UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Departamento de Sistemas de Computação

Interface de controle de navegação de robô para presença remota em museu

Murilo Luz Stucki

São Carlos - SP

Interface de controle de navegação de robô para presença remota em museu

Murilo Luz Stucki

Orientador: Eduardo do Valle Simões

Monografia referente ao projeto de conclusão de curso dentro do escopo da disciplina SSC0670 – Projeto de Formatura I do Departamento de Sistemas de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP para obtenção do título de Engenheiro de Computação.

Área de Concentração: Engenharia de Software para Controle de Robôs Móveis

USP – São Carlos
23 de Novembro de 2021

“Não é nossa função controlar todas as marés do mundo, mas sim fazer o que pudermos para socorrer os tempos em que estamos inseridos” - Gandalf, O Branco.

(J. R. R. Tolkien)

Agradecimentos

Agradeço à minha família, colegas, irmãos e todos aqueles que acreditaram no meu potencial e me motivaram a chegar até aqui, à todos que ofereceram-me suporte nos momentos mais difíceis e de dúvida da minha vida.

RESUMO

Diante de um cenário de pandemia e escassez de recursos, o maior desafio na administração de museus e espaços culturais é a manutenção do seu funcionamento perante medidas de distanciamento social. Este trabalho explora uma das soluções que surgiu dentro do Instituto de Ciências Matemáticas e de Computação (ICMC) para ressignificar o papel do seu Museu de Computação durante a pandemia, a solução, que levou o nome de Robô Museu, se trata da implementação de um sistema robótico para a realização de visitas remotas ao museu. Neste trabalho é proposta e implementada uma arquitetura para o sistema de interface web para o projeto Robô Museu, utilizando-se as tecnologias mais recomendadas para a área de telepresença robótica. Foi desenvolvida uma aplicação Node.js utilizando a tecnologia WebRTC (*Web Real Time Communication*), a aplicação serve duas páginas HTML diferentes, uma para os usuários visitantes, e outra para um computador que servirá de interface entre a Internet e o microcontrolador do robô. A aplicação foi testada utilizando um servidor NGINX no endereço 'https://principia.icmc.usp.br/robo-museu/'. Os testes realizados mostram que a aplicação é capaz de conectar em tempo real os usuários com a interface e transmitir um fluxo de vídeo/áudio da interface para múltiplos usuários. A aplicação também é capaz de transmitir um fluxo de comandos de um usuário principal para a interface, que por sua vez transmite estes comandos para o endereço IP do microcontrolador do robô conectado à rede Wi-Fi.

SUMÁRIO

CAPÍTULO 1: INTRODUÇÃO

1.1. Contextualização e Motivação

A pandemia da Covid-19 impactou negativamente toda a sociedade mundial, o setor cultural não foi exceção, a crise afetou a maioria das pessoas e das organizações que trabalham neste setor, como aponta uma pesquisa da UNESCO (UNESCO, 2020). Entre os problemas causados estava a impossibilidade de realizar visitas a museus, galerias, teatros e outros espaços importantes para a sociedade por mais de um ano, e durante este período muitas ideias surgiram para ajudar a manutenção destes espaços, inclusive dentro do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP) surgiram algumas ideias para serem implementadas no Museu de Computação[1], uma delas foi a de construir um robô que possa ser controlado remotamente e filme o ambiente a sua volta, para que usuários assistam essas imagens em tempo real e sintam-se livres para explorar o ambiente.

Rapidamente foi reconhecido o potencial do projeto do robô de acesso remoto, a ideia transformou-se em utilizar o robô para que escolas públicas e outras instituições de ensino pudessem fazer visitas remotamente em espaços de interesse da escola, principalmente espaços em que a visita presencial é inviável. Portanto o objetivo do projeto intitulado Robô Museu[2] tornou-se desenvolver um robô móvel junto a uma plataforma de controle remoto, que possa ser facilmente replicada e implementada por outras instituições, de forma a facilitar o acesso remoto do público aos espaços de cultura, além de influenciar ativamente que alunos de escola pública se interessem mais por atividades culturais e ensinar sobre tecnologia e robótica. Este trabalho terá como foco a plataforma web que será usada para controlar o robô, contribuindo com um protótipo para testes iniciais do robô, e até como base para futuras implementações.

Para que uma experiência de telepresença seja eficaz ela precisa transmitir a sensação do usuário de estar em um ambiente diferente (MINSKY, M., 1980), a vantagem de utilizar robôs móveis para esta tarefa, além das câmeras e microfones, é dar ao usuário remoto a possibilidade de se mover e interagir fisicamente com o ambiente por meio do robô (KRISTOFFERSSON et al., 2013). Atualmente, a área de telepresença robótica cresce

[1] "Museu de Computação Prof. Odelar Leite Linhares", https://mc.icmc.usp.br/.
[2] Repositório Robô Museu por Eduardo do Valle Simões, https://gitlab.com/simoesusp/robo-museu.

rapidamente, e existem muitos exemplos de robôs para usos comerciais e também não comerciais. Um exemplo é o uso de telepresença robótica para auxiliar pessoas com deficiências locomotoras (MARAFA, N. A.; FILHO, W. B. V., 2019), e como grande parte dos trabalhos atuais na área ele utiliza a API (Application Programming Interface) para comunicação em tempo real entre navegadores chamada WebRTC[3] (Web Real Time Communication), Singh et al. (2013) produziram um artigo de análise da performance do WebRTC e seus algoritmos de controle de congestão, e nos testes obtiveram latências de no máximo 1 segundo para conexões congestionadas. Este trabalho faz uso de tecnologias atuais já utilizadas na área para a criação de uma interface web de telepresença robótica (MELENDEZ-FERNANDEZ et al, 2017), a diferença está em dois pontos, apresentados a seguir, que derivam do caso específico em que se encontra o projeto do Robô Museu, e os tipos de uso intencionados pelos seus idealizadores.

1. Criar uma aplicação na qual seja possível mais de um usuário remoto assistir a transmissão do robô ao mesmo tempo, mas garantir que apenas um usuário (usuário principal) possa mandar comandos para o robô. A definição do usuário principal poderá ser feita por meio de um sistema de cadastro e agendamento de horário.
2. Reduzir ao máximo a interação humana necessária localmente após a instalação, no robô e no computador servindo de transmissor. Permitindo que após o sistema estar devidamente instalado, um usuário remoto consiga estabelecer uma conexão com o robô sem o auxílio de uma pessoa no local.

O primeiro ponto tem o intuito de aumentar o potencial uso educacional do sistema, possibilitando por exemplo que uma classe de alunos em aulas remotas possam assistir a transmissão juntos enquanto o professor controla o robô, ou também que usuários não principais possam assistir a visita de um usuário principal em horários muito requisitados, otimizando assim o uso do robô e melhorando a experiência do usuário. O segundo ponto visa facilitar a manutenção do sistema pela instituição, reduzindo a necessidade de treinar e alocar funcionários para operar o robô. E para o usuário visa diminuir o tempo necessário para estabelecer a conexão com o robô. Tendo em vista os pontos apresentados, este trabalho poderá contribuir com uma forma de implementação de telepresença robótica que melhor atende a demanda de um espaço cultural, como um museu, estudar as técnicas e os desafios e fomentar o debate acerca da área de telepresença robótica.

[3] "WebRTC." https://webrtc.org/. Acessado em 23 nov.. 2021.

1.2. Objetivos

O objetivo deste trabalho é desenvolver uma aplicação web de código aberto, utilizando tecnologias Open Source[4] e com performance adequada às necessidades do projeto Robô Museu, como Node.js[5], NGINX[6], WebRTC e HTML[7]. Esta aplicação deve ser capaz de servir de interface entre um usuário inexperiente e um robô móvel. O intuito principal é o usuário conseguir, de forma confortável e sem possuir conhecimento básico sobre robótica ou computação, interagir com o robô e fazê-lo navegar por um espaço, enquanto o robô captura imagens do ambiente e as transmite para o usuário em tempo real. A comunicação deve ser feita de forma simples e com a menor latência possível entre o envio dos comandos e a alteração da imagem na tela do usuário, isso para garantir uma melhor imersão para a pessoa que está visitando o espaço por telepresença. Uma condição de sucesso baseado na latência seria obter um número menor que 2 segundos, se aproximando dos resultados do trabalho de Singh et al. (2013).

1.3. Organização do Trabalho

A organização deste trabalho consiste de 4 capítulos, sendo este o primeiro. O segundo capítulo, REVISÃO BIBLIOGRÁFICA, apresentará revisões da terminologia básica da área, explicações sobre as bibliotecas e projetos de código aberto relevantes e a literatura relacionada a este projeto. No terceiro capítulo, DESENVOLVIMENTO DO TRABALHO, o projeto será discutido em detalhes, serão apresentadas motivações de todas as escolhas feitas, assim como a execução e os resultados do trabalho. Por fim, no quarto capítulo, CONCLUSÃO, serão apresentadas as contribuições, os erros cometidos e outras conclusões finais que o autor deste trabalho pôde alcançar a partir dos resultados obtidos. Todas as etapas deste trabalho foram realizadas à distância pelo autor e seu orientador, o Prof. Eduardo do Valle Simões, e sem a possibilidade de testar o sistema em situações reais, os resultados se limitam aos obtidos em simulações. Os testes restantes, como o de integração com os outros componentes do Robô Museu, poderão ser feitos futuramente quando todos os componentes estiverem em etapas finais de desenvolvimento. Assim serão sugeridos os próximos passos e trabalhos futuros que poderão ser derivados deste e também, ao final, feitas algumas

[4] "The Open Source Definition", https://opensource.org/docs/osd.
[5] "Node.js.", https://nodejs.org/en/about/.
[6] "NGINX.", https://nginx.org/en/.
[7] "HTML Standard", https://html.spec.whatwg.org/multipage/.

considerações sobre o curso de graduação no qual o autor está atualmente matriculado a partir de sua visão pessoal, e sobre a relação deste trabalho com o curso e o que foi aprendido durante o desenvolvimento do projeto.

CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA

2.1. Considerações iniciais

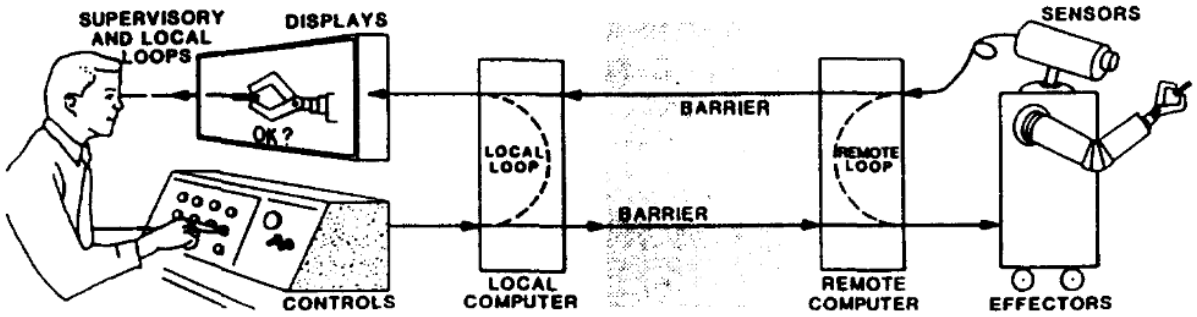
Neste capítulo serão apresentadas revisões da literatura existente relacionada a área de telepresença robótica, conceitos e técnicas avaliados durante o desenvolvimento do trabalho, terminologias importantes para a compreensão deste projeto e a metodologia adotada para a implementação do sistema. São apresentadas explicações das funcionalidades das bibliotecas de código aberto mais importantes e também sobre as ferramentas Open Source empregadas na solução apresentada neste trabalho.

2.2. Telepresença e Teleoperação Robótica

O termo teleoperação, traduzido do inglês *teleoperation*, é o nome dado a qualquer tipo de operação realizada em uma máquina por um controlador remoto, pode ser tão simples quanto mudar o canal em uma televisão com um controle remoto, ou mais complexa como controlar um robô na superfície de Marte a partir de uma estação espacial na órbita do planeta (SCHILLING et al., 1997). Atenta-se para o fato de que o termo teleoperação não necessariamente tem ligação com a robótica, apenas quando a máquina controlada se trata de um sistema robótico, por este motivo é comum encontrar definições deste termo como a área que engloba outros tópicos como a telepresença e a telerobótica, este último foi definido por T. B. Sheridan (1989) como tratando-se da teleoperação usada em sistemas robóticos semi-autônomos, dotados de sensores e inteligência artificial para realizar tarefas informadas por um supervisor remoto. A partir da análise da Figura 1 nota-se a semelhança entre o conceito de telerobótica, apresentado por T. B. Sheridan, e a descrição do uso do projeto Robô Museu, sendo a Internet a barreira entre o controlador e o robô. Não coincidentemente este

trabalho está inserido na área da telerobótica, e faz uso tanto de conceitos derivados da telepresença.

Figura 1: Conceito de Telerobótica



Fonte: T. B. SHERIDAN, Telerobotics (1989)

O termo telepresença, traduzido do inglês *telepresence*, não é tão bem definido quanto os apresentados anteriormente e pode ser encontrado em diversas áreas acadêmicas com significados variados. Na área de foco deste trabalho, a robótica, é usado com o significado de transmitir a sensação de estar em outro ambiente com o auxílio de robôs, e foi popularizado por Marvin Minsky em 1980, e em seu artigo ele descreve a telepresença como uma evolução da teleoperação, de forma que ao passo em que as tecnologias de controle remoto avançam, mais imersivos seus sistemas se tornam. Segundo Minsky (1980), esse deve ser o objetivo da área de teleoperações para que possam resolver problemas atuais de tarefas que põem em risco o trabalhador que poderão ser feitas por robôs usando a telepresença, e até trabalhos comuns poderiam ser feitos remotamente, eliminando a necessidade de locomoção do trabalhador e que criaria um mercado mais livre. A tecnologia e o mundo evoluíram diferente do que Minsky previu a 40 anos atrás, mas parte do que foi dito por ele é válido também nos dias de hoje, e podemos dizer que o modelo de economia remota está começando a tomar forma e mudanças na área estão acontecendo rapidamente. A discussão sobre os impactos positivos e negativos no trabalho e nas relações humanas causados pelos avanços da telepresença foi iniciada por Minsky e outros acadêmicos da época e continua até hoje, ganhando maior importância a cada dia (DONNELLY, R.; JOHNS, J. 2021). O projeto descrito neste trabalho faz uso do conceito de telepresença na tentativa de desenvolver uma interface imersiva, dentro das limitações do robô e da conexão que serão utilizadas no projeto final do Robô Museu.

2.3. Técnicas de Transmissão de Dados pela Internet

A Internet tornou- se uma ferramenta indispensável para a sociedade atual, pela qual é possível compartilhar informações entre computadores no mundo todo conectados à rede global, utilizando uma coleção de protocolos TCP/IP. Os protocolos definem as regras gerais de comunicação entre dois computadores ou processos interligados pela rede, assim garantem que eles "falem a mesma língua" e consigam processar corretamente os dados recebidos pela conexão (ELIAS, G.; LOBATO, L. C. 2013, p. 56-62). Os protocolos da família TCP/IP são divididos por funcionalidade entre 4 camadas de abstração, Aplicação, Transporte, Rede e Interface de Rede. Para este trabalho são utilizados protocolos dentro da família TCP/IP que garantem baixa latência na transmissão dos dados e que são reconhecidos pela maioria dos navegadores de uso comum, como Google Chrome, Firefox, Safari, etc. Os protocolos mais importantes para o entendimento do trabalho são descritos nos subtópicos a seguir.

2.3.1. HTTP/HTTPS

Hypertext Transfer Protocol (HTTP)[8] é um protocolo utilizado para transmissão de documentos chamados de *Hypermedia*, como por exemplo páginas HTML, imagens e arquivos JavaScript. Basicamente ele segue o modelo cliente-servidor, no qual o cliente abre uma conexão e envia uma requisição para o servidor. O servidor, então, envia uma resposta com os arquivos requisitados pelo cliente ou com uma mensagem de erro (ELIAS, G.; LOBATO, L. C. 2013, p. 377). Utilizando um navegador para fazer as requisições, o usuário verá os arquivos de resposta apresentados em uma interface amigável que esconde toda a troca de mensagens com o servidor e processos internos necessários para exibir certos arquivos corretamente, como códigos HTML (GARSIEL, T.; IRISH, P., 2011).

HyperText Transfer Protocol Secure (HTTPS)[9] é basicamente a versão criptografada do protocolo HTTP e utiliza protocolos de segurança de camada de transporte, o SSL ou o TLS, proporcionando uma conexão segura entre cliente e servidor, atualmente a segurança de dados é um tópico muito discutido e se torna cada dia mais importante, ao passo que as pessoas estão cada vez mais conectadas, por conta disso a maioria dos sites estão adotando o protocolo HTTPS ao invés do HTTP (DURUMERIC, Z. et al., 2013).

[8] "HTTP - MDN Web Docs." https://developer.mozilla.org/en-US/docs/Web/HTTP/.
[9] "HTTPS - Glossário - MDN Web Docs." https://developer.mozilla.org/pt-BR/docs/Glossary/https.

2.3.2. WebSockets

Utilizando os protocolos HTTP/HTTPS para fazer a comunicação entre cliente e servidor é sempre necessário que o cliente envie uma requisição para o servidor para que ele possa receber uma resposta, mas para aplicações que necessitam de uma resposta do servidor em tempo real essa configuração não é a ideal. O protocolo WebSocket[10] foi desenvolvido para estes casos. Ele permite estabelecer uma comunicação bidirecional entre o cliente e o servidor por uma única conexão TCP/IP. O cliente envia uma requisição HTTP/HTTPS com o cabeçalho UPGRADE, se o servidor oferecer suporte para o WebSocket ele envia uma resposta com "101 Switching Protocols", dessa forma a conexão full-duplex é estabelecida e ambos podem enviar mensagens por esta conexão a qualquer momento alterando os protocolos HTTP/HTTPS para o WebSocket, até que um dos lados feche a conexão. Dessa forma, em uma aplicação web em que são necessários envios constantes de dados ou atualizações em tempo real, o cliente não precisa enviar requisições para o servidor para obter esses dados, o servidor pode simplesmente enviar a mensagem quando houver necessidade (IETF, 2011).

2.3.3. Peer-to-Peer e ICE

Para uma aplicação que necessita a transferência de dados com baixa latência entre dois clientes, utilizar um servidor central para repassar os dados usando o modelo cliente-servidor pode ser muito lento e custar muitos recursos do servidor, nestes casos, e como é o caso deste projeto, uma opção é usar o modelo peer-to-peer, ou seja, fazer com o que os clientes estabeleçam uma conexão direta entre eles sem utilizar um servidor para repassar os dados. Desta forma os clientes, que agora serão chamados de *peers*, enviam os dados diretamente para o endereço um do outro (SHIRKY, C. 2000).

Um dos problemas da conexão peer-to-peer é garantir a segurança e a confiança dos dados, por conta disso, atualmente, a maioria dos roteadores de redes privadas dificulta conexões desse tipo entre máquinas locais e máquinas fora da rede privada, por conta de um método chamado *network address translators* (NATs), que altera o endereço IP da máquina local para o endereço IP público do roteador, e então todos os dados endereçados para a rede privada são controlados pelo roteador que retransmite para o endereço IP local correto, assim

[10] "WebSocket – Wikipédia, a enciclopédia livre." https://pt.wikipedia.org/wiki/WebSocket. Acessado em 23 nov.. 2021.

o endereço privado interno das máquinas da rede não é exposto (ELIAS, G.; LOBATO, L. C. 2013, p. 137). Como um *peer* dentro de uma rede privada não tem acesso ao endereço público da sua rede, ele não consegue informar ao outro *peer* fora da rede seu endereço correto para estabelecer a conexão direta. Para solucionar este problema e também o problema de *firewalls* que bloqueiam este tipo de conexão com *peers* não confiáveis, foi criada a técnica chamada Interactive Connectivity Establishment (ICE), que utiliza os protocolos *Session Traversal Utilities for NAT* (STUN) e *Traversal Using Relays around NAT* (TURN) para contornar as barreiras criadas por redes privadas (IETF, 2018).

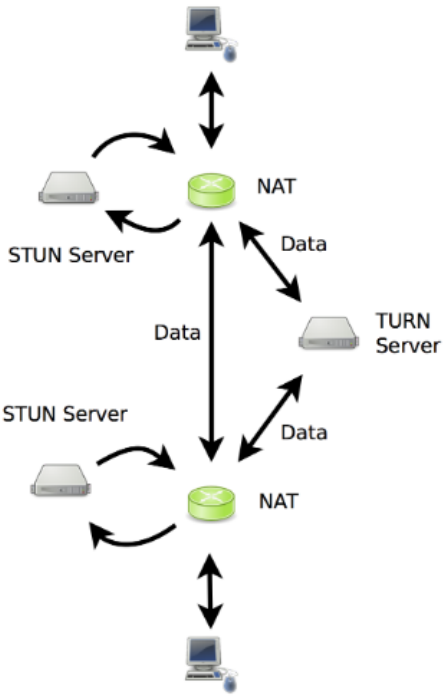
2.3.4. STUN e TURN

Session Traversal Utilities for NAT (STUN) é um protocolo cliente-servidor em que o cliente, dentro de uma rede privada, envia uma requisição para o servidor na rede pública que oferece o serviço de STUN, chamado de STUN *Server*, e então o servidor envia uma resposta contendo informações sobre o tipo de NAT que a rede privada do cliente utiliza, o endereço de IP e a porta públicos utilizados pelo NAT de sua rede que aceitem uma conexão *peer-to-peer* (NTWG, 2008). Em posse dessa informação sobre si mesmo, o cliente pode seguir o protocolo ICE e enviar esta informação como uma 'oferta' para uma outra máquina que deseja-se estabelecer uma conexão *peer-to-peer*, este outro *peer* então deve utilizar o STUN da mesma forma e enviar uma 'resposta' de volta para o *peer* inicial com as informações sobre ele obtidas pelo STUN, e por fim cada *peer* poderá utilizar a informação recebida do seu par para estabelecerem a conexão. A forma como a 'oferta' e a 'resposta' são trocadas não é especificada pelo protocolo ICE, mas a recomendação é fazer esta troca de maneira segura e criptografada, em muitas aplicações é utilizado um servidor web confiável para transmitir estas informações.

Existem casos em que somente o STUN não é suficiente para contornar o NAT de algumas redes, um dos exemplos é quando o NAT altera o mapeamento dos endereços locais da rede privada dependendo do endereço do remetente da mensagem, e se o endereço do remetente está presente na sua tabela interna de endereços confiáveis (SRISURESH, P. et al., 2008, p. 4). Nestes casos é utilizado o protocolo *Traversal Using Relays around NAT* (TURN), ele define o uso de servidores intermediários, chamados de TURN *Servers*, que irão retransmitir a conexão entre os *peers*. Um *peer* que deseja utilizar um TURN *Server* deve enviar uma requisição para o servidor que disponibiliza esse serviço, e então o servidor irá responder com uma tupla, endereço de IP e porta, que o servidor deixará disponível para a

conexão *peer-to-peer*, esta tupla que será repassada para o outro *peer* como ‘oferta’ ou ‘resposta’, junto com outras informações definidas pelo protocolo ICE. Por fim, quando o outro *peer* enviar alguma mensagem pela conexão, ele enviará a mensagem para o TURN *Server* que então repassará a mensagem para o endereço do outro *peer* (IETF, 2010).

Figura 2: Exemplo de uso de STUN e TURN



Fonte: GUNAY MERT KARADOGAN, Evaluating WebSocket and WebRTC in the Context of a Mobile Internet of Things Gateway (2013).

A Figura 2 apresenta um exemplo esquemático de como funcionam o STUN *Server* e o TURN *Server*, ressaltando que os *peers* podem utilizar servidores de STUN diferentes e como o servidor de TURN se encaixa entre a conexão dos *peers*. Utilizar servidores de TURN pode acabar diminuindo o valor da principal vantagem de se utilizar uma conexão *peer-to-peer*, que seria a redução da latência por não utilizar um intermediário, por conta disso que o protocolo ICE define parâmetros para que só seja utilizado o TURN caso o método com STUN não funcione.

2.4. Bibliotecas e Softwares de Código Aberto

Durante o processo de desenvolvimento deste trabalho, algumas bibliotecas e ferramentas de software foram utilizadas para auxiliar na obtenção da aplicação final com todos os requisitos necessários. As mais importantes para a solução obtida serão apresentadas nessa seção, o resto dos softwares utilizados serão discutidos brevemente no Capítulo 3. Os softwares aqui apresentados auxiliam na implementação dos aspectos chave da aplicação apresentados nas seções anteriores.

2.4.1. NGINX

NGINX[11] é um servidor HTTP gratuito e de código aberto, que também pode atuar como um servidor de *proxy* reverso, que é quando um servidor é usado para retransmitir requisições de clientes para outros servidores, muito usado por grandes websites para balancear a carga dos servidores. O servidor NGINX utiliza uma arquitetura assíncrona baseada em eventos, o que o torna uma solução escalável até para grandes websites com muito tráfego, como por exemplo Netflix[12], Hulu[13] e Pinterest[14]. O servidor NGINX é um software gratuito, Open Source e está entre os servidores web mais populares segundo pesquisa mensal feita por NETCRAFT (2021), além de ser mais responsivo e escalável do que seu maior competidor, o Apache Web Server[15], segundo PRAKASH et al. (2015).

2.4.2. Node.js

Node.js[16] é um software de código aberto e multiplataforma que define a si próprio como um ambiente de execução de códigos JavaScript, ele foi introduzido em 2009 e desde então ganhou popularidade para uso em aplicações *server-side* por sua velocidade, simplicidade e natureza assíncrona. O Node.js usa como base a máquina virtual Google Chrome V8[17] para compilar o código JavaScript, e a maior diferença de sua arquitetura para outras tecnologias é o fato de executar o código em *single-thread* e ser *event-driven*, ou seja, ele utiliza uma única *thread* para executar todas as requisições, e cada requisição é tratada como um evento não-bloqueante. Outras tecnologias para o mesmo uso são *multi-thread*. Isso

[11] "NGINX Wiki" https://www.nginx.com/resources/wiki/.
[12] "Netflix" https://www.netflix.com/br/.
[13] "Hulu" https://www.hulu.com/.
[14] "Pinterest - Brasil." https://br.pinterest.com/.
[15] "The Apache HTTP Server Project." https://httpd.apache.org/.
[16] "node/README.md - GitHub." https://github.com/nodejs/node/blob/master/README.md.
[17] "V8 JavaScript engine." https://v8.dev/docs.

significa que a cada nova requisição uma nova *thread* é criada no servidor, consumindo mais recursos (CHANIOTIS, I. K., 2015).

O Node.js também é amplamente utilizado por desenvolvedores web por utilizar a linguagem JavaScript, que também é a linguagem padrão utilizada por aplicações web no *client-side* e pode ser carregada em arquivos HTML, por conta disso a maioria dos desenvolvedores da área já possui familiaridade com a linguagem, o que também simplifica o desenvolvimento da aplicação, possibilitando o uso da mesma linguagem tanto no *frontend* quanto no *backend*. Outra vantagem do Node.js é o seu gerenciador de pacotes *Node Package Manager* (NPM)[18], que além de gerenciar todos os múltiplos pacotes de software que uma aplicação Node.js pode utilizar simultaneamente, o NPM também possui o maior repositório de software do mundo, com mais de 1 milhão de pacotes de código aberto.

2.4.3. Socket.IO

Socket.IO é uma biblioteca de código aberto presente no repositório do NPM que simplifica o uso do protocolo WebSockets entre cliente e servidor utilizando Node.js. Esta biblioteca também oferece suporte para falhas na conexão WebSocket, substituindo o protocolo automaticamente para um HTTP *long polling* caso a conexão por meio do WebSocket não consiga ser estabelecida. Além disso, a Socket.IO permite que o cliente tente se reconectar automaticamente caso a conexão seja perdida. Estas características trazem confiabilidade para o software, além de simplificar o código da aplicação (KARADOGAN, G. M. 2013, p. 19).

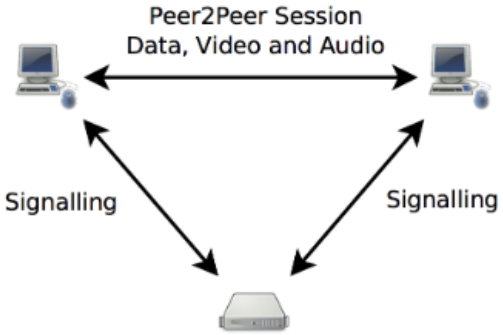
2.4.4. WebRTC

Web Real-Time Communication (WebRTC) é um projeto open source com o propósito de possibilitar o desenvolvimento de aplicações em tempo real de qualidade em navegadores e plataformas móveis, e também definir protocolos padrões para a comunicação. É uma tecnologia já utilizada na área da telepresença robótica com sucesso (MELENDEZ-FERNANDEZ et al, 2017). Ele permite a troca de dados em tempo real por canais específicos de áudio, vídeo e mensagens, utilizando conexões *peer-to-peer* estabelecidas pelo protocolo ICE. O WebRTC utiliza Interfaces de Programação de Aplicação (APIs) para gerenciar a conexão *peer-to-peer*: (i) fazendo a escolha das técnicas mais adequadas para fazer a conexão, como UDP ou TCP, usar ou não um TURN *Server*, entre

[18] "npm." https://www.npmjs.com/.

outras, dependendo da configuração de cada *peer*; (ii) obtendo os fluxos de dados de áudio e vídeo de cada usuário pelos navegadores; (iii) criando e gerenciando os diversos canais de dados dentro da conexão para os fluxos de áudio e de vídeo e canais de dados arbitrários, estes chamados de *Data Channels* (W3C, 2021).

Figura 3: Arquitetura do WebRTC



Fonte: GUNAY MERT KARADOGAN, Evaluating WebSocket and WebRTC in the Context of a Mobile Internet of Things Gateway (2013).

A Figura 3 ilustra a arquitetura usual de uma aplicação utilizando WebRTC, os dois *peers* necessitam de um servidor central para fazerem a sinalização, que se trata basicamente da troca das informações descritas nos protocolos ICE, STUN e TURN necessárias para estabelecer a conexão *peer-to-peer*. Um importante ponto para este trabalho é que o WebRTC também suporta arquiteturas de comunicação diferentes da um-para-um, como a um-para-muitos que será implementada neste trabalho. A arquitetura um-para-muitos faz basicamente um usuário principal, que será chamado de transmissor, faz diversas conexões *peer-to-peer* com usuários diferentes, que serão chamados de espectadores, e então o transmissor transmite os mesmos fluxos de dados para todos os espectadores simultaneamente (KARADOGAN, G. M. 2013).

2.5. Considerações Finais

Neste capítulo foram revisadas técnicas da área de telerobótica e sua relação com este trabalho e o projeto do Robô Museu. Também foram apresentadas as terminologias mais importantes para o trabalho e discutidas algumas das técnicas, protocolos e tecnologias usadas para desenvolver uma aplicação de comunicação de áudio, vídeo e texto em tempo real, utilizando a Internet como meio. No capítulo seguinte será descrito o desenvolvimento deste trabalho utilizando os elementos apresentados neste capítulo.

CAPÍTULO 3: DESENVOLVIMENTO DA INTERFACE DE CONEXÃO COM O ROBÔ

3.1. Considerações iniciais

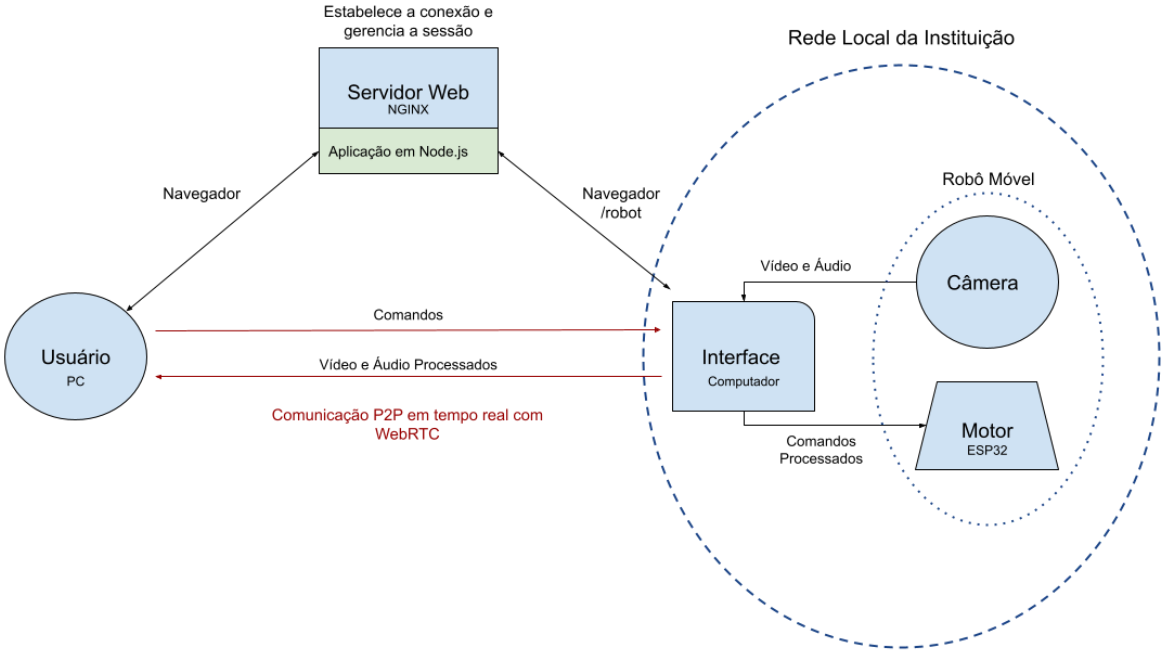
Neste capítulo o desenvolvimento deste trabalho será descrito em detalhes. Serão apresentados os passos de todo o processo de desenvolvimento junto aos resultados obtidos de cada etapa do trabalho e discutir-se-ão as falhas encontradas e subsequentes correções e revisões adotadas durante o desenvolvimento. Ao final, serão discutidas as principais dificuldades de execução e limitações do trabalho desenvolvido e serão sintetizadas as lições aprendidas durante o decorrer deste projeto.

3.2. Projeto

Este trabalho visa construir uma aplicação web que possa ser utilizada como interface para controlar um robô remotamente e tenha suporte para reprodução ao vivo de vídeo. Para isso, primeiramente é preciso definir toda a arquitetura da comunicação entre múltiplos usuários e o robô, os usuários utilizando a aplicação web por meio de um navegador de internet, e o robô conectado à rede Wi-Fi da instituição.

Na Figura 4 é apresentada a arquitetura geral da aplicação que foi desenvolvida, ela é dividida em 3 módulos principais: (i) o computador do usuário que se conectará por meio de um navegador com o servidor central e em seguida com o computador de interface do robô; (ii) o servidor web central que estará servindo a aplicação em Node.js e as páginas HTML para o usuário e para a interface utilizando o protocolo HTTPS, a aplicação Node.js será responsável por fazer a troca de sinais do protocolo ICE entre o usuário e a interface; (iii) a interface por sua vez também se conectará com o servidor central e depois com o usuário utilizando um navegador web, e após completa a sinalização descrita no protocolo ICE será estabelecida a conexão *peer-to-peer* com o usuário utilizando o WebRTC.

Figura 4: Arquitetura Simplificada da Aplicação.



3.3. Desenvolvimento da Interface Proposta

Nesta seção serão descritos os módulos apresentados na seção anterior com mais detalhes, e também as etapas do processo de desenvolvimento da aplicação como um todo.

3.3.1. Definição da Arquitetura e das Tecnologias

Existem diversos métodos diferentes de se fazer a conexão entre duas máquinas pela internet, portanto o passo inicial é decidir qual desses métodos é o mais apropriado para este trabalho. Para isso é preciso definir as características mais críticas, as quais o método de comunicação escolhido deve garantir para o sucesso da aplicação.

Esta primeira parte do desenvolvimento do projeto exige o estudo extensivo das tecnologias atuais e então a realização de testes para a comparação entre os métodos estudados para a definição mais aceitável da arquitetura. Primeiramente foi escolhido utilizar uma conexão *peer-to-peer* entre o usuário e a interface do robô, esta solução garante uma menor latência na comunicação comparada com a outra solução considerada de utilizar o servidor central para transmitir toda a comunicação entre os usuários. A proposta de utilizar um servidor para transmitir os fluxos de áudio e vídeo foi testada localmente em uma rede privada e utilizando o protocolo *Real-Time Messaging Protocol* (RTMP). Mesmo os dados não passando pela Internet foi obtida uma latência de aproximadamente 10 segundos, o que

foi considerado alta demais para garantir uma boa experiência de usuário. Isso se deve ao fato do usuário ter que receber a imagem da câmera do robô, identificar um caminho a seguir e controlar o robô para desviar de obstáculos e se aproximar do ponto de destino, uma latência de 10 segundos poderia resultar em dificuldades no controle preciso do robô. Em seguida foi testada a solução *peer-to-peer* utilizando o protocolo WebRTC, e foi obtida uma latência consistente abaixo de 1 segundo, considerada mais que satisfatória seguindo as condições de sucesso determinadas nos objetivos deste trabalho.

Um artigo publicado por Melendez-Fernandez et al. (2017) no *International Journal of Advanced Robotic Systems* propõe uma arquitetura para telepresença robótica pela Web utilizando WebRTC para a comunicação, mostrando resultados práticos positivos. Segundo o artigo, entre as vantagens da arquitetura apresentada estão a compatibilidade multi-plataforma, o uso de tecnologias *open-source* e uma interface simples e amigável para usuários inexperientes, sem necessidade de instalação ou uso de softwares por parte do usuário, fora o próprio navegador web. Esta pesquisa apresentou uma arquitetura com a maioria das características desejadas no projeto do Robô Museu, e por este motivo ela serviu de base para a implementação deste trabalho.

Após a escolha do método de conexão, é definida a arquitetura da aplicação como um todo, baseada na arquitetura proposta por Melendez-Fernandez et al. (2017) e adaptando para o uso do projeto Robô Museu, tendo em vista os casos de uso e os requisitos para se utilizar o WebRTC. Como ilustrado na Figura 4, a arquitetura da aplicação é composta por:

- Um servidor web remoto NGINX que servirá a aplicação em NodeJS contendo uma página para os usuários acessarem e o usuário principal controlar o robô, e outra página específica para ser acessada pela interface do robô para transmitir os fluxos de áudio e vídeo e receber os comandos do usuário. A aplicação também será responsável por repassar os sinais do protocolo ICE entre o usuário e a interface necessários para estabelecer a conexão *peer-to-peer* entre eles. É importante ressaltar que os fluxos de dados, tanto de áudio e vídeo do robô, quanto os comandos do usuário não serão transmitidos pelo servidor, então mesmo se a conexão com o servidor for perdida, a conexão *peer-to-peer* é mantida normalmente, e o servidor não terá como visualizar os dados que estão sendo trocados pelos usuários e o robô.
- Um computador conectado à mesma rede local do robô via Wi-Fi, que servirá como interface e processará e repassará tanto os comandos recebidos do

usuário para o robô, quanto o fluxo de dados de áudio e vídeo, vindos da câmera e microfone do robô, para o usuário. É recomendado que este computador tenha uma conexão rápida e estável com a Internet. É possível que seja preciso instalar neste computador um software de terceiros para poder capturar os fluxos de dados vindos da câmera do robô, como o OBS Studio[19], mas isso depende do modelo da câmera instalada no robô e está além do escopo deste trabalho.

- O robô móvel que será controlado. Este projeto será baseado na montagem apresentada na página do GitLab do projeto Robô Museu[20], que no caso se trata de um robô móvel controlado por um microcontrolador ESP32[21] e uma câmera IP montada em um suporte fixo. A ESP32 e a câmera estarão conectados via Wi-Fi na mesma rede que o computador usado para a interface. A câmera IP enviará sua gravação para o computador da interface e a ESP32 receberá os comandos dele por esta conexão Wi-Fi.
- O computador do usuário que irá acessar a página web. Por essa página serão realizadas todas as ações necessárias, desde estabelecer a conexão, como também reproduzir o vídeo e áudio em tempo-real e mandar os comandos para o robô. Será necessário que o computador do usuário possua uma boa conexão com a internet para garantir a menor latência possível, e acessar a página por meio de um navegador atualizado que suporte o WebRTC. Hoje em dia, os navegadores mais comumente utilizados possuem suporte para o WebRTC.

3.3.2. Configuração do Servidor Web

Definida a arquitetura da aplicação, o próximo passo é a implementação da mesma. No primeiro momento o servidor foi a preocupação inicial, a solução foi utilizar um servidor já estabelecido e em funcionamento, que é utilizado para servir a página do projeto de extensão Principia Robôs na Escola[22]. O servidor do Principia é fornecido pelo Instituto de Ciências Matemáticas e de Computação (ICMC) da USP, e roda atualmente a versão do

[19] "Open Broadcaster Software | OBS." https://obsproject.com/.
[20] Repositório Robô Museu - Eduardo do Valle Simões - GitLab https://gitlab.com/simoesusp/robo-museu.
[21] "ESP32 Wi-Fi & Bluetooth MCU I Espressif Systems." https://www.espressif.com/en/products/socs/esp32.
[22] "Principia - Projeto Robôs na Escola - USP." https://principia.icmc.usp.br/.

Ubuntu 16.04[23] e o servidor web NGINX Open Source. Como o servidor já estava em funcionamento e servindo uma página em sua porta de acesso padrão, o único trabalho necessário foi o de configurar o servidor NGINX para atuar como um proxy reverso[24], para isso foram adicionadas dentro do arquivo de configuração do NGINX 'sites-available/default' as linhas apresentadas no Código Fonte apresentado na Figura 5.

Figura 5: Configuração de proxy-reverso do servidor.

```
location /robo-museu/ {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $host;
        proxy_pass_request_headers on;
        proxy_pass https://127.0.0.1:4000/;

        # Enables WebSockets
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
}

location  /socket.io/ {
        proxy_pass https://principia.icmc.usp.br/robo-museu/socket.io/;
}

location /watch.js {
        proxy_pass https://principia.icmc.usp.br/robo-museu/watch.js;
}

location /robot/broadcast.js {
        proxy_pass https://principia.icmc.usp.br/robo-museu/robot/broadcast.js;
}

location /robot/broadcast-bundle.js {
        proxy_pass https://principia.icmc.usp.br/robo-museu/robot/broadcast-bundle.js;
}

location /styles.css {
        proxy_pass https://principia.icmc.usp.br/robo-museu/styles.css;
}
```

Assim as requisições para para o endereço 'https://principia.icmc.usp.br/robo-museu' serão redirecionadas para a porta 4000 do servidor, esta foi a porta escolhida para o servidor da aplicação, todas as requisições enviadas para ela serão recebidas e respondidas pelo servidor web executando em Node.JS, cujo código se encontra no arquivo server.js[25].

Com o servidor funcionando e preparado, a próxima etapa da implementação é construir a aplicação principal em Node.js.

---

[23] "Ubuntu 16.04 LTS (Xenial Xerus)." https://ubuntu.com/16-04.

[24] "Proxy Reverso" https://www.profissionaisti.com.br/proxy-reverso-uma-seguranca-a-mais-para-seu-ambiente/.

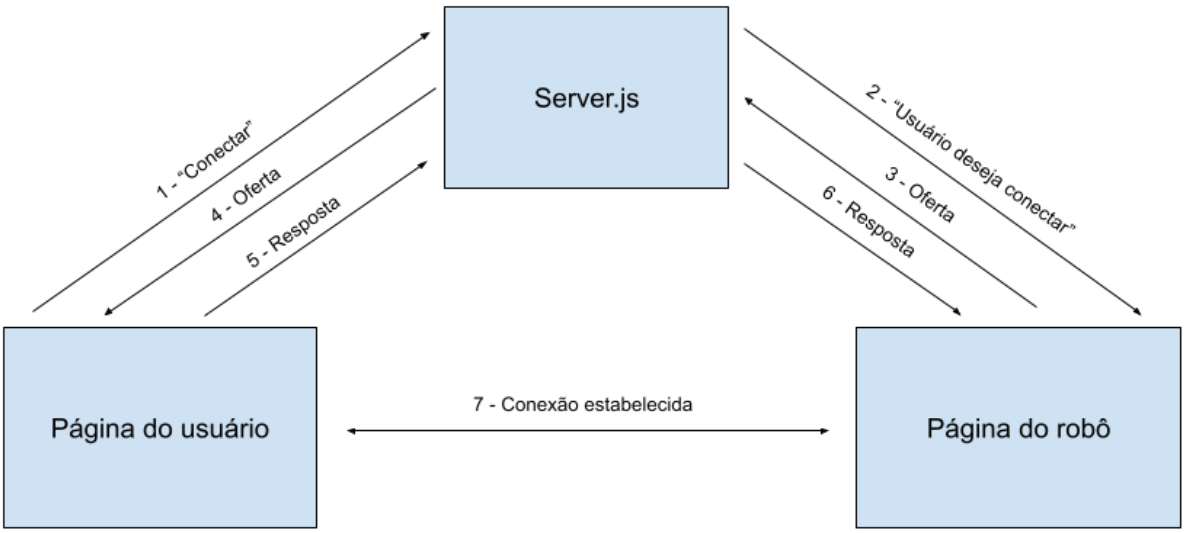
[25] Arquivo server.js https://gitlab.com/simoesusp/robo-museu/-/blob/master/MURILO/robot-remote-interface/server.js.

3.3.3. Implementação da Aplicação Node.js

A aplicação Node.js em si pode ser dividida em três partes, como os módulos deste trabalho, definidas pelos arquivos de código JavaScript usados pela aplicação. Todos os arquivos de código descritos neste trabalho estão disponíveis publicamente no repositório do projeto Robô Museu[26] no Gitlab sob a licença Open Source MIT[27]. Estes arquivos são: (i) o arquivo server.js, rodando no servidor, que cuidará da troca de informações necessárias para estabelecer a conexão peer-to-peer entre o usuário e o controlador do robô; (ii) o arquivo watch.js, o qual será carregado como um *script* na página HTML do usuário para obter os comandos inseridos pelo mesmo nos elementos HTML da página, e convertê-los em mensagens que serão enviadas para o servidor ou para o controlador do robô, e o código também será responsável por receber os dados enviados pelo robô e exibi-los para o usuário; e por fim (iii) o arquivo broadcast.js, que será utilizado na página acessada pelo controlador do robô, responsável pela troca de mensagens com o servidor, enviar os dados de áudio e vídeo para o usuário, e também processar e depois transmitir os comandos do usuário para o microcontrolador do robô. Lembrando que o código responsável por interpretar os comandos e controlar os motores, usado pelo microcontrolador, não faz parte do escopo deste projeto.

A Figura 6 ilustra um caso de uso em que o usuário e o controlador do robô estabelecem com sucesso uma conexão WebRTC, utilizando o servidor como intermediário.

Figura 6: Caso de uso de sucesso de conexão.



[26] Repositório da aplicação - GitLab https://gitlab.com/simoesusp/robo-museu/-/tree/master/MURILO/robot-remote-interface.

[27] "The MIT License | Open Source Initiative." https://opensource.org/licenses/MIT.

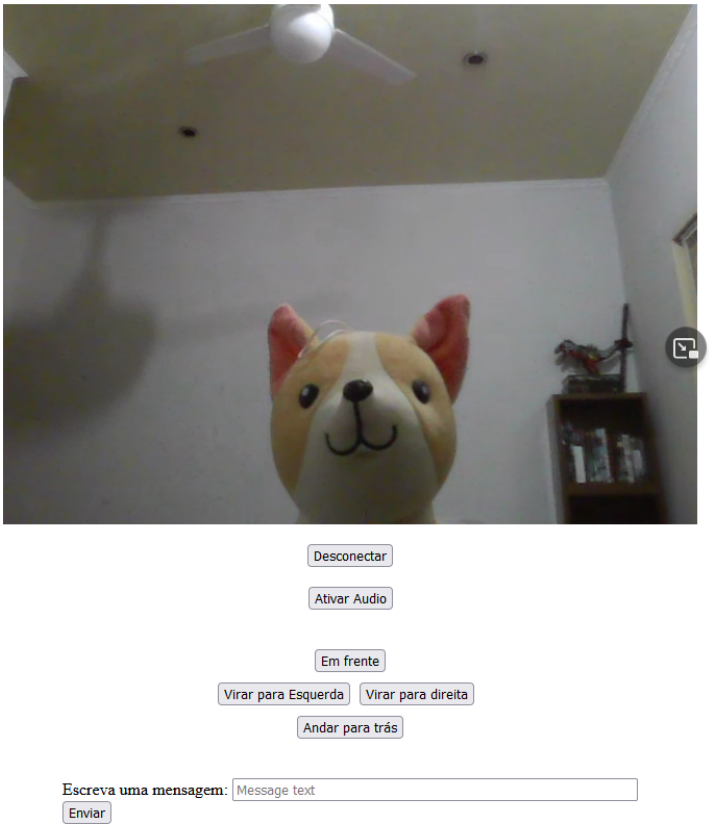
No arquivo de código server.js foram utilizados os seguintes módulos obtidos pelo *Node Package Manager* (NPM). Express.js[28], que é Framework para o desenvolvimento de aplicações web para Node.js e é utilizado para servir as páginas HTML para os usuários e para a interface do robô ao executar o código do server.js pelo Node.js. O módulo HTTPS[29], que é utilizado para criar o servidor com o protocolo HTTP utilizando TLS/SSL para certificar a segurança, o website principia.icmc.usp.br no qual esta aplicação está sendo servida já utiliza o protocolo HTTPS, portanto a aplicação deste trabalho fará uso dos mesmos certificados de segurança fornecidos pela Autoridade Certificadora (CA) Let's Encrypt[30]. O módulo Socket.io, apresentado no Capítulo 2, tem como função principal implementar o protocolo WebSocket que é utilizado pelos usuários e pela interface do robô para enviarem a sinalização ICE e outras informações importantes para o servidor. O Código Fonte apresentado na Figura 7, é um exemplo de como é utilizado o Socket.io nesta aplicação, o comando 'socket.on()' cria uma função assíncrona que espera por uma mensagem da interface pelo WebSocket com o título "mainWatcher", e então ao receber uma mensagem com este título o servidor repassa a mensagem para o usuário cujo *id* é igual ao *id* que foi passado na mensagem da interface. Esta troca de mensagens em específico tem a função de avisar a página do usuário que este se trata de um usuário principal, e sabendo disso, a página HTML mostrará na tela o painel de comandos do robô, como é visível na Figura 8.

Figura 7: Exemplo em código de uso do WebSocket no server.js.

```
53    socket.on("mainWatcher", (id, message) => {
54      console.log("new main watcher");
55      socket.to(id).emit("mainWatcher", socket.id, message);
56    });
57  });
```

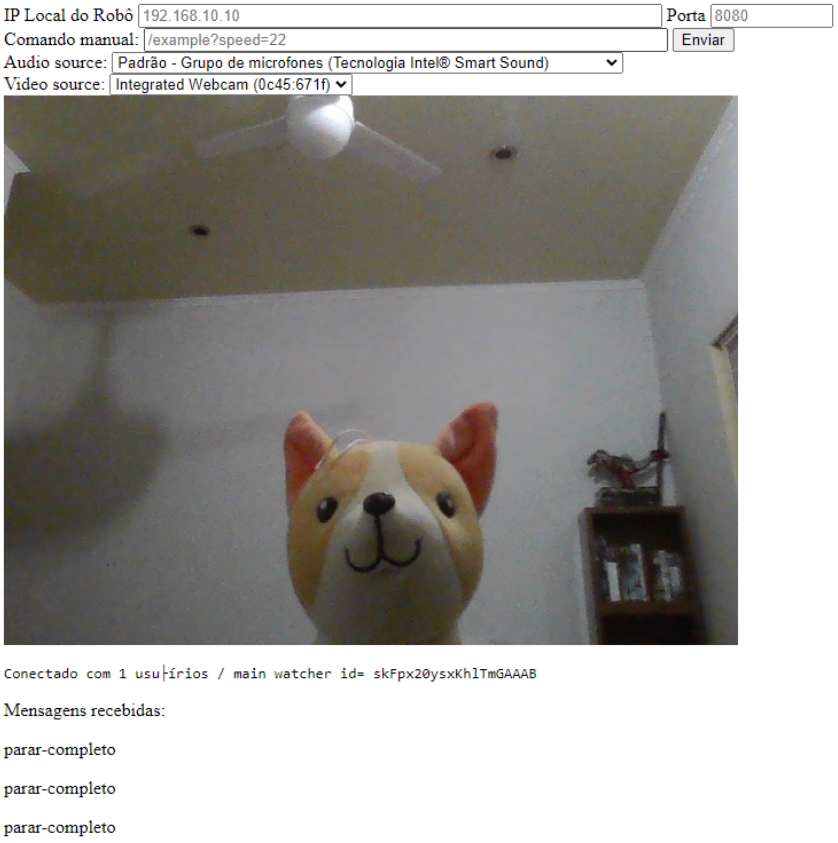
[28] "Express - framework de aplicativo da web Node.js." https://expressjs.com/pt-br/.
[29] "HTTPS | Node.js v17.1.0 Documentation." https://nodejs.org/api/https.html.
[30] "ISRG CP v3.1 - Let's Encrypt." https://letsencrypt.org/documents/isrg-cp-v3.1/.

Figura 8: Tela do usuário principal conectada ao robô.



Após implementado o código do servidor o próximo passo foi implementar os códigos das páginas de usuário e da interface. O arquivo de código watch.js é carregado como um *script* HTML na página fornecida para os usuários. A página em si possui apenas um botão de 'Conectar' na tela. Ao clicar nesse botão o código em watch.js tomará conta de fazer toda a sinalização necessária para estabelecer a conexão, começando pelo envio de uma mensagem para o servidor pelo WebSocket que se deseja conectar. A partir deste ponto, desde que a interface esteja conectada, o usuário e a interface trocarão os sinais de 'oferta' e 'resposta' definidas pelo protocolo ICE usando o servidor central como meio de comunicação até que a conexão seja estabelecida. Caso a interface não tenha ainda um usuário principal conectado, este título será garantido para o novo usuário da forma que foi discutida no parágrafo anterior. Com a conexão WebRTC estabelecida, o vídeo fornecido pela interface será apresentado para o usuário em tempo real, caso o usuário obtenha o título de 'main watcher' o painel de comandos estará visível como aparece na Figura 8.

Figura 9: Tela da interface do robô.



Já o arquivo de código broadcast.js é carregado como um *script* na página HTML acessada pela interface, como demonstrado na Figura 9. Nela o controlador poderá visualizar o vídeo que será compartilhado, a quantidade atual de usuários conectados, o id do usuário principal e uma lista com os comandos recebidos do usuário. Todas essas informações são atualizadas pelo código broadcast.js em tempo real. É necessário que o controlador preencha dois dos campos de texto manualmente, um com o endereço IP local do robô e outro com a porta que o microcontrolador do robô utiliza para receber os comandos por Wi-Fi, essas informações serão usadas para enviar os comandos para o microcontrolador. O controlador ainda poderá selecionar as fontes de áudio e vídeo caso haja mais de uma. O código da interface sempre irá esperar um usuário tentar se conectar. Quando isso acontecer, o código em broadcast.js ficará responsável por estabelecer a conexão WebRTC com o usuário automaticamente.

3.4. Dificuldades e Limitações

Os navegadores mais modernos suportam por padrão uma interface JavaScript para gerenciar as conexões WebRTC, com métodos para obter candidatos ICE, usar servidores STUN e TURN, criar *Data Channels*, etc. Esta foi a técnica utilizada na implementação deste trabalho. Apesar de existirem módulos no repositório do NPM que dizem simplificar a implementação do WebRTC, foi preferível utilizar os métodos padrões, pois em testes realizados com os módulos do NPM, todos apresentaram alguma desvantagem para a aplicação desenvolvida.

Ao implementar a parte do código responsável por enviar os comandos do computador da interface para o microcontrolador do robô, o autor deparou-se com um problema - o módulo necessário para enviar as requisições HTTP com os comandos não era disponibilizado para ser utilizado de maneira *client-side*, ou seja, apenas o código do servidor poderia enviar requisições HTTP automaticamente e não o navegador. A solução foi utilizar um software chamado Browserify[31] para tornar isso possível. O Browserify analisa o código recursivamente e agrupa todos os módulos importados no código em um único arquivo chamado de *bundle*. Ao servir este *bundle* como um *script* na página no lugar do broadcast.js, o navegador consegue interpretar todo o código em JavaScript, incluindo o módulo HTTPS que era necessário para fazer as requisições para o endereço de IP do microcontrolador. O controlador só terá o trabalho de liberar o envio de requisições ao IP do microcontrolador pelo navegador, já que a maioria dos navegadores classifica uma requisição HTTPS para um IP local como não segura, pois não é possível garantir a autenticidade dos certificados TLS/SSL.

O próximo passo é o de testes da aplicação. Nesta etapa os arquivos de código foram clonados direto no servidor usando um repositório Git[32] remoto pela plataforma GitLab[33], e foram executados com o Node.js. Também foi criada uma pequena aplicação JavaScript que simula a presença do robô conectado à rede local do controlador, disponível no mesmo repositório do projeto Robô Museu com o nome de ‘command-receiver’[34], esta aplicação apenas recebe os comandos enviados pelo controlador e os exibe no terminal, e seu objetivo era possibilitar os testes mesmo sem a presença de um robô real.

---

[31] "browserify/browserify - GitHub." https://github.com/browserify/browserify.
[32] "user-manual Documentation - Git." https://git-scm.com/docs/user-manual.
[33] "GitLab: Iterate faster, innovate together." https://about.gitlab.com/.
[34] Aplicação ‘command-receiver’ https://gitlab.com/simoesusp/robo-museu/-/tree/master/MURILO/command-receiver.

A aplicação foi testada tanto localmente, quanto remotamente. Foram encontradas algumas falhas de conexão por WebSockets durante os testes quando o usuário e o controlador do robô se encontravam em redes privadas separadas. Mas por conta das funcionalidades do módulo Socket.io a aplicação funciona normalmente mesmo com as falhas no WebSocket, já reconectando automaticamente ou substituindo o protocolo e para um HTTP *long polling*.

O último passo do projeto é o de análise dos resultados dos testes e correção de falhas. As falhas críticas foram resolvidas, enquanto algumas não-críticas foram deixadas para trabalhos futuros por falta de tempo. Mais testes são necessários para garantir a robustez da solução apresentada. Por fim, apesar dos problemas de conexão do WebSocket apresentados, a aplicação conseguiu conectar-se e transmitir os dados com sucesso entre o computador do usuário e o "robô" (simulação de um robô no caso deste trabalho), com uma latência média observada de aproximadamente 1 segundo, tudo utilizando uma conexão segura para os padrões atuais utilizando protocolos HTTPS, *WebSockets Secure* (WSS) e WebRTC.

3.5. Considerações Finais

Neste capítulo discutiu-se o desenvolvimento deste trabalho, a implementação do código da aplicação, e também as dificuldades e problemas encontrados ao longo do projeto. Foi construída uma aplicação para a interface entre usuário e robô através da web, utilizando protocolos seguros, e foram realizados testes de funcionalidade da aplicação. No capítulo seguinte, serão apresentadas as conclusões deste trabalho.

CAPÍTULO 4: CONCLUSÃO

Com os testes realizados ao final do processo de desenvolvimento foram obtidos resultados satisfatórios: a aplicação desenvolvida alcançou os objetivos de conectar múltiplos usuários à transmissão de áudio e vídeo do robô com uma latência média dentro das expectativas determinadas para a aplicação (menor que 2 segundos), sendo apenas um usuário, intitulado como principal, que pode enviar os comandos para o robô, que são corretamente recebidos pela interface e encaminhados para o microcontrolador do robô por meio da rede Wi-Fi.

Apesar do resultado positivo na funcionalidade da aplicação, o trabalho não conseguiu atingir o nível de qualidade gráfica esperado no início do desenvolvimento para a interface de usuário nas duas páginas HTML apresentadas. Este problema ocorreu devido ao fato de surgirem dificuldades durante o processo de desenvolvimento que demandaram maior atenção para outras áreas, para economizar tempo, foi decidido implementar uma interface de usuário simples apenas para demonstrar as funcionalidades do projeto.

4.1. Contribuições

Uma das contribuições deste trabalho é a de fornecer um modelo de aplicação de código aberto, que poderá ser utilizado para a implementação de sistemas de telepresença de maneira simples e acessível. Principalmente como sistema para o projeto do Robô Museu, para o qual este trabalho foi inicialmente moldado.

Outra contribuição é o aprendizado documentado neste trabalho sobre a implementação dos métodos de conexão utilizados com tecnologias de código aberto disponíveis atualmente, os desafios da área de telepresença e toda a sua complexidade. Destaca-se a importância de, ao início do processo de desenvolvimento de um trabalho, definir um cronograma com todas as etapas do processo reservando um período de tempo realista especialmente para correção de erros e cobrir eventuais dificuldades que atrasem o desenvolvimento.

4.2. Trabalhos Futuros

Por conta de sua natureza, este trabalho apenas pode ser considerado completo quando combinado com outros módulos em um sistema maior com testes em situações reais, como por exemplo, compondo o sistema de telepresença robótica em um museu. Entretanto, a aplicação desenvolvida neste trabalho ainda precisa de ajustes e correções antes de poder ser integrada de forma satisfatória em um sistema completo, como uma interface mais amigável para o usuário. Além disso, é necessário que outros sistemas sejam implementados para complementar a aplicação apresentada. Portanto, propõe-se os seguintes passos com o fim de chegar à implementação completa de um sistema robusto de telepresença robótica:

1. Implementação de um sistema de cadastro e autenticação de usuários;
2. Implementação de um sistema de agendamento de acesso ao robô;
3. Melhoria do *design* da interface para uma melhor experiência de usuário;
4. Testes de integração e de campo do sistema completo, com o robô em um local apropriado, com usuários pertencentes ao público alvo que se deseja atingir.

REFERÊNCIAS

UNESCO. **Pesquisa de percepção dos impactos da COVID-19 nos setores cultural e criativo do Brasil: resumo**. UNESCO Office in Brasilia, 2020. Disponível em: <https://unesdoc.unesco.org/ark:/48223/pf0000375069> Acesso em 16 nov. 2021.

MINSKY, Marvin. **TELEPRESENCE**. OMNI Magazine, Junho 1980. Disponível em: <https://web.media.mit.edu/~minsky/papers/Telepresence.html> Acesso em 16 nov. 2021.

KRISTOFFERSSON, A.; CORADESCHI, S.; LOUTFI, A. **A Review of Mobile Robotic Telepresence**, *Advances in Human-Computer Interaction*, vol. 2013, Article ID 902316, 17 pages, 2013. Disponível em: <https://doi.org/10.1155/2013/902316> Acesso em 18 nov. 2021.

MARAFA, N. A.; FILHO, W. B. V. **DESENVOLVIMENTO DE UM ROBÔ MÓVEL DE TELEPRESENÇA DESTINADO AO APOIO A PESSOAS COM DEFICIÊNCIAS LOCOMOTORAS**, 12º Congresso Brasileiro de Inovação e Gestão de Desenvolvimento de Produto, Blucher Engineering Proceedings, Volume 2, 2019, Pages 599-614, ISSN 2357-7592. Disponível em: <www.proceedings.blucher.com.br/article-details/desenvolvimento-de-um-rob-mvel-de-telepresena-destinado-ao-apoio-a-pessoas-com-deficincias-locomotoras-33590> Acesso em 20 nov. 2021.

SINGH, V.; LOZANO, A. A.; OTT, J. **Performance Analysis of Receive-Side Real-Time Congestion Control for WebRTC**, *2013 20th International Packet Video Workshop*, 2013, pp. 1-8, doi: 10.1109/PV.2013.6691454.

MELENDEZ-FERNANDEZ, F.; GALINDO, C.; GONZALEZ-JIMENEZ, J. **A web-based solution for robotic telepresence**. International Journal of Advanced Robotic Systems, November-December, 2017, pages 1-19. Disponível em: <https://doi.org/10.1177/1729881417743738> Acesso em 12 nov. 2021.

SCHILLING, K.; ROTH, H.; LIEB, R. **Teleoperations of rovers. From Mars to education**, *ISIE '97 Proceeding of the IEEE International Symposium on Industrial Electronics*, 1997, pp. SS257-SS262 vol.1, doi: 10.1109/ISIE.1997.651772.

SHERIDAN, T. B. Telerobotics, Automatica, Volume 25, Issue 4, 1989, Pages 487-507, ISSN 0005-1098. Disponível em: <https://doi.org/10.1016/0005-1098(89)90093-9> Acesso em 20 nov. 2021.

DONNELLY, R.; JOHNS, J. **Recontextualising remote working and its HRM in the digital economy**: An integrated framework for theory and practice, The International Journal of Human Resource Management, 2021, 32:1, 84-105, DOI: 10.1080/09585192.2020.1737834.

ELIAS, G.; LOBATO, L. C. **Arquitetura e Protocolos de Rede TCP-IP**, 2. ed. Rio de Janeiro: RNP/ESR, 2013. 414 p. 56-62.

DURUMERIC, Z.; KASTEN, J.; BAILEY, M.; HALDERMAN, J. A. **Analysis of the HTTPS Certificate Ecosystem**, Proc. of the 13th Internet Measurement Conference (IMC'13), Oct. 2013.

IETF (Internet Engineering Task Force). **The WebSocket Protocol,** Request for Comments: 6455, Category: Standards Track, ISSN: 2070-1721. Dezembro de 2011. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6455> Acesso em 22 nov. 2021.

SHIRKY, C. **What Is P2P… And What Isn't**, The O'Reilly Network, Novembro 2000. Disponível em: <http://anet.sourceforge.net/cached/p2p/13/472.html> Acesso em 19 nov. 2021.

IETF (Internet Engineering Task Force). **Interactive Connectivity Establishment (ICE):** A Protocol for Network Address Translator (NAT) Traversal**,** Request for Comments: 8445, Category: Standards Track, ISSN: 2070-1721. Julho de 2018. Disponível em: <https://datatracker.ietf.org/doc/html/rfc8445> Acesso em 19 nov. 2021.

NTWG (Network Working Group). **Session Traversal Utilities for NAT (STUN)**, Request for Comments: 5389, Category: Standards Track. Outubro de 2008. Disponível em: <https://datatracker.ietf.org/doc/html/rfc5389> Acesso em 19 nov. 2021.

SRISURESH, P.; FORD, B.; KEGEL, D. **State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)**, Network Working Group, Request for Comments: 5128, Category: Informational. Março de 2008. Disponível em: <https://www.ietf.org/rfc/rfc5128.txt> Acesso em 20 nov. 2021.

IETF (Internet Engineering Task Force). **Traversal Using Relays around NAT (TURN):** Relay Extensions to Session Traversal Utilities for NAT (STUN)**,** Request for Comments: 5766, Category: Standards Track, ISSN: 2070-1721. Abril de 2010. Disponível em: <https://datatracker.ietf.org/doc/html/rfc5766> Acesso em 19 nov. 2021.

NETCRAFT. **October 2021 Web Server Survey**, 2021. Disponível em: <https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html> Acesso em 23 nov. 2021.

PRAKASH, P.; BIJU, R.; KAMATH, M. **Performance analysis of process driven and event driven web servers**, *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, 2015, pp. 1-7, doi: 10.1109/ISCO.2015.7282230.

CHANIOTIS, I. K.; KYRIAKOU, KI.D.; TSELIKAS, N.D. **Is Node.js a viable option for building modern web applications? A performance evaluation study**. *Computing* **97,** 1023–1044 (2015). https://doi.org/10.1007/s00607-014-0394-9

KARADOGAN, G. M. **Evaluating WebSocket and WebRTC in the Context of a Mobile Internet of Things Gateway**, Master of Science Thesis, KTH Royal Institute of Technology Stockholm, Sweden. 2013.

W3C (The World Wide Web Consortium), **WebRTC 1.0: Real-Time Communication Between Browsers**, W3C Recommendation 26 January 2021. Disponível em: <https://www.w3.org/TR/webrtc/> Acesso em 20 nov. 2021.