diferentes *tiles* como *Stamen Terrain*, *Stamen Toner*, *Stamen Water Color*, *CartoDB Positron*. Por padrão, Folium define o *OpenStreetMap* como *tile* padrão. Na Figura 5.28 temos o mapa de Teresina/PI com o *tiles Stamen Terrain* que mostra o relevo.

```
1 mapa_teresina = folium.Map(
2     location=[-5.088889, -42.801944],
3     tiles= 'Stamen Terrain'
4 )
5 mapa_teresina
```

Figura 5.28. Mapa interativo de Teresina/PI com o *tiles Stamen Terrain*.

Como agora sabemos que cada *tileset* fornece informações de uma maneira diferente e serve a um propósito diferente, podemos sobrepô-los para obter mais informações apenas traçando um único mapa. Podemos fazer isso adicionando diferentes camadas de blocos a um único mapa. Na Figura 5.29 temos o mapa de Teresina/PI com os *tiles Stamen Terrain*, *Stamen Toner*, *Stamen Water Color*, *CartoDB Positron*, *Carto Dark Matter*, além do *OpenStreetMap*, adicionado através do método `TitleLayer()`, da linha 1 até a linha 5.

```
1 folium.TileLayer('Stamen Terrain').add_to(mapa_teresina)
2 folium.TileLayer('Stamen Toner').add_to(mapa_teresina)
3 folium.TileLayer('Stamen Water Color').add_to(mapa_teresina)
4 folium.TileLayer('cartodbpositron').add_to(mapa_teresina)
5 folium.TileLayer('cartodbdark_matter').add_to(mapa_teresina)
6 folium.LayerControl(collapsed=False).add_to(mapa_teresina)
7
8 mapa_teresina
```

Figura 5.29. Mapa interativo de Teresina/PI com vários *tiles*.

É possível observar na Figura 5.29 que foi adicionado cinco camadas de *tiles* diferentes a um único mapa e agora se tem 6 camadas diferentes de conjuntos de *tiles*. Também foi adicionado ao mapa o `LayerControl()`, linha 6 da Figura 5.29, que fornece um ícone no canto superior direito do mapa para alternar entre as diferentes camadas.

Criando Marcadores

Marcadores são um dos itens mais utilizados para marcar uma localização em um mapa. Por exemplo, quando se usa o Google Maps para navegação, é marcada a localização de origem por um marcador e o destino é marcado por outro marcador. Vale ressaltar que os marcadores estão entre as coisas mais importantes e úteis em um mapa interativo.

Folium fornece uma classe `folium.Marker()` para criar marcadores em um mapa interativo. Basta passar a latitude e longitude do local, mencionar um *pop-up* e um *tooltip* e adicioná-lo ao mapa. A plotagem de marcadores é um processo de duas etapas. Primeiro, você precisa criar um mapa básico no qual seus marcadores serão colocados e, em seguida, adicionar seus marcadores a ele. Na Figura 5.30 é definido um *array* com as coordenadas da Ponte Estaiada João Isidoro França[14], um dos mais importantes pontos turísticos da capital piauiense, linha 1, e em seguida é criado um *marker* com essa coordenada e adicionado ao mapa de Teresina, linhas de 3 a 5. O resultado do código da Figura 5.30 pode ser observado na Figura 5.31.

```
1 coordenadas_ponte_estaiada = [-5.069861, -42.802472]
2
3 folium.Marker(
4   coordenadas_ponte_estaiada, popup="<i>Ponte Estaiada</i>"
5 ).add_to(mapa_teresina)
6
7 mapa_teresina
```

Figura 5.30. Código para criação do *marker* nas coordenadas da Ponte Estaiada em Teresina/PI.

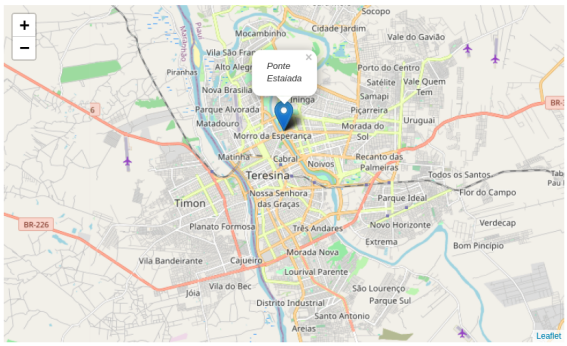


Figura 5.31. Mapa interativo de Teresina/PI com o *marker* nas coordenadas da Ponte Estaiada.

É possível também personalizar a aparência de um *marker*. O Folium fornece a classe `folium.Icon()` que pode ser usada para criar ícones personalizados para *markers*. O construtor da classe `icon()` recebe três parâmetros - `color`, `prefix` e `icon`, a cor, prefixo e ícone respectivamente. O parâmetro `color` é usado para alterar a cor do *marker*, `prefix` é usado para selecionar a origem do ícone (**fa** para *Fontawesome* e **glificon** para *Glyphicons*) e o `icon` é usado para selecionar o nome do ícone. Na Figura 5.32 temos a criação de dois *markers* personalizados para a Universidade Estadual do Piauí, da linha 1 até a linha 5, e para a Universidade Federal do Piauí, da linha 7 até a

[14]Coordenadas disponível em `https://shorturl.at/dryAF`

linha 11, e ambos adicionado ao mapa de Teresina/PI, criado anteriormente. O resultado do código da Figura 5.32 pode ser observado na Figura 5.33.

```
1 folium.Marker(
2     location=[-5.0778331015812, -42.82593947402195],
3     popup="UESPI",
4     icon=folium.Icon(color="green", prefix='fa',icon='university'),
5 ).add_to(mapa_teresina)
6
7 folium.Marker(
8     location=[-5.06143163033608, -42.79473533169222],
9     popup="UFPI",
10    icon=folium.Icon(color="red", prefix='glyphicon',icon='home'),
11 ).add_to(mapa_teresina)
12
13 mapa_teresina
```

Figura 5.32. Código com a criação de *markers* personalizados para as universidades de Teresina/PI.

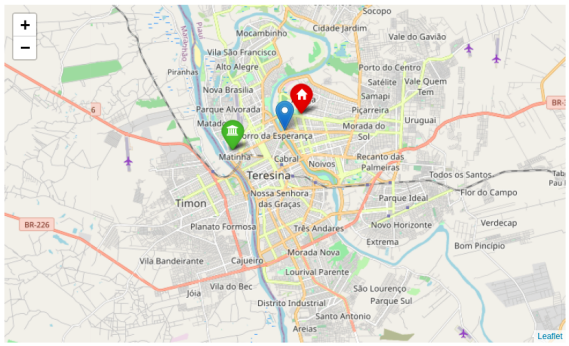


Figura 5.33. Mapa interativo com os *markers* personalizados para as universidades de Teresina/PI.

Antes de continuar a explorar algumas das funcionalidades mais comuns do Folium, vamos gerar um GeoDataFrame com 100 pontos aleatório contidos nos setores censitários urbanos de Teresina/PI, criados na Figura 5.6. Na Figura 5.34 temos um possível código para geração desse GeoDataFrame.

```
1  import random
2  from shapely.geometry import Point
3
4  teresina = setores_censitarios_urbano_teresina.dissolve()
5  minx, miny, maxx, maxy = teresina.bounds.iloc[0]
6
7  cont = 0
8  colecao_de_pontos = []
9
10 while cont < 100:
11     p = Point(random.uniform(minx, maxx), random.uniform(miny, maxy))
12     if teresina.contains(p).iloc[0]:
13         colecao_de_pontos.append({
14             'latlong': (p.y, p.x),
15             'geometry': p
16         })
17         cont += 1
18
19 pontos_aleatorios_teresina = gpd.GeoDataFrame(colecao_de_pontos)
20 pontos_aleatorios_teresina.crs = setores_censitarios_urbano_teresina.crs
```

Figura 5.34. Exemplo de código para geração de um GeoDataFrame com 100 pontos aleatório contidos nos setores censitários urbanos de Teresina/PI.

Na linha 1 da Figura 5.34 temos a importação da função *built-in* Python para geração de números aleatório segundo uma função uniforme de probabilidade. Já linha 2

é feito a importação da classe *Point* do *shapely* para criar as geometrias para o GeoDataFrame. Na linha 4 é feita a agregação das geometrias do setores censitários através da função `dissolve()` do GeoPandas, em seguida extraímos os limites da agregação resultante dos setores censitários através da função `bounds`. Nas linhas de 10 até 17 temos laço de repetição responsável por gerar um ponto aleatório a partir dos limites da agregação dos setores censitários e verificar se ele está contido dentro dos limites dos setores censitários urbanos de Teresina/PI. Após é criado um novo GeoDataFrane com esse conjuntos de pontos aleatórios, linha 19, com o mesmo sistema de coordenadas dos setores censitários, linha 20.

De posse do GeoDataFrame da Figura 5.34, podemos adicioná-lo ao mapa interativa de Teresina/PI criado na Figura 5.27, através da classe `GeoJson` do Folium (Figura 5.35).

```
1 folium.GeoJson(
2     pontos_aleatorios_teresina,
3     marker= folium.Marker(
4         icon=folium.Icon(
5             color="black",
6             prefix='fa',
7             icon='bug'
8         )
9     )
10 ).add_to(mapa_teresina)
11
12 mapa_teresina
```

Figura 5.35. Adição do GeoDataFrame com 100 pontos aleatório ao mapa interativo de Teresina/PI.

O primeiro parâmetro, obrigatório, do construtor da classe `GeoJson` do Folium é os dados que queremos visualizar, no nosso exemplos, o conjunto de pontos aleatórios criados na Figura 5.34, além de um *marker* customizado para eles. O resultado do código da Figura 5.35 pode ser observado na Figura 5.36.

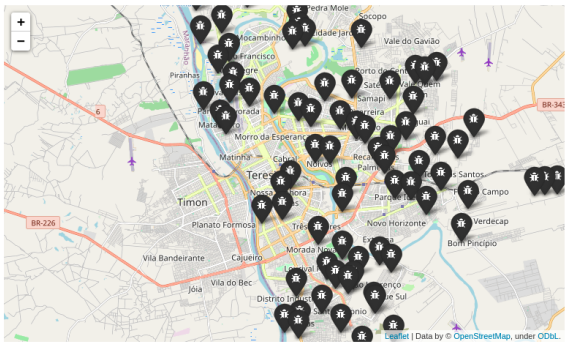


Figura 5.36. Mapa interativo de Teresina/PI com 100 *markers* criados aleatoriamente.

Como pode ser observado na Figura 5.36, os marcadores parecem estar empilhados e um pouco bagunçados. Uma forma de organizar os *markers* é através de *clusters* de *markers*. O Folium disponibiliza um *plugin* com essa finalidade, o **MarkerCluster**. A Figura 5.37 é apresentado como utilizar o *plugin* **MasterCluster** com o nosso GeoDataFrame criado na Figura 5.34.

```
1  from folium.plugins import MarkerCluster
2
3  marker_cluster = MarkerCluster().add_to(mapa_teresina)
4
5  folium.GeoJson(
6      pontos_aleatorios_teresina,
7      marker= folium.Marker(
8          icon=folium.Icon(
9              color="black",
10             prefix='fa',
11             icon='bug'
12         )
13     )
14 ).add_to(marker_cluster)
15
16 mapa_teresina
```

Figura 5.37. Exemplo de utilização do *plugin* MasterCluster do Folium.

Na linha 1 da Figura 5.37 é feita a importação do *plugin* **MasterCluster**, na linha 3 é criado um objeto **MasterCluster** e adicionamos ele ao mapa interativo de Teresina/PI. Ao invés de adicionar o nosso GeoDataFrame de pontos aleatórios direto no mapa, como é feito na Figura 5.35, adicionamos o nosso GeoDataFrame de pontos ao objeto **MasterCluster** criado na linha 3. O resultado do código da Figura 5.37 pode ser observado na Figura 5.38.

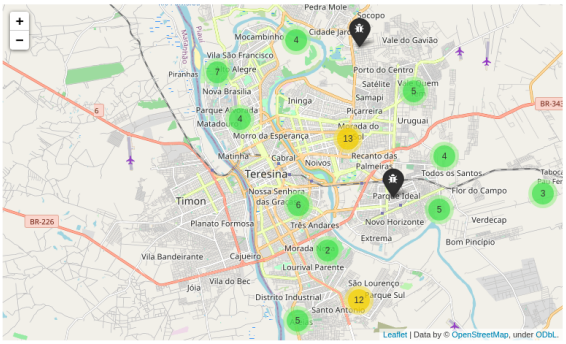


Figura 5.38. Mapa interativo de Teresina/PI com 100 *markers* criados aleatoriamente agrupados.

Criando *HeatMaps*

É possível também implementar *HeatMaps*, ou mapas de calor, usando Folium. Um *HeatMap* é uma representação gráfica de dados que usa um sistema de codificação de cores para representar diferentes valores. Isso é útil para monitorar a intensidade das estatísticas regionais com mais facilidade em uma determinada região por exemplo.

Para criação de um *HeatMap* o Folium disponibiliza um *plugin* chama justamente *HeatMap* e para exemplificar, vamos criar um *HeatMap* com o GeoDataFrame criado na Figura 5.34. Na Figura 5.39 temos o código para essa finalidade. Na linha 1 é feita a importação do *plugin* do Folium *HeatMap* responsável por criar o mapa de calor. Já na linha 3 é feita a criação do *HeatMap* passando como parâmetro uma lista de pontos na forma `[lat, lng]` que se deseja plotar, também é possível passar uma lista na [lat, lng, weight]) ou fornecer um `numpy.array (n,2)` ou `(n,3)`. O resultado do código da Figura 5.39 pode ser observado na Figura 5.40.

```python
from folium.plugins import HeatMap

HeatMap(pontos_aleatorios_teresina['latlong'].tolist()).add_to(mapa_teresina)
mapa_teresina
```

Figura 5.39. Código para criação de um *HeatMap*.

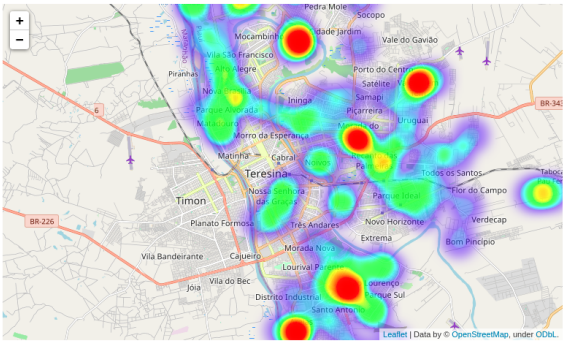


Figura 5.40. Mapa de calor de Teresina/PI com 100 *markers* criados aleatoriamente.

Criando Mapas Coropléticos

Muitas vezes lidamos com dados espaciais cuja a localização está associada a áreas delimitadas por polígonos. Isso ocorre na maioria das vezes quando não se dispõe da localização exata dos eventos, sendo portanto agregados por municípios, bairros ou setores censitários. A forma usual de apresentação de dados agregados por áreas é através do **mapas coropléticos** ou coloridos com o padrão espacial do fenômeno [Monteiro et al. 2004].

É possível criar um mapa coroplético através do parâmetro `style_function` da classe `GeoJson` do Folium, fornecendo uma função que possa especificar um estilo dependendo do recurso que se queira mapear. Para exemplificar a criação de mapas coropléticos, utilizaremos a coluna `area` do GeoDataFrame de setores censitários urbanos de Teresina/PI criada na Figura 5.24. A Figura 5.41 é apresentado uma possível solução.

```python
from branca.colormap import linear

colormap = linear.YlOrRd_04.scale(
    setores_censitarios_urbano_teresina['area'].min(), setores_censitarios_urbano_teresina['area'].max()
)

colormap.caption = "Escala de Cor para Área dos Setores Censitários"
colormap.add_to(mapa_teresina)

area_dict = setores_censitarios_urbano_teresina.set_index("CD_SETOR")["area"]

folium.GeoJson(
    setores_censitarios_urbano_teresina,
    name="area",
    style_function=lambda feature: {
        "fillColor": colormap(area_dict[feature['properties']["CD_SETOR"]]),
        "color": "black",
        "weight": 1,
        "dashArray": "5, 5",
        "fillOpacity": 0.3,
    },
).add_to(mapa_teresina)

mapa_teresina
```

Figura 5.41. Código de exemplo para criação de um mapa coroplético.

Inicialmente é feito a importação da função que utilizaremos para mapear um valor para uma cor RGB (da forma `#RRGGBB`), Nas linhas de 3 a 5 criamos a nossa paleta de cores de amarelo até vermelho, com o limite inferior como sendo a menor área do e o limite superior como sendo a maior área, em seguida é definido uma legenda para a nossa

paleta, linha 7 e depois ela é adicionada ao nosso mapa interativo de Teresina/PI, linha 8. Já na linha 10 é criado um dicionário para mapear o setor censitário à sua respectiva área. Entre as linhas 15 e 21, é definida uma função anônima que mapeia para cada setor censitário (*feature* do GeoJSON) um dicionário com a cor RBB e outros parâmetros de definição do nosso mapa coroplético. O resultado do código da Figura 5.41 pode ser observado na Figura 5.42.

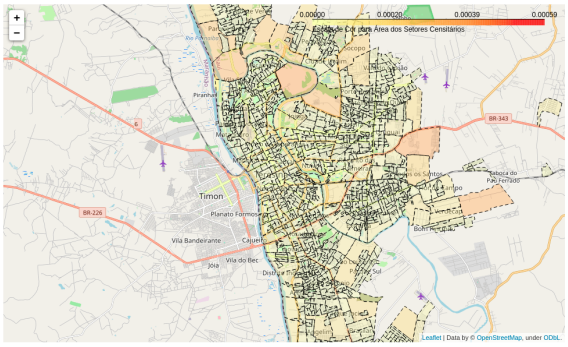


Figura 5.42. Mapa coroplético dos setores censitários urbanos de Teresina/PI a partir da sua respectiva área.

O Folium também disponibiliza uma classe `Choropleth` para criação de mapas corropléticos de forma mais rápida. Assim como na classe `GeoJson`, é possível fornecer a ela um nome de arquivo, um dicionário ou um GeoDataFrame. A Figura 5.43 apresenta um exemplo de como usá-la, utilizando a coluna `area` do GeoDataFrame de setores censitários urbanos de Teresina/PI do exemplo anterior.

```
1  folium.Choropleth(
2      geo_data = setores_censitarios_urbano_teresina,
3      data = setores_censitarios_urbano_teresina[["CD_SETOR", "area"]],
4      columns = ["CD_SETOR", "area"],
5      key_on = "feature.properties.CD_SETOR",
6      fill_color = "YlOrRd",
7      fill_opacity = 0.3,
8      legend_name = "Escala de Cor para Área dos Setores Censitários",
9  ).add_to(mapa_teresina)
10
11 mapa_teresina
```

Figura 5.43. Código de exemplo para criação de um mapa coroplético por meio da classe Choropleth.

Na linha 2 da Figura 5.43 o parâmetro `geo_data` define a origem das geometrias que serão utilizadas, no exemplo utilizamos GeoDataFrame com os setores censitários urbanos de Teresina/PI. Já o parâmetro `data` é definido um novo DataFrame com o código do setor censitário, que será a chave para vinculação ao GeoJSON, e o valor da área, como dado para o mapa coroplético, linha 3 da Figura 5.43. O parâmetro `colmns` é especificado a chave de vinculação dos dados e a coluna de dados que no neste exemplo é a área, linha 4 da Figura 5.43. O parâmetro `key_on` é definida a variável do `geo_data` do GeoJSON para vincular os dados, linha 5 da Figura 5.43. Note que esse parâmetro deve começar com '*feature*' e estar em notação de objeção *JavaScript*, como por exemplo '*feature.id*' ou '*feature.properties.statename*'. Os demais parâmetros utilizados no construtor da classe define a cor da área a ser preenchida, sua opacidade e a legenda da paleta de co-

res utilizada, respectivamente. O resultado do código da Figura 5.43 pode ser observado na Figura 5.44.

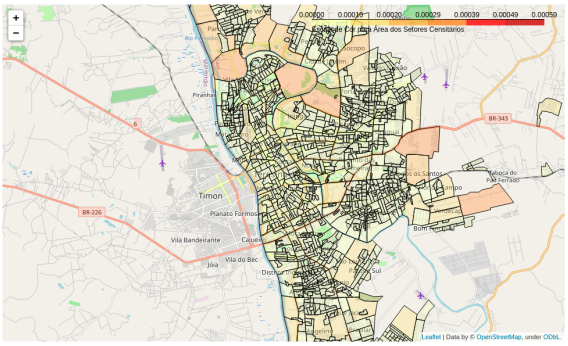


Figura 5.44. Mapa coroplético dos setores censitários urbanos de Teresina/PI a partir da sua respectiva área criado com a classe Choropleth.

O Folium disponibiliza diversos outros recursos e *plugins* e demonstrar todos eles está fora do escopo deste minicurso, no entanto, no seguinte link `https://shorturl.at/mvEIP` é possível encontrar uma lista de vários *notebooks* de exemplos com todos eles.

5.5. Considerações finais

Este capítulo apresentou os conceitos gerais sobre a Análise de Dados Geoespaciais através da linguagem de programação Python e alguns exemplos de como extrair conhecimento e valor por meio de bibliotecas como `pandas`, `geopandas`, `shapely`, `pyproj`, `matplotlib`, `cartopy`, dentre outras.

A Seção 5.2, foi apresentado alguns conceitos sobre um dos formatos mais utilizados em aplicações de análise geoespaciais, Dados Vetoriais, as ferramentas e pacotes Python necessários para começar a análise de dados geoespaciais, além dos conceitos centrais da utilização de GeoDataFrames. Já a Seção 5.3 apresentou as relações e operações para dois objetos espaciais. Por fim, a Seção 5.4 discutiu as formas de visualização que podem ser utilizadas no estudo e na aplicação de dados geoespaciais. Muitas outros aspectos, ferramentas e técnicas de análise de dados geoespaciais poderiam ter sido abordados, porém, como é apenas uma introdução esses e outros pontos ficará para trabalhos futuros.

Agradecimentos

Este trabalho foi realizado com apoio financeiro da CAPES, CNPq, Fundação de Amparo à Pesquisa e ao Desenvolvimento Científico e Tecnológico do Maranhão (FAPEMA) e Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo 2020/16578-5. Agradecemos também ao ICMC-USP e ao LCR por oferecerem a infraestrutura necessária para este estudo.

Referências

[crs] Coordinate reference systems. https://docs.qgis.org/2.8/en/docs/gentle_gis_introduction/coordinate_reference_systems.html. Acessado em: 10-09-2021.

[Agafonkin 2014] Agafonkin, V. (2014). Leaflet: an open-source javascript library for mobile-friendly interactive maps. *Accessed September*, 21:2020.

[Bolstad 2016] Bolstad, P. (2016). *GIS fundamentals: A first text on geographic information systems*. Eider (PressMinnesota).

[Coelho 2017] Coelho, A. S. (2017). Introdução a análise de dados com python e pandas. *Anais Eletrônicos ENUCOMP*, pages 862–876.

[Domingues et al. 2020] Domingues, A., Silva, F., Santos, L., Souza, R., Coimbra, G., and Loureiro, A. A. F. (2020). Dados geoespaciais: Conceitos e técnicas para coleta, armazenamento, tratamento e visualização. *Sociedade Brasileira de Computação*.

[Lawhead 2015] Lawhead, J. (2015). *Learning geospatial analysis with Python*. Packt Publishing Ltd.

[Lopes et al. 2019] Lopes, G. R., Almeida, A. W. S., Delbem, A. C., and Toledo, C. F. M. (2019). Introdução à análise exploratória de dados com python. In *Minicursos ERCAS ENUCMPI 2019*, pages 160–176, Porto Alegre, RS, Brasil. SBC.

[Maling 2013] Maling, D. H. (2013). *Coordinate systems and map projections*. Elsevier.

[Monteiro et al. 2004] Monteiro, A. M. V., Câmara, G., Carvalho, M., and Druck, S. (2004). Análise espacial de dados geográficos. *Brasília: Embrapa*.

[Pimentel et al. 2021] Pimentel, J. F., Oliveira, G. P., Silva, M. O., Seufitelli, D. B., and Moro, M. M. (2021). Ciência de dados com reprodutibilidade usando jupyter. *Sociedade Brasileira de Computação*.

[Rosa and BRITO 2013] Rosa, R. and BRITO, J. L. S. (2013). Introdução ao geoprocessamento. *UFU: Apostila. Uberlândia*.

[Shen 2014] Shen, H. (2014). Interactive notebooks: Sharing the code. *Nature News*, 515(7525):151.

[Strobl 2008] Strobl, C. (2008). Dimensionally extended nine-intersection model (de-9im).

[Zheng et al. 2014] Zheng, Y., Capra, L., Wolfson, O., and Yang, H. (2014). Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):1–55.

Capítulo

6

Realidade Aumentada no contexto de Computação Ubíqua: conceitos, características e ferramentas da plataforma Android

Joel Machado Pires, João Soares de Oliveira Neto

Abstract

This chapter presents the main characteristics of Ubiquitous Computing and Augmented Reality and how these technologies can be used together to improve the user experience in urban space. Moreover, we discuss the features offered by the Android platform for the creation of Augmented Reality applications based on the location of users and objects placed in the environment where the users are. To illustrate the issues covered in this chapter, we present the process of designing, modeling and implementing the Intelligent Campus app - a set of resources and services based on Augmented Reality to provide information about buildings and other objects arranged on a university campus.

Resumo

Este capítulo apresenta as características principais da Computação Ubíqua e Realidade Aumentada e de que maneira estas tecnologias podem ser utilizadas em conjunto visando melhorar a experiência do usuário no espaço urbano. Além disso, são discutidos os recursos oferecidos pela plataforma Android para a criação de aplicativos de Realidade Aumentada baseados na localização dos usuários e de objetos presentes no ambiente. Para ilustrar os assuntos abordados neste capítulo, é detalhado o processo de concepção, modelagem e implementação do app Campus Inteligente, um conjunto de recursos e serviços baseados em Realidade Aumentada para oferecer informações sobre prédios e outros objetos presentes num campus universitário.

6.1. Introdução

Ao prever um futuro em que a Computação estaria presente em todos os lugares, ambientes e contextos, além de estar disponível a todas as pessoas a todo momento, Weiser (1991) inspirou o surgimento de várias tecnologias e inovações necessárias para

que este universo projetado se tornasse realidade. Conexões sem fio, soluções para comunicação móvel, processamento distribuído e interfaces naturais - baseadas em toque, voz e gestos - foram alguns dos avanços iniciais. Todavia, os anos seguintes testemunharam o surgimento de uma sucessão de novos conceitos e ferramentas concebidas para aumentar a sensação de onipresença.

Se algumas áreas/tecnologias fortaleceram a Computação Ubíqua e podem ser consideradas como habilitadoras - como é o caso Computação em Nuvem, Computação Vestível, GPS e sistemas de localização, sensores e tecnologias de identificação etc. - outras áreas e tecnologias sofreram forte impacto, foram impulsionadas ou ganharam novos direcionamentos graças à Computação Ubíqua, como é o caso de Interação Humano-computador, Hardware, Arquitetura de Sistemas, Redes Sociais, Comércio Eletrônico, Realidade Virtual e também Realidade Aumentada (RA), entre outras.

A possibilidade de o indivíduo continuar conectado às redes de computadores mesmo em movimento, modificou na verdade a forma como interagimos com o ambiente e entre nós mesmos (REID et al., 2016). A miniaturização dos equipamentos permitiu o acesso a mais informação e a uma variedade de serviços dentro do perímetro urbano ou até mesmo na zona rural (REID et al., 2016): saúde, educação, monitoramento e segurança, localização e entretenimento, por exemplo. Nesse contexto, a RA pode ser vista como uma tecnologia urbana, ou seja, uma solução tecnológica concebida para melhorar a qualidade de vida urbana, em termos de energia, transporte, planejamento urbano, segurança, acesso a serviços, entre outros (JIANG et al., 2016).

RA é a combinação entre o mundo real e objetos virtuais, com os quais podem-se interagir em tempo real (DUBOIS; NIGAY, 2000; ZHOU; DUH; BILLINGHURST, 2008). Essa tecnologia tornou-se mais popular com a difusão de dispositivos móveis, cujas câmeras capturam imagens do mundo real e sobre as quais são inseridos objetos que adicionam informações à imagem real, ou seja, aumentam a realidade. As iniciativas de Cidades Inteligentes (CI) têm se beneficiado bastante do uso de RA. Tais iniciativas buscam melhorar a qualidade da governança de espaços urbanos e aumentar a qualidade de vida dos cidadãos a partir do oferecimento de tecnologias urbanas digitais (ALBINO; BERARDI; DANGELICO, 2015).

Em Cidade Inteligentes, a Realidade Aumentada pode ser usada para o reconhecimento de objetos no espaço urbano; inclusão de informações sobre prédios, praças, ruas etc.; mobilidade urbana, definição de rotas e navegação interna/externa; construção civil e arquitetura; poluição e meio ambiente; Tecnologia Assistiva e ferramentas de inclusão; manutenção e reparo; entretenimento; turismo; esporte e lazer; entre outros setores (KAJI et al., 2018; KAVAKLI, 2015; RASHID et al., 2017).

O objetivo deste capítulo é evidenciar e discutir os conceitos de RA e Computação Ubíqua, suas características e apresentar os recursos oferecidos pela plataforma *Android* para a criação de aplicativos de RA baseados na localização dos usuários e de objetos presentes no espaço urbano. Para ilustrar os assuntos abordados ao longo do minicurso, será detalhado todo o processo de concepção, modelagem e implementação do app *Campus Inteligente*, um conjunto de recursos e serviços baseados em RA para melhorar a experiência dos usuários de um campus universitário.

A Seção 6.2 aborda o tema Computação Ubíqua, enquanto que a Seção 6.3 apresenta os principais conceitos relacionados com Realidade Aumentada. Em seguida,

na Seção 6.4, é discutido o estudo de caso: a criação de uma aplicação de Realidade Aumentada para um Campus Inteligente. A Seção 6.5 traz as considerações finais e aponta possibilidades de continuação de estudos nos temas abordados neste capítulo.

6.2. Computação Ubíqua

A Computação Ubíqua trata do desenvolvimento de soluções, na área de tecnologia da informação, que visam interagir com o usuário por meio de dispositivos que podem ser distribuídos no ambiente e que estão presentes no dia a dia das pessoas. Este paradigma computacional se contrapõe a ambientes restritos e controlados, como domicílios, escritórios e empresas, em que a conexão a redes de computadores se dá por meio de cabos ou conexões sem fio de curta distância. No contexto ubíquo, o usuário tem maior flexibilidade para se deslocar por longas distâncias, permanecendo o tempo todo conectado a redes de computadores (KUMAR, 2009).

Já que o indivíduo encontra-se constantemente imerso em ambientes que entrelaçam o real e o virtual e dada a onipresença computacional, a expectativa é que os dispositivos físicos e suas interfaces estejam diluídos, imperceptíveis e invisíveis de tão próximos que estão do cotidiano humano (WEISER, 1999). Embora a visão de Weiser tenha sido fortemente baseada em computadores, ele mesmo já sinalizava para o surgimento de dispositivos vestíveis - ou *wearable devices,* em inglês. Tais dispositivos ganharam grande notoriedade a partir do lançamento de vários modelos e aplicações, que foram amplamente adotados para fins gerais e específicos, tais como monitoramento de saúde, entretenimento, segurança, tecnologia assistiva, identificação etc.

Garantir o grau de transparência sugerido para sistemas ubíquos - para a comunicação e mobilidade, por exemplo - exige um enorme investimento no desenvolvimento de sistemas distribuídos, interface de usuários e projeto de interação. As características que diferenciam sistemas ubíquos de outros sistemas é que eles são executados em ambientes personalizáveis, centrados no usuário e cuja interação acontece de maneira menos perceptível. Além disso, são sistemas sensíveis ao ambiente, ou ao contexto em que estão inseridos. Quanto mais conscientes do contexto ao seu redor, mais estes sistemas são capazes de se adaptarem, atuarem e controlarem o ambiente. De maneira resumida, os requisitos de sistemas ubíquos são (POSLAD, 2009; WEISER, 1999):

- Os dispositivos precisam estar conectados à rede, distribuídos e acessíveis de maneira transparente;
- A interação humano-computador precisa estar o mais implícita, oculta e camuflada possível;
- Necessitam ser sensíveis ao contexto, a fim de otimizar o seu funcionamento no ambiente em que se encontra;
- Os dispositivos podem funcionar de maneira autônoma, sem interferência humana e auto-gerenciáveis;
- Podem lidar com uma multiplicidade de ações e interações dinâmicas. Isso pressupõe alguma forma de Inteligência Artificial a fim de tratar:
    - interações incompletas e não-determinísticas;
    - cooperação e competição entre membros da mesma equipe;

- interações sofisticadas a partir do compartilhamento de contexto, semântica e metas.

Poslad (2009) sintetiza que os elementos de um sistema ubíquo podem ser agrupados em dois *clusters*: (a) contexto pessoal, econômico e social centrado no ser humano; e, (b) uma camada física formada por um ecossistema de elementos vivos (seres humanos) e inanimados (dispositivos e hardware). Para o autor, tais elementos não são mutuamente exclusivos; pelo contrário: eles se sobrepõem e precisam ser combinados.

Muitas das atividades humanas podem ser automatizadas por máquinas em sistemas ubíquos, permitindo que informações e serviços sejam acessados quando e onde necessário. Dadas as suas características intrínsecas e a fim de atender aos seus requisitos, os sistemas ubíquos devem lidar com algumas limitações (ALENCAR et al., 2014; KUMAR, 2009; ROLIM et al., 2015):

- Instabilidade, perda de conexão ou área de sobra;
- Confiabilidade e disponibilidade dos recursos computacionais;
- Restrições de interação, como tela pequena ou falta de botões em dispositivos vestíveis;
- Limitações de hardware, como processamento, memória e espaço para armazenamento reduzidos;
- Autonomia de energia e bateria;
- Heterogeneidade de dispositivos e sistemas (fornecedores e plataformas diferentes);
- Segurança e privacidade em ambientes distribuídos;
- Padronização na comunicação entre dispositivos de fornecedores diferentes;
- Gerenciamento e coordenação de atividades entre dispositivos diversos

O desenvolvimento de aplicações ubíquas, atendendo às especificidades destas aplicações, requer plataformas e ferramentas específicas, assim como a infraestrutura em que essas aplicações são executadas. Empresas do setor de software e de hardware já fornecem ambientes robustos para que desenvolvedores criem suas soluções e para que tais soluções sejam executadas oferecendo aos usuários as suas funcionalidades. A plataforma Android, por exemplo, foi projetada de maneira a oferecer (a) recursos para desenvolvedores e (b) um ambiente para execução de aplicações ubíquas garantindo interoperabilidade, mobilidade, heterogeneidade, adaptabilidade, interação implícita, auto-gerenciamento e recursos de Realidade Aumentada (HASSAN, 2008).

Em particular, os *smartphones Android* fazem parte do cotidiano da maioria dos usuários de dispositivos móveis (NABILA POPAL; RYAN REITH, 2021). A interação com estes dispositivos é por meio de tela sensível ao toque e, apesar de possuírem telas majoritariamente pequenas, têm poder de processamento suficiente para processamento gráfico, isso possibilita inúmeras aplicações de entretenimento. Ainda, muitos deles possuem GPU e compatibilidade com a biblioteca gráfica *Vulcan*, que torna a interação do usuário mais fluida, visto as vantagens desses recursos para o aumento de performance.

O desenvolvimento para *Android* pode ser feito com diversas linguagens de programação, como *Python, Kotlin, Java, C#, C/C++. Java* e *JavaScript com React*

Native. As mais usuais são *Java* e *Kotlin*, utilizando ambiente de desenvolvimento *Android Studio*, que formam uma poderosa ferramenta. Aplicações ubíquas voltadas para diversos setores foram desenvolvidas na plataforma Android, tais como: saúde (GEMAN et al., 2018; HII; CHUNG, 2011), automação residencial (PIYARE, 2013), educação (SHANMUGAPRIYA, 2011), entre outros setores.

6.3. Realidade Aumentada

Quando um usuário é inserido em um ambiente virtual, diz-se que a tecnologia utilizada é Realidade Virtual (RV), quando o ambiente recebe objetos virtuais, diz-se que a tecnologia utilizada é Realidade Aumentada. Um exemplo de aplicação de RV é os óculos VR, ele permite que o usuário visualize um ambiente virtual e que se pareça que esteja dentro desse ambiente. Os filtros que inserem elementos nas fotos, figurinhas, máscaras, perucas, óculos, etc., presentes no aplicativo Instagram (INSTAGRAM, [s.d.]), é um exemplo de aplicação de RA.

Sabe-se que a tecnologia tem avançado na área digital e as que envolvem RA estarão cada vez mais presentes no cotidiano. No entretenimento, os jogos eletrônicos, filmes e aplicativos de redes sociais estão contanto com esta tecnologia ("Pokémon GO", 2016; RAUSCHNABEL; ROSSMANN; TOM DIECK, 2017). Também, RA tem potencial em outros contextos, como na educação e em aplicações em Cidades Inteligentes (MADI; ALBAKRY; IBRAHIM, 2020).

No entanto, o desenvolvimento de aplicações de RA tem suas barreiras computacionais, seja por limitações de hardware, seja pela complexidade de algumas tarefas, e a percepção do ambiente pelo dispositivo é uma delas.

Quando um usuário está interagindo com um ambiente, por meio de uma aplicação mobile, as posições dos objetos no ambiente mudam em relação ao dispositivo, de acordo com o movimento dado a ele pelo o usuário, por exemplo, quando o usuário gira o celular, balança ou aponta para algum lugar. Essa dinâmica deve ser considerada nas aplicações, podendo ser feita com a exploração dos recursos disponíveis nos dispositivos (GRUBERT; GRASSET, 2013).

Com auxílio dos sensores, é possível aumentar o dinamismo e realismo nas aplicações. A câmera pode fornecer imagens em tempo real; o GPS fornece a localização instantânea do dispositivo; o sensor magnético junto com o acelerômetro e giroscópio orientam o dispositivo no ambiente; enquanto a tela sensível ao toque recebe comandos do usuário e retorna as imagens da aplicação.

Com isso, uma forma de orientar o dispositivo no ambiente, fazendo com que a interação seja dinâmica, é a combinação dos sensores magnético, giroscópio e acelerômetro, como sugerido por Lawitzki (2011) em seu tutorial na Web. Visão computacional é outra forma, que se tornou possível com o avanço do hardware dos dispositivos móveis.

Por outro lado, outro requisito das aplicações de RA, no caso das aplicações mobile, são as alterações nas imagens obtidas pela câmera. Essas alterações podem ser desenhos de objetos 2D e 3D, por exemplo. Por sua vez, estes desenhos demandam processamento gráfico e podem ser feitos com bibliotecas gráficas, como *OpenGL ES* e *Vulkan*.

Estas ferramentas podem ser utilizadas nos *smartphones Android*. Ademais, a plataforma *Android* é a mais frequente e possui a maior variedade de *hardware*, mas tendo câmeras, GPS, tela sensível ao toque, sensor magnético, acelerômetro, giroscópio, acesso à internet e processamento de alta performance, como características em comum (KO; CHANG; JI, 2013). Apesar disso, muitas vezes, ainda não é possível utilizar orientação com sensores, visão computacional, e processamento gráfico ao mesmo tempo. Então, uma comparação das vantagens da utilização de um ou outro recurso pode ser feita, para que a aplicação funcione.

Existem ferramentas, de nível mais alto, para desenvolvimento de aplicações de RA na plataforma *Android,* considerando as limitações de hardware, como as "*Tool Kits*" para RA[1], de forma similar, têm-se o Vuforia SDK[2]. Estas podem ser utilizadas como soluções pré-modeladas para aplicações de RA. A vantagem é que boa parte das integrações com sensores, visão computacional e processamento gráfico já estão desenvolvidas e otimizadas, assim, o desenvolvedor da aplicação de RA em si não precisa se preocupar muito com aspectos da computação gráfica e otimização dos algoritmos. Isso permite um desenvolvimento mais rápido e suscetível a menos erros.

Portanto, vê-se que as aplicações de RA estão tendendo para dispositivos móveis, consequentemente, para a plataforma *Android,* por ser mais popular. Além disso, as limitações ainda existentes estão sendo minimizadas por meio de melhoramento de *hardware* e soluções de *software*. Com isso, aplicações de RA estão cada vez mais comuns.

6.3.1 Aplicações e exemplos

O Livro Mágico é um tipo de aplicação de RA, usando-se de um livro para inserir elementos estáticos, que podem variar com a movimentação das páginas. De forma análoga, projetores podem inserir imagens de objetos, estáticos ou animados, em uma superfície real, como mostram as figuras Figura 6.1 (a) e Figura 6.1 (b) (CRAIG, 2013).

[1] ARToolKit; ARToolKit for Android

[2] Vuforia SDK

(a) (b)

Figura 6.1. Exemplo de uma aplicação de RA a estilo de Livro Mágico. Fonte: (CRAIG, 2013)

Quando se usa um espelho para refletir uma imagem real em conjunto com elementos adicionais, tem-se o Espelho Mágico, a exemplo, uma pessoa pode se olhar no espelho, projetando uma peruca que foi desenhada no próprio espelho físico. Assim como o Livro Mágico, este estilo tem suas similaridades com aplicações mais atuais, como a projeção de uma barba, peruca ou qualquer um objeto, em uma imagem tirada com a câmera, utilizando o aplicativo Instagram, por exemplo. A Figura 6.2 mostra um exemplo desse estilo.

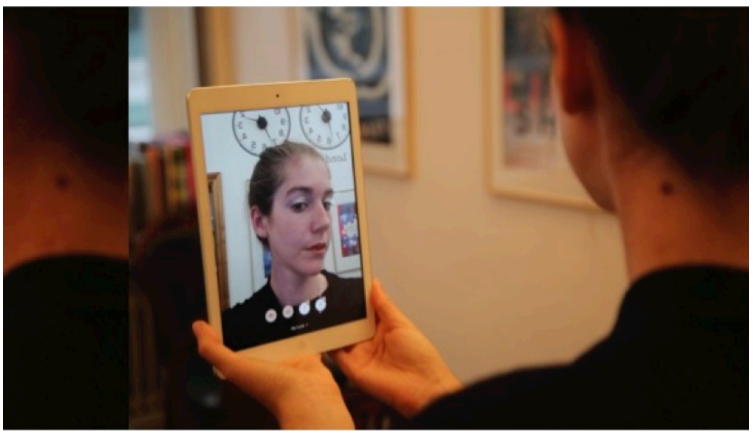


Figura 6.2. Aplicação em dispositivo móvel que insere maquiagem na imagem obtida pela câmera frontal. Fonte: (JAVORNIK et al., 2016)

Um dos estilos mais frequentes é o Janela e Portas Mágicas, no geral, apresentam-se imagens animadas em uma janela no ambiente e esta janela pode ser um cartaz com alguma imagem que servirá de espaço para conter alguma animação. O Lente Mágica é um estilo que abrange a maioria das aplicações de RA, pois envolve o uso de telas, que

podem ser de computadores e smartphones. A Figura 6.3 mostra um exemplo que combina o estilo Janelas e Portas Mágicas com o Lente Mágica, feito por KAN, TENG e CHOU (KAN; TENG; CHOU, 2009), utilizando QR code como marcador.

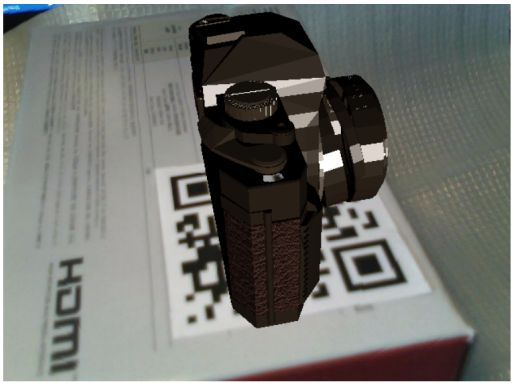


Figura 6.3. Tela de uma aplicação que utiliza uma imagem com QR Code como janela para inserir um objeto virtual na imagem, em tempo de execução. Fonte: (KAN; TENG; CHOU, 2009)

Entendendo estes estilos, pode-se estendê-los e combiná-los para aplicar RA nos diversos contextos. Como MADI et al, que constataram que um aplicativo de RA estimula a criatividade de crianças no aprendizado de Hajj, através de um estudo de caso de uma aplicação de RA nesse contexto. Esta aplicação foi em dispositivos móveis, para o auxílio no aprendizado de Hajj de crianças na Malásia, baseada no construtivismo.

IRWANSYAH et al. (IRWANSYAH et al., 2018) fizeram um aplicativo para estudo de estrutura molecular, na disciplina de Química, com uso de RA ao estilo Janelas e Portas Mágicas. Mostraram uma metodologia com uso de *Sketchup*, para modelar objetos 3D; *Corel Draw X5*, para criar o marcador; e, *Unity* para construir um jogo executável no sistema *Android*.

RUAN e JEONG (RUAN; JEONG, 2012) mostraram o uso de *QR code,* como marcador, para aplicações de RA em smartphones *Android*. Discutiram as vantagens comparado com o *ARToolKit*, onde colocou-se evidencia a possibilidade de guardar e ler informações, por meio de codificação e decodificação, com processamento mais rápido. A aplicação que eles mostraram foi feita com uso da câmera de *smartphone* para capturar imagens de *QR code* no ambiente. A parte de identificação do código, extração de informações e inserção de um objeto 3D na imagem, foram desenvolvidas com auxílio das bibliotecas *OpenCV* e *OpenGL*.

RODELLO e BREGA (RODELLO; BREGA, 2011) fizeram uma revisão sistemática sobre realidade aumentada no contexto de marketing. Entre os trabalhos discutidos, mostraram um aplicativo mobile de mapeamento. O funcionamento desta aplicação consiste em mostrar ícones com nomes dos locais, junto com as distâncias em tempo, como mostra a Figura 6.4. Pinto e Centeno (2012) também implementaram uma aplicação similar à da aplicação do Bradesco em dispositivos móveis.

Figura 6.4. Tela da aplicação de mapeamento do Bradesco implementada na plataforma mobile iOS. Fonte: (RODELLO; BREGA, 2011)

BADOUCH et *al*. (BADOUCH et al., 2018) fizeram uma revisão bibliográfica sobre trabalhos de RA, no contexto de cidades inteligentes, onde é discutido as limitações e dificuldades a serem vencidas, que aparecem nos trabalhos revisados. Entre eles, destaca-se um aplicativo feito com a ferramenta *ARKit*, para *smartphones iOS*, que insere um objeto 3D em tempo de execução em uma imagem obtida pela câmera. Um segundo exemplo de aplicação, é um sistema que recebe dados de diversos sensores distribuídos na cidade de Santander (Cantábria, Espanha) e envia esses dados para um dispositivo móvel em tempo real, para que o usuário seja informado sobre o tempo, pontos de ônibus, serviços de motoqueiros, etc. Enquanto o celular captura imagens da rua, essas informações são transmitidas na tela, inseridas nas imagens. O terceiro, é um óculos inteligente, *Google Glass*, criado pela *Google*, que usa câmera, microfone e acesso a internet para dispor objetos no ambiente de forma virtual, para o usuário. Uma das dificuldades destacadas é em relação ao poder de processamento do dispositivo para executar os aplicativos de RA, onde o público pode não ter dispositivos na faixa ideal de hardware. A privacidade e segurança também é um fator que pode ser problema para alguns protótipos, como o *Google Glass*.

KOUNAVIS et al. (2012) Examinaram o estado da arte na área de RA, para dispositivos móveis, avaliaram os objetivos, limitações tecnológicas de vários trabalhos e propuseram um modelo de desenvolvimento para aplicações nessa área, para o turismo. Eles citaram várias ferramentas com potencial para desenvolver aplicações com realidade aumentada, entra-se em destaque a *IN2AR*, que faz uso de características de imagens em geral, onde elas podem ser usadas como marcador ao invés de somente *QR code*, porém, não está disponível para smartphones *Android*. Entre as aplicações destacadas no contexto de turismo, tem-se o Tuscany+(iOS) que obtém as informações dos locais que são visitados na internet e desenha elas na tela, que é semelhante a ideia proposta do presente trabalho. Outro exemplo é o Basel AR Tourist Guide, que dispõe de informações disponíveis de museus, restaurantes, hotéis e shopping centers, da cidade de Basel na tela. Já o StreetMuseum, coloca imagens históricas das ruas na tela quando o usuário aponta a câmera para elas.

Cabe ressaltar ainda, que com o avanço da tecnologia de hardware, aplicações de RA com uso de *Deep Learning* tornaram-se possíveis, tanto para computadores de mesa, quanto em dispositivos móveis. WIBOWO, HARTANTO e WARTANTO (2020) desenvolveram uma aplicação para detecção de câncer de pele, na plataforma *Android*, que é uma aplicação de RA ao estilo Lente Mágica, como ilustrado na Figura 6.5, o resultado é um retângulo desenhado na tela, com um nome e probabilidade de acerto.

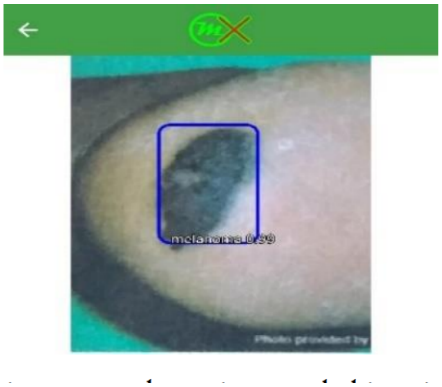


Figura 6.5. Aplicação Android que detecta câncer de pele, através de uma imagem. Fonte: (WIBOWO; HARTANTO; WIRAWAN, 2020).

Também, utilizando a técnica de *Deep Learning*, CHEN et al e JEONG (CHEN et al., 2021; JEONG, 2020), cada, fizeram aplicações que detectam um objeto na imagem real e desenham alguma informação. Dessa forma, pode-se estender o uso de marcadores nas imagens, para qualquer objeto real, como demonstrado pelos autores.

Então, percebe-se uma relação entre os estilos de RA, destacados por CRAIG (2013), com os trabalhos da literatura. Além disso, nota-se a diversidade de aplicações de RA, assim como o leque de ferramentas existentes.

6.4. Exemplo prático: Campus Inteligente

Uma consequência dos projetos de Cidades Inteligentes foi o surgimento dos Campus Inteligentes: regiões universitárias formadas por prédios e instalações voltadas para atividades acadêmicas de ensino, pesquisa e extensão voltadas para a graduação e para a pós-graduação e equipadas com uma variedade de serviços digitais ubíquos e a disponibilidade de informações contextuais para a comunidade de usuários deste espaço. A bem da verdade, conforme defende Ferreira e Araújo (2018), o Campus Inteligente pode ser comparado a Cidades Inteligentes menores, que buscam atender às demandas dos seus residentes oferecendo serviços de forma eficiente, reduzindo custos, mas, num ambiente controlado. Nesse tipo de projeto, não se pode perder de vista o contexto educacional em que está inserido e a necessidade de proporcionar experiências de qualidade no que diz respeito ao ensino-aprendizagem, mas, também em relação aos serviços que orbitam em torno das questões acadêmicas: biblioteca, restaurante, mobilidade etc.

No contexto de Campus Inteligentes, várias universidades têm oferecidos uma gama de aplicações e funcionalidades - geralmente sob a forma de aplicativos para smartphones com diversas finalidades (FERREIRA; ARAÚJO; SANTOS, 2018; TAROUCO et al., 2017): conscientização ambiental, controle de tráfego, ensino, saúde, segurança, infraestrutura, por exemplo.

Pensando no contexto prático de turismo e mobilidade, verificou-se que existem poucas soluções que permitem, ao usuário, o aumento da percepção de um ambiente externo desconhecido. No entanto, de forma similar aos trabalhos citados, pode-se aplicar RA para tal objetivo.

Nesse contexto, limitando-se ao caso do campus de Cruz das Almas - Bahia, da UFRB, pensou-se numa aplicação Android que mostra informações a respeito dos prédios. Tais informações são horário de abertura e fechamento, e o tipo de serviços que o oferecidos nos prédios.

Esta aplicação deve ser na plataforma Android, para abranger um maior número de usuários e, ainda, proporcionar boa experiência aos usuários. Para isso, necessita-se de uma interface limpa e um algoritmo de média ou baixa demanda de processamento, para evitar travamentos durante a execução.

Com relação à modelagem do sistema, considere a Figura 6.6, que mostra a disposição dos recursos utilizados. A ideia é fazer uma aplicação Android que obtém imagens da câmera e informações de acelerômetro, sensor geomagnético e GPS, em tempo de execução, daí, exibir uma imagem do ambiente real com um objeto virtual, que indiquem informações a respeito do local de onde o usuário está.

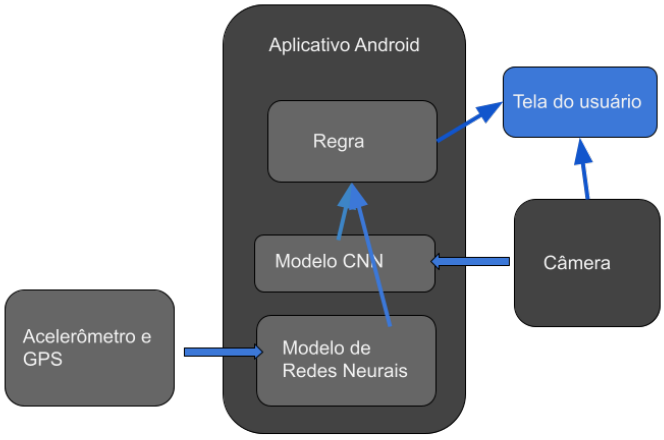


Figura 6.6. Disposição dos componentes no sistema de RA, na plataforma Android. Fonte: Autoria própria.

Naa Figura 6.6, o componente “Modelo CNN” é um modelo de Redes Neurais Convolucionais para detecção de objetos, com a arquitetura Mobile Net V2 (YOUNIS et al., 2020). Esta arquitetura permite obter as coordenadas da posição de um objeto em uma imagem, com estas coordenadas, é possível desenhar um objeto virtual, ao estilo Janela e Portas Mágicas, por exemplo.

Para treinar um modelo de detecção de objetos, deve-se construir uma base de dados para ajuste e testes. No presente trabalho, utilizou-se 320 imagens, distribuídas entre 9 tipos diferentes de blocos do campus de Cruz das Almas, da UFRB, representadas pela Figura 6.7. Cada imagem foi anotada com o programa LabelImg (TZUTALIN, 2015), com retângulos que contenham os blocos.

Figura 6.7. Parcela do conjunto de imagens montado para treinamento e teste do modelo de detecção de objetos. Fonte: Google Maps (GOOGLE MAPS, 2021)

Um passo a passo, em forma de tutorial, para treinamento de um modelo de detecção de objetos utilizando a arquitetura Mobile Net v2 pode ser encontrado no repositório do *TensorFlow*.

O componente “Modelo de Redes Neurais” é um regressor, que tem os valores instantâneos do magnetômetro e acelerômetro como entrada, e retorna o menor ângulo que à tela do celular faz com o vetor sul->norte geométrico (N-S). Este ângulo serve para confirmar o bloco identificado pelo modelo de detecção de objetos, através de uma regra. Este modelo de regressão é necessário, pois os valores lidos pelo magnetômetro também variam de acordo com a inclinação do eixo y do celular em relação ao normal do globo terrestre, mesmo mantendo a direção da normal à tela ($\widehat{n_c}$) constante, em relação ao vetor N-S , como mostrado na Figura 6.8. Dessa forma, outros sensores devem auxiliar na

definição da direção de $\widehat{n_C}$ em relação às coordenadas geométricas, como o sensor acelerômetro, que retorna as intensidades da gravidade nos três eixos do celular.

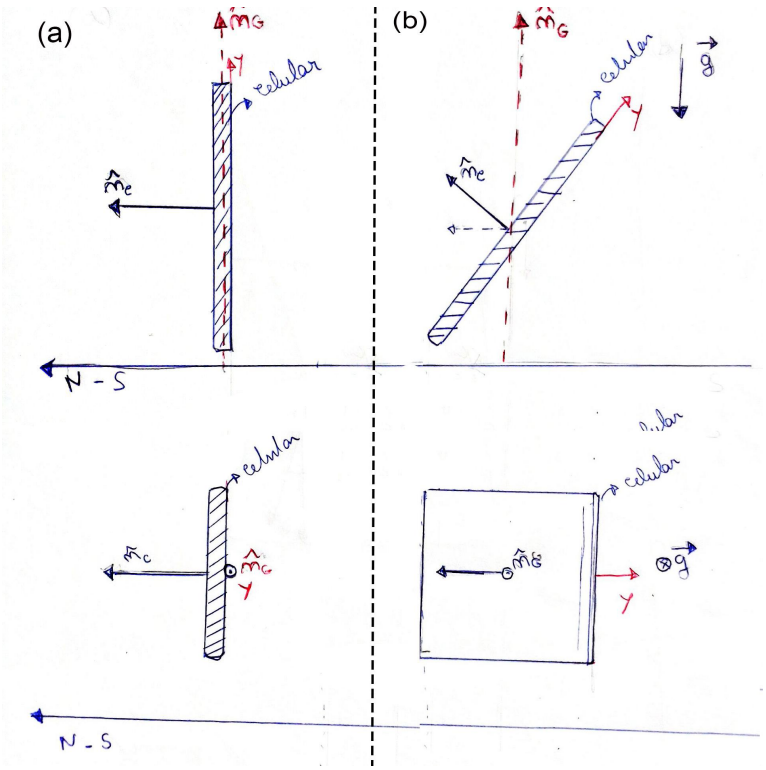


Figura 6.8. Diagrama de vetores estudados no smartphone, com relação ao ambiente e as leituras dos sensores. $\hat{n}_G$é o vetor unitário normal à superfície terrestre; $\widehat{n_C}$ é o vetor unitário normal à superfície paralela à tela do celular. g é o vetor aceleração gravitacional. (a) Celular posicionado de forma vertical. (b) Celular inclinado.

O sensor magnético também retorna intensidades do campo magnético terrestre para os três eixos do dispositivo. Então, quer-se obter um conjunto de parâmetros que definem a relação entre essas 6 variáveis dos sensores, para definir uma angulação de $\widehat{n_C}$ com o vetor *N-S*, no presente problema, um modelo de redes neurais com duas camadas totalmente conectadas é suficiente. Este modelo é demonstrado pela Figura 6.9. O componente "Regra" tem o papel de receber os resultados da detecção dos blocos nas imagens obtidas pela câmera do celular, confirmar, por meio do resultado do modelo de regressão e decidir se desenha ou não na tela. Este desenho é feito com a biblioteca Canvas, do *Android*.

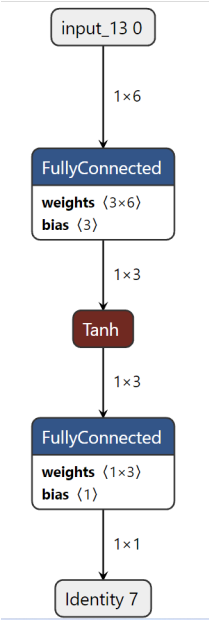


Figura 6.9. Arquitetura do modelo de regressão. Fonte: autoria própria.

A Figura 6.10 ilustra como a regra principal funciona. Este processo se repete continuamente, enquanto o usuário estiver na tela de execução.

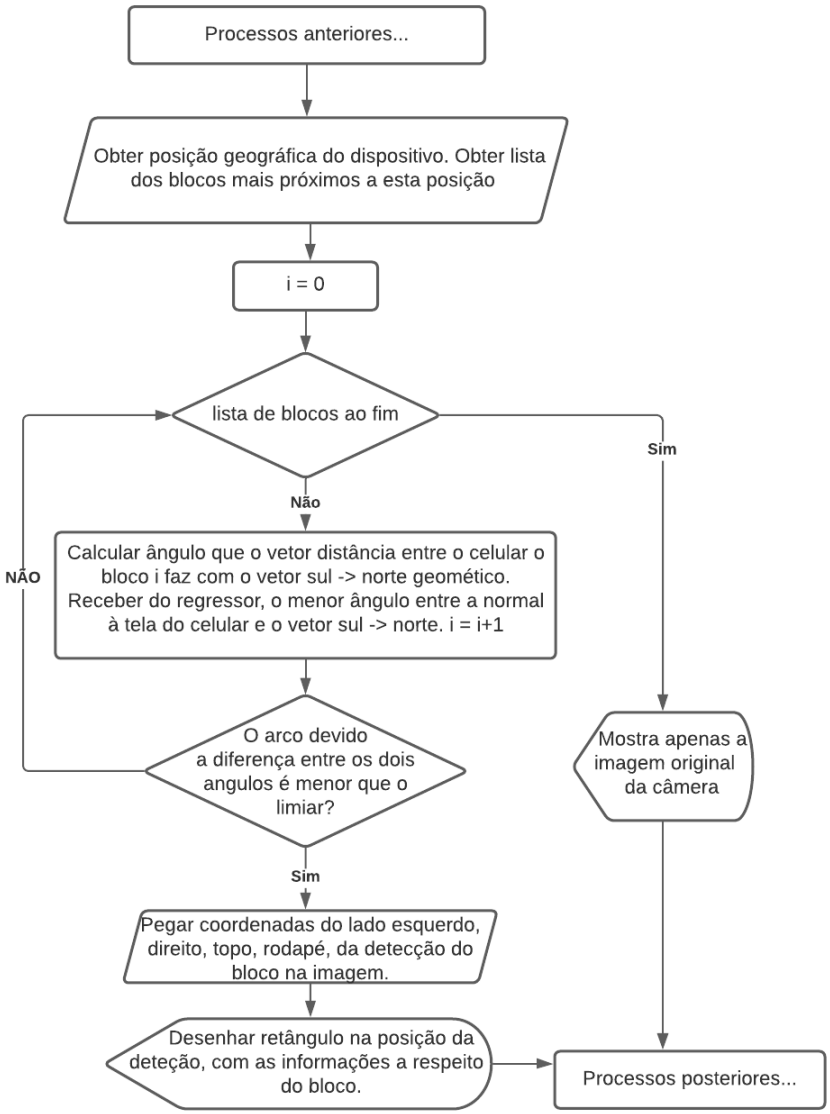


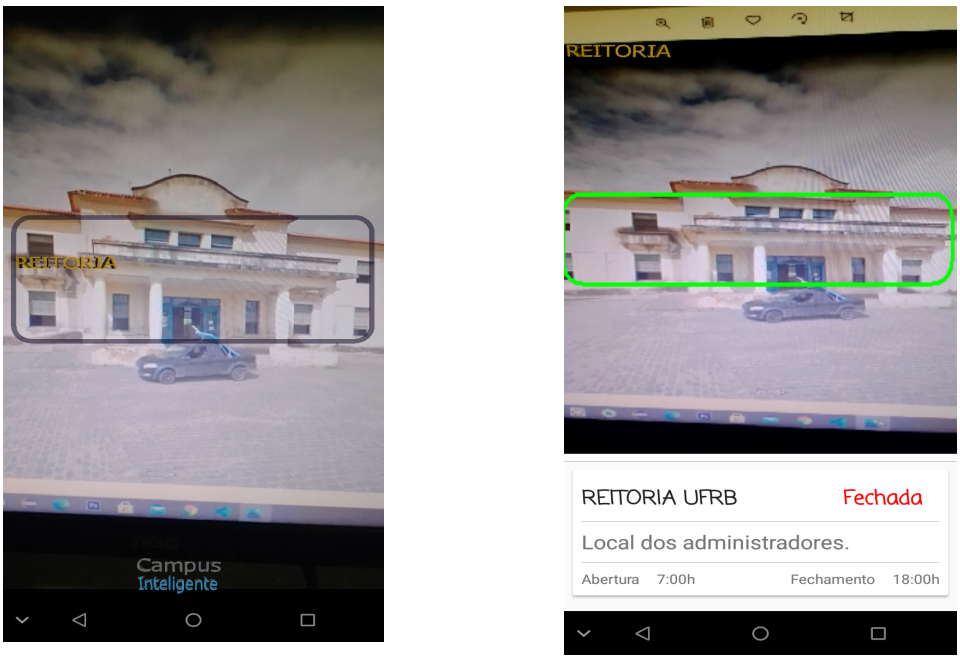
Figura 6.10. Fluxograma principal do aplicativo Android. Fonte: Autoria própria.

Após modelagem do sistema, segue-se para a implementação. Para isso, utiliza-se o ambiente de desenvolvimento *Android Studio*, com as linguagens *Java* e *Kotlin* como principais. Com relação desenvolvimento e treinamento dos modelos de *Machine Learning*, para detecção e regressão, foi utilizada a linguagem de programação Python.

A interface principal da aplicação implementada é mostrada pela Figura 6.11. A implementação teve como base a aplicação de exemplo do TensorFlow Lite, para detecção de objetos, disponível no repositório (TENSORFLOW COMMUNITY, [s.d.]).

Figura 6.11. Frame de vídeo de captura de tela durante teste em campo da aplicação, com regressor desativado. Fonte: autoria própria.

As figuras Figura 6.12 (a) e Figura 6.12 (b) mostram os resultados da aplicação *Android* implementada.



(a) (b)

Figura 6.12. (a) Tela principal do aplicativo ao detectar um prédio. (b) Tela de "mais informações". Fonte: autoria própria.

6.5. Considerações finais

Baseada na informação, comunicação e no conhecimento, a sociedade contemporânea é caracterizada pela conexão permanente às redes de comunicação. A grande maioria das atividades cotidianas realizadas pelos seres humanos nos dias de hoje são executadas online. Isso se dá graças a uma infraestrutura de hardware, software e telecomunicações conhecida como Computação Ubíqua. Conectados a smartphones, laptops, wearables podemos acessar a serviços e aplicações, como Realidade Aumentada, a qualquer hora em qualquer lugar. A Computação Ubíqua proporciona tecnologias baseadas em dispositivos, conexão a redes e software que permitem aos usuários, mesmo em movimento e longe de cabos de conexão, acessarem aplicações e serviços armazenados remotamente.

Por outro lado, a Realidade Aumentada possibilita ampliar os mecanismos de visualização da informação, bem como enriquecer o conteúdo apresentado ao usuário com o acréscimo de mais dados que a princípio não estão visíveis no ambiente real. Setores como educação, entretenimento e saúde têm disponibilizado soluções inovadoras baseadas em RA tornando os processos de ensino-aprendizagem, formação, diversão e compreensão do funcionamento do próprio corpo mais fáceis, agradáveis e divertidos.

Podemos acessar aplicações de Realidade Aumentada nos deslocando e recebendo informações na tela sobre prédios, ruas, avenidas e praças. A associação de RA e Computação Ubíqua tem mostrado grande potencial sobretudo no desenvolvimento de aplicações baseadas na localização do usuário no espaço urbano e em soluções para Cidades Inteligentes. Jogos como Pokemon Go, soluções para turismo e instrumentos para visualização da informação são alguns exemplos de aplicação de RA no contexto de Computação Ubíqua.

Neste capítulo foi mostrado os conceitos de Realidade Aumentada e Computação Ubíqua. Com isso, fez-se uma revisão de literatura, avaliação de ferramentas disponíveis e possíveis aplicações na plataforma Android, assim como seu potencial para a área. Mostrou-se também, o processo de planejamento, modelagem e implementação do aplicativo Campus Inteligente, utilizando ferramentas recentes, como *Deep Learning e* a linguagem de programação Kotlin.

Como próximo passos no percurso do aprendizado convergindo os assuntos Computação Ubíqua e Realidade Aumentada sugere-se incluir estudo de Internet das Coisas e sensores.

Referências

ALBINO, V.; BERARDI, U.; DANGELICO, R. M. Smart Cities: Definitions, Dimensions, Performance, and Initiatives. **Journal of Urban Technology**, v. 22, n. 1, p. 3–21, 2 jan. 2015.

ALENCAR, T. S. DE et al. **Addressing the Users' Diversity in Ubiquitous Environments through a Low Cost Architecture**. SpringerLink. **Anais**... In: INTERNATIONAL CONFERENCE ON UNIVERSAL ACCESS IN HUMAN-COMPUTER INTERACTION. Springer, Cham, 22 jun. 2014. Disponível em:

<https://link-springer-com.ez67.periodicos.capes.gov.br/chapter/10.1007/978-3-319-07446-7_43>. Acesso em: 22 abr. 2017

BADOUCH, A. et al. **Augmented Reality services implemented within Smart Cities, based on an Internet of Things Infrastructure, Concepts and Challenges: an overview**. Proceedings of the Fourth International Conference on Engineering & MIS 2018. **Anais**...2018.

CHEN, J.-W. et al. A smartphone-based application for scale pest detection using multiple-object detection methods. **Electronics**, v. 10, n. 4, p. 372, 2021.

CRAIG, A. B. **Understanding augmented reality: Concepts and applications**. [s.l.] Newnes, 2013.

DA SILVA, D.; DA COSTA, C.; RIGHI, R. **Um Modelo de Realidade Aumentada no âmbito do Turismo Ubíquo**. Anais do VI Simpósio Brasileiro de Computação Ubíqua e Pervasiva. **Anais**...SBC, 2014.

DUBOIS, E.; NIGAY, L. **Augmented Reality: Which Augmentation for Which Reality?** Proceedings of DARE 2000 on Designing Augmented Reality Environments. **Anais**...: DARE '00.New York, NY, USA: ACM, 2000. Disponível em: <http://doi.acm.org/10.1145/354666.354695>. Acesso em: 16 dez. 2014

FERREIRA, F. H.; ARAÚJO, R. M.; SANTOS, R. P. DOS. **Perspectivas para a Implantação de Câmpus Inteligentes**. Anais da Escola Regional de Sistemas de Informação do Rio de Janeiro (ERSI-RJ). **Anais**... In: ANAIS DA V ESCOLA REGIONAL DE SISTEMAS DE INFORMAÇÃO DO RIO DE JANEIRO. SBC, 16 out. 2018. Disponível em: <https://sol.sbc.org.br/index.php/ersi-rj/article/view/4652>. Acesso em: 20 out. 2021

FERREIRA, F. H. C.; ARAÚJO, R. M. Campus Inteligentes: Conceitos, aplicações, tecnologias e desafios. **RelaTe-DIA**, 29 jan. 2018.

GEMAN, O. et al. **Ubiquitous Healthcare System Based on the Sensors Network and Android Internet of Things Gateway**. 2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). **Anais**... In: 2018 IEEE SMARTWORLD, UBIQUITOUS INTELLIGENCE COMPUTING, ADVANCED TRUSTED COMPUTING, SCALABLE COMPUTING COMMUNICATIONS, CLOUD BIG DATA COMPUTING, INTERNET OF PEOPLE AND SMART CITY INNOVATION (SMARTWORLD/SCALCOM/UIC/ATC/CBDCOM/IOP/SCI). out. 2018.

GOOGLE MAPS. **UFRB - Google Maps**. Disponível em: <https://goo.gl/maps/hF7qnQ1XCbtaMCh28>. Acesso em: 18 out. 2021.

GRUBERT, J.; GRASSET, R. **Augmented reality for Android application development**. [s.l.] Packt Publishing Ltd, 2013.

HASSAN, Z. S. **Ubiquitous computing and android**. 2008 Third International

Conference on Digital Information Management. **Anais**... In: 2008 THIRD INTERNATIONAL CONFERENCE ON DIGITAL INFORMATION MANAGEMENT. nov. 2008.

HII, P.-C.; CHUNG, W.-Y. A Comprehensive Ubiquitous Healthcare Solution on an Android™ Mobile Device. **Sensors**, v. 11, n. 7, p. 6799–6815, jul. 2011.

INSTAGRAM. **Instagram**. Disponível em: <https://instagram.com/>. Acesso em: 11 out. 2021.

IRWANSYAH, F. S. et al. **Augmented reality (AR) technology on the android operating system in chemistry learning**. IOP conference series: Materials science and engineering. **Anais**...IOP Publishing, 2018.

JAVORNIK, A. et al. **Revealing the shopper experience of using a" magic mirror" augmented reality make-up application**. Conference on designing interactive systems. **Anais**...Association for Computing Machinery (ACM), 2016.

JEONG, D. **Road Damage Detection Using YOLO with Smartphone Images**. 2020 IEEE International Conference on Big Data (Big Data). **Anais**...IEEE, 2020.

JIANG, Q. et al. **Citizen Sensing for Improved Urban Environmental Monitoring**. Research article. Disponível em: <https://www.hindawi.com/journals/js/2016/5656245/>. Acesso em: 12 dez. 2017.

KAJI, S. et al. Augmented reality in smart cities: applications and limitations. **Journal of Engineering Technology**, v. 6, n. 1, p. 18, 2018.

KAN, T.-W.; TENG, C.-H.; CHOU, W.-S. **Applying QR code in augmented reality applications**. Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry. **Anais**...2009.

KAVAKLI, M. A people-centric framework for mobile augmented reality systems (MARS) design: ArcHIVE 4Any. **Human-centric Computing and Information Sciences**, v. 5, n. 1, p. 37, 1 dez. 2015.

KO, S. M.; CHANG, W. S.; JI, Y. G. Usability principles for augmented reality applications in a smartphone environment. **International Journal of Human-Computer Interaction**, v. 29, n. 8, p. 501–515, 2013.

KUMAR, S. **Challenges for Ubiquitous Computing**. 2009 Fifth International Conference on Networking and Services. **Anais**... In: 2009 FIFTH INTERNATIONAL CONFERENCE ON NETWORKING AND SERVICES. abr. 2009.

LAWITZKI, P. **Paul Lawitzki | software developer and game designer**. Disponível em: <http://plaw.info/articles/sensorfusion/#articles>. Acesso em: 20 out. 2021.

MADI, N. M.; ALBAKRY, N.; IBRAHIM, N. AR Mobile Application in Learning Hajj for Children in Malaysia: A Preliminary Study. 2020.

NABILA POPAL; RYAN REITH. **Smartphone Market ShareSmartphone Market Share**, 29 jul. 2021. Disponível em: <https://www.idc.com/promo/smartphone-market-

share/os>. Acesso em: 23 set. 2020

PINTO, F. S.; CENTENO, J. A. S. A realidade aumentada em smartphones na exploração de informações estatísicas e cartográficas. **Boletim de Ciências Geodésicas**, v. 18, p. 282–301, 2012.

PIYARE, R. Internet of Things, Smart Home, Home Automation, Android Smartphone, Arduino. **Internet of Things**, p. 7, 2013.

Pokémon GO. Disponível em: <https://pokemongolive.com/>. Acesso em: 11 out. 2021.

POSLAD, S. **Ubiquitous Computing: Smart Devices, Environments and Interactions**. 1ª edição ed. Chichester, U.K: John Wiley & Sons Inc, 2009.

RASHID, Z. et al. Using Augmented Reality and Internet of Things to improve accessibility of people with motor disabilities in the context of Smart Cities. **Future Generation Computer Systems**, v. 76, p. 248–261, 1 nov. 2017.

RAUSCHNABEL, P. A.; ROSSMANN, A.; TOM DIECK, M. C. An adoption framework for mobile augmented reality games: The case of Pokémon Go. **Computers in Human Behavior**, v. 76, p. 276–286, 2017.

REID, S. E. et al. **Pervasive Mobile Services for Active Aging: An Exploratory Investigation into the Relationship Between Willingness-to-Adopt Mobile Devices and Shopping Experience**. SpringerLink. **Anais**... In: INTERNATIONAL CONFERENCE ON SMART HOMES AND HEALTH TELEMATICS. Springer, Cham, 25 maio 2016. Disponível em: <https://link-springer-com.ez67.periodicos.capes.gov.br/chapter/10.1007/978-3-319-39601-9_19>. Acesso em: 22 abr. 2017

RODELLO, I. A.; BREGA, J. R. F. Realidade virtual e aumentada em ações de marketing. **Realidade Virtual e Aumentada: Aplicações e Tendências**, v. 1, p. 44–57, 2011.

ROLIM, C. O. et al. An Ubiquitous Service-Oriented Architecture for Urban Sensing. **SpringerLink**, p. 1–10, 2015.

RUAN, K.; JEONG, H. **An augmented reality system using Qr code as marker in android smartphone**. 2012 Spring Congress on Engineering and Technology. **Anais**...IEEE, 2012.

SHANMUGAPRIYA, M. **Designing an M-Learning Application for a Ubiquitous Learning Environment in the Android Based Mobile Devices Using Web Services**, 2011.

TAROUCO, L. M. R. et al. Internet das Coisas na Educação trajetória para um campus inteligente. **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**, v. 6, n. 1, p. 1220, 27 out. 2017.

TENSORFLOW COMMUNITY. **TensorFlow Lite Object Detection Android Demo**. Disponível em: <https://github.com/tensorflow/examples>. Acesso em: 16 out. 2021.

TZUTALIN. **LabelImg**. [s.l: s.n.].

WEISER, M. The Computer for the 21st Century. **SIGMOBILE Mob. Comput. Commun. Rev.**, v. 3, n. 3, p. 3–11, jul. 1999.

WIBOWO, A.; HARTANTO, C. A.; WIRAWAN, P. W. Android skin cancer detection and classification based on MobileNet v2 model. **International Journal of Advances in Intelligent Informatics**, v. 6, n. 2, p. 135–148, 2020.

YOUNIS, A. et al. **Real-time object detection using pre-trained deep learning models MobileNet-SSD**. Proceedings of 2020 the 6th International Conference on Computing and Data Engineering. **Anais**...2020.

ZHOU, F.; DUH, H. B.-L.; BILLINGHURST, M. **Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR**. 2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality. **Anais**... In: 2008 7TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY. set. 2008.