Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informática – DAINF
Engenharia de Computação
Oficina de Integração 3 (EEX23) – S71 – 2020/1

Relatório Técnico

Wherebot - Um robô de mapeamento de salas e qualidade de sinal Wi-Fi

Amanda Schmidt de Lima – amandalima@alunos.utfpr.edu.br
Heron Gomes Fernandez – heron@alunos.utfpr.edu.br
Lucas da Silva Nolasco – nolascoslucas@gmail.com
Pedro Romano Splendore – splendore@alunos.utfpr.edu.br
Wagner Rodrigues Ulian Agostinho – wagner.rua@hotmail.com

Outubro de 2020

Resumo

Devido à dificuldade de prever o sinal WiFi, muitas empresas não conseguem adequar o sinal transmitido no local, fazendo com que os usuários tenham dificuldade de acesso à internet. Esse projeto propõe um equipamento para automatizar site surveys, ajudando a identificar a distribuição desequilibrada de sinal *wireless* e atender essa demanda crescente no ambiente corporativo. Esse relatório apresenta uma visão geral do projeto, explicando como foi utilizada cada tecnologia e como elas foram integradas para formar o *Wherebot*. Para a prova de conceito e entrega de resultados confiáveis, o robô foi simulado em diversos ambientes utilizando *ROS* e *Gazebo*. Esse projeto envolve um sistema de navegação, a infraestrutura de um servidor e uma página web como fonte de interação com o usuário.

1 Introdução

Nos últimos anos a disponibilização de conexão com a internet tem avançado drasticamente e já é possível vislumbrar um mundo onde tudo pode estar conectado [1]. E um dos meios imprescindíveis para esse acesso, são as conexões wireless [2] que estão presentes em praticamente todos os dispositivos disponíveis no mercado atualmente. Devido a isso a implementação de redes wireless em ambientes corporativos e domésticos se torna essencial nos dias de hoje e traz consigo alguns problemas, como por exemplo zonas mortas na rede [3], tais quais podem ser solucionados com a realização de site surveys na rede [4]. O *Wherebot* surgiu da compreensão dessa necessidade, automatizando a medição de sinais wireless e criando mapas de calor que facilitam a identificação de problemas na rede.

O *Wherebot* é um robô capaz de mapear um ambiente enquanto executa a medição da qualidade de sinais Wi-Fi, criando assim mapas de calor que facilitam a interpretação dos resultados obtidos.

Mais detalhes do funcionamento do robô são apresentados nas próximas seções.

A seção 2, a seguir, apresenta o funcionamento geral do projeto e os requisitos para o funcionamento desse. Na seção 3, são apresentadas as tecnologias utilizadas e a motivação da escolha dessas. Na seção 4, o desenvolvimento é colocado em foco, explicando cada parte do projeto, incluindo o robô, o servidor e a aplicação web. Por fim, a seção 5 mostra os resultados finais, desafios e conclusão.

2 Especificação do Projeto

O *Wherebot* pode ser dividido em três partes: o robô que faz o mapeamento e exploração do ambiente, o servidor que controla o funcionamento do robô e a página Web que é a interface de usuário para acessar tais funcionalidades. A Figura 1 apresenta a interação entre as partes, sendo que a *web page* representada na parte de baixo do diagrama roda localmente no robô, então pode ser considerada uma parte do mesmo.

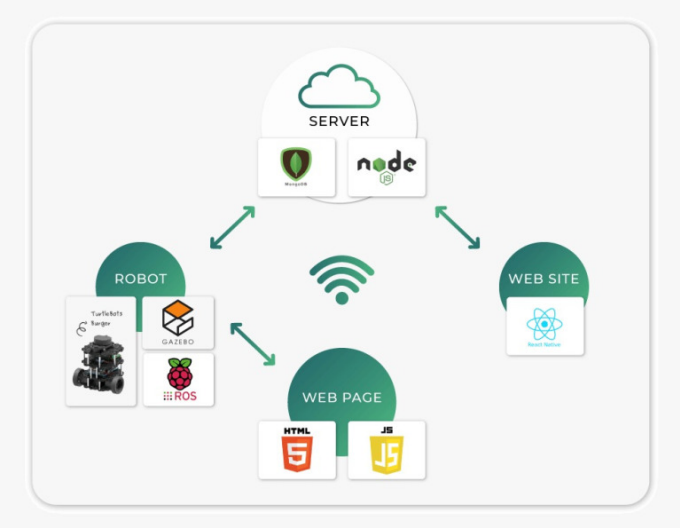

Figura 1: Diagrama simplificado do projeto

A utilização do *Wherebot* é dividida em duas etapas. Na primeira etapa, o usuário acessa a aplicação web e faz o próprio cadastro (caso não possua uma conta) e junto é realizado o cadastro do código de robô para definir tal usuário como dono. Na segunda etapa, o usuário acessa a rede Wi-Fi local do robô para acessar a página web responsável por configurar em qual rede Wi-Fi ele deve se

conectar, dessa forma ele é será capaz de enviar os resultados obtidos ao servidor. É interessante notar que as etapas são assíncronas e, após realizadas, o usuário poderá solicitar na aplicação web a análise do ambiente, é importante entretando lembrar que o ambiente que será mapeado deve estar totalmente fechado.

Inicialmente, o robô seria construido pela equipe utilizando um *Raspberry Pi 3 Model B* como sistema embarcado, um *Microsoft Kinect* de primeira geração para mapeamento do espaço e o *Robot Operating System* como ambiente de execução dos algoritmos a serem desenvolvidos. Toda estrutura mecânica do robô, assim como os atuadores, seriam definidos pela equipe.

Dada a situação sanitária atual, a qual uma pandemia mundial (o COVID-19) ainda não foi controlada, tornou-se inviável a reunião presencial da equipe para o desenvolvimento das partes mecânicas e integração dessas.

Assim, é importante notar que no desenvolvimento desse projeto todas as partes mecânicas e eletrônicas foram substituidas por execução de simulações que utilizam como modelo de robô o *Turtlebot3 Burger* que atende todas as demandas do projeto e faz com que todo o projeto possa ser repassado da simulação para o robô real com facilidade, tornando a execução do projeto ainda válida mesmo em ambiente simulado.

O servidor web recebe todas as requisições do robô e de usuário. São utilizados o *AWS*, para guardar as imagens de mapas de calor enviadas ao servidor e também todas as informações do banco de dados *MongoDB*, além disso o *NodeJS* foi utilizado como ferramenta principal de desenvolvimento, possibilitando que todo código fonte fosse escrito em *JavaScript*.

Os resultados obtidos da análise do Wherebot e o controle de suas ações são acessados através de uma aplicação web, que disponibiliza também um cadastro fácil de usuários. Um *website* é interessante para essa aplicação pois facilita a vizualização dos mapas de calorpelos usuários.

Por fim, a tabela 1 mostra os principais requisitos funcionais definidos do projeto.

Tabela 1: Requisitos funcionais do projeto

Robô e simulação	
RF01	O robô deverá ter um pacote responsável pela navegação autônoma.
RF02	O robô deverá ter um pacote ROS para criar um mapa 2D de suas proximidades utilizando SLAM.
RF03	O robô deverá ter um pacote ROS responsável pela criação de *heatmap* e comunicação com o servidor.
RF04	O robô deverá ter um adaptador WiFi adicional.
RF05	O robô deverá ter um script para a criação de uma rede Ad-Hoc.
RF06	O robô deverá ter um servidor web com uma página web a qual permite a configuração da rede WiFi que o robô se conectará para acessar o servidor.
RF07	A simulação do robô deverá ter um pacote ROS responsável pelo ambiente virtal e robô virtual.
RF08	O robô deverá ter uma chave única para registro de usuário.

Website	
RF01	O site deverá ter um mecanismo de registro de novo usuário.
RF02	O site deverá ter um mecanismo de autenticação.
RF03	O site deverá ter um campo onde o usuário possa conectar o robô à conta dele.
RF04	O site deverá ter um botão para requisitar o início de um site survey.
RF05	O site deve mostrar o heatmap quando o site survey terminar.
RF06	A aplicação deve mostrar uma lista de sinais WiFi, para que o usuário possa escolher qual sinal ele quer ver o heatmap.
Servidor e Banco de Dados	
RF02	O servidor web deve ter uma rota para registro de novos usuários.
RF03	O servidor web deve ter uma rota para autenticação de usuário.
RF04	O servidor web deve ter uma rota para conectar a chave de robô ao usuário.
RF05	O servidor web deve ter uma rota para requisitar o estado do robô.
RF07	O servidor web deve ter uma rota para requisitar heatmaps de um robô.
RF08	O servidor web deve ter uma rota para enviar heatmaps.
RF11	O banco de dados deve ter uma alocação de dados persistente para o perfil de usuário e para a informação de robô.

3 Tecnologias Utilizadas

Esta seção descreverá as tecnologias utilizadas e suas funções no projeto. A seção 3.1 apresenta os componentes do robô; A seção 3.2 apresenta a ferramenta utilizada nas simulações; A seção 3.3 apresenta a estrutura do servidor; por fim, a seção 3.4 apresenta as tecnologias da página web.

3.1 Robô

3.1.1 Robot Operating System (ROS)

O *Robot Operating System* [5] é uma estrutura flexível para desenvolver *software* para robôs. É uma coleção de ferramentas, bibliotecas e convenções que visam simplificar a tarefa de criar um comportamento robusto e complexo de robôs em uma ampla variedade de plataformas robóticas.

3.1.2 Turtlebot3 Burger

O *Turtlebot3* [6] é um robô móvel pequeno, acessível, programável e baseado em ROS para uso em educação, pesquisa, hobby e prototipagem de produto. A versão *Turtlebot3 Burger* foi escolhida para esse projeto, ela tem como componentes principais um *Raspberry Pi 3 B+*, uma placa embarcada *OpenCR*, dois atuadores XL430 e um sensor 360º de distância a laser (*LiDAR*).

3.2 Simulação

3.2.1 Gazebo

Gazebo [7] é um simulador de robótica 3D de código aberto, ele oferece a capacidade de simular com precisão e eficiência robôs em ambientes internos e externos complexos, utilizando mecanismos de física robustos.

3.3 Servidor

3.3.1 NodeJS

NodeJS [8] trata-se de uma aplicação de código aberto, *cross-platform*, e de um *runtime* de JavaScript que é capaz de executar um código de JavaScript a nível *backend* fazendo com que seja usado para desenvolvimento de servidores.

3.3.2 MongoDB

MongoDB [9] é um software de banco de dados orientado a documentos livre, de código aberto e multiplataforma. Ele é classificado como um programa de banco de dados *NoSQL*, o *MongoDB* usa documentos semelhantes a JSON com esquemas.

3.3.3 Amazon Web Services

Amazon Web Services [10], também conhecido como AWS, é uma plataforma de serviços de computação em nuvem, que formam uma plataforma de computação na nuvem oferecida pela *Amazon* [11].

3.4 Aplicação Web

3.4.1 React

Para criação de toda interface web a biblioteca *ReactJS* [12] foi utilizada. O projeto *React* consiste em uma biblioteca JavaScript de código aberto, mantida pelo Facebook, com foco em criação de interfaces de usuário.

3.4.2 Heroku

Heroku [13] é uma plataforma para hospedagem de aplicações na nuvem, ele possui suporte para diversas linguagens incluindo NodeJS e ReactJS. Oferece serviços de DNS e é facilmente escalável.

4 Desenvolvimento

4.1 Robô

4.1.1 Turtlebot

O Turtlebot foi escolhido para esse projeto por ser uma das plataformas de robôs mais populares no ramo da educação e da pesquisa. Além disso, possui firmware, software e hardware abertos, o que estimula e fomenta sua modificação. Ele foi desenvolvido também para ter um custo acessível, ser pequeno e com sensores precisos, o que vai de encontro ao mínimo esperado para a realização desse projeto. Quanto aos seus sensores, vale destacar a presença do LiDAR (*Light Detection And Ranging*), que é uma tecnologia que consiste em detectar a distância e/ou mais informações de algum objeto no espaço por meio da luz. Vemos o modelo escolhido na figura 2 e sua representação virtual no Gazebo na figura 3.

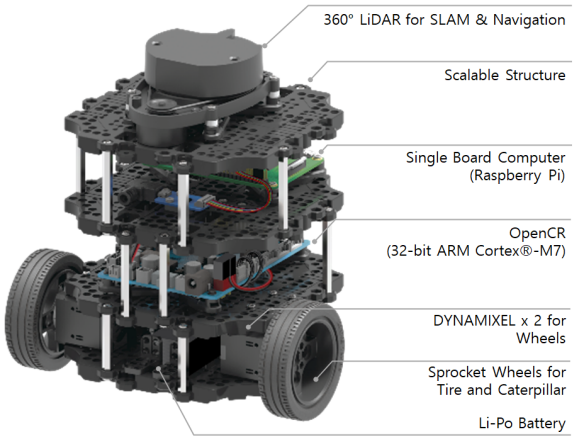

Figura 2: TurtleBot3 Burger

Figura 3: TurtleBot3 Burger simulado no ambiente Gazebo

4.1.2 ROS

Praticamente todo software do robô foi construído com base na estrutura de nós já fornecida pelo *ROS*. Um nó pode ser visto como um processo que executa algum código, e pode se comunicar com outros nós através, principalmente, de tópicos. Esses nós devem operar resolvendo diferentes tipos de problemas e no final a junção de todos os nós será responsável por todo sistema de controle do robô. A comunicação entre nós, como mencionado anteriormente, é geralmente feita por meio de tópicos, então quando um determinado nó publica algum tipo de informação em um tópico, todos os outros nós inscritos neste tópico recebem essa informação, a Figura 4 mostra de forma visual esse funcionamento.

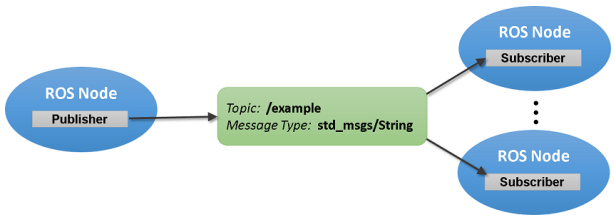

Figura 4: Representação da comunicação entre nós utilizando tópicos.

4.1.3 SLAM

O *SLAM*, ou então, *Simultaneous Localization and Mapping* é um problema que se caracteriza pela necessidade de localizar um robô enquanto se constrói um mapa do ambiente [14]. Isso não é uma tarefa simples visto que para um bom mapa é necessário uma boa estimativa sobre a localização do robô, enquanto para localizar o robô é necessário um bom mapa. Com o objetivo de resolver esse problema, vários algoritmos frutos de pesquisas nessa área foram propostos ao longo dos anos. Entre eles está o *OpenSLAM's Gmapping* [15]. Este algoritmo propõe utilizar o histórico de trajetória do robô em conjunto com os dados dos sensores (odômetro e *LiDAR*) para construir o mapa, melhorando os seus resultados ao levar em conta a evolução do sistema. Além disso, ele possui uma implementação em *ROS* sob o nome de *Gmapping*, o que permitiu que esse pacote fosse utilizado no desenvolvimento do *WhereBot* para a construção do mapa 2D do ambiente.

4.1.4 Navegação

A movimentação do robô no ambiente pode ser dividida em duas partes principais: a construção do mapa e a construção do *heatmap*. Para a primeira foi utilizado o pacote *explore_lite* [16] enquanto a segunda foi implementada pelo próprio grupo. Todas essas duas navegações operam sobre a *navigation stack*

[17] do *ROS* que é, de forma geral, um conjunto de pacotes que possuem como objetivo facilitar a navegação em robôs que utilizam esse sistema. Os pacotes utilizam os mapas que estão sendo construídos pelo algoritmo de *SLAM* para planejar um caminho seguro para o robô até um determinado ponto, permitindo a navegação com base em objetivos. Isso significa que para mover o robô para uma determinada posição no mapa basta indicar as suas coordenadas.

Primeira Exploração O pacote utilizado implementa uma movimentação para exploração de fronteiras baseado em uma estratégia gulosa. Para isso, utilizando a matriz de dados que representa o mapa que está sendo construído, o algoritmo verifica as células vizinhas ao robô procurando por pontos do mapa desconhecidos, ou seja, onde ainda não há dados sobre essa posição. Esse processo se repete avançando para as células vizinhas que conhecidamente não possuem obstáculos. Ao se encontrar uma célula de fronteira, procura-se células conectadas a esta primeira que possuam a mesma condição, formando assim uma nova fronteira. Um exemplo do resultado final encontrado para esse algoritmo pode ser visto na Figura 5. Nela, as fronteiras calculadas são marcadas em azul e os pontos verdes indicam o custo de cada uma dessas fronteiras, sendo as maiores fronteiras as que possuem o menor custo de acordo com a escolha gulosa utilizada. Com as fronteiras calculadas, escolhe-se a que possui o maior tamanho e, consequentemente, o menor custo.

Por fim, para definir o ponto para onde o robô deve se mover, o algoritmo adota a estratégia de calcular o centróide dos pontos que compõem a fronteira escolhida. Entretanto, tal estratégia pode resultar em um impasse no caso do centróide da fronteira ser a própria região onde o robô já está. Um exemplo claro dessa situação é o robô começando a exploração no meio de uma sala. Nesse caso, a fronteira inicial seria um círculo ao redor do robô e, consequentemente, o centróide dessa fronteira seria a posição do próprio robô. Para solucionar esse problema foi feita uma modificação no programa, fazendo o robô se mover para o ponto da fronteira mais próximo da sua posição atual quando for detectado que o robô já está sob o centróide. Caso o robô não consiga atingir esse objetivo calculado, esse ponto é adicionado em uma lista negra para evitar que o robô fique infinitamente tentando alcançar essa região sem sucesso. Esse processo é repetido continuamente até não encontrar fronteiras a se explorar.

Segunda Exploração Após a primeira exploração terminar, com o mapa já construído, é feita uma segunda movimentação visando visitar todo o ambiente, visto que a primeira exploração, dependendo do mapa, pode ser concluída sem se percorrer toda a área. Para definir o caminho a ser percorrido, o primeiro passo adotado é verificar a região dentro do ambiente que o robô deve explorar. As células adjacentes à posição inicial do robô são verificadas e então o processo é repetido sucessivamente para as células vizinhas das últimas células verificadas, todos os pontos que estão marcados no mapa como livre de obstáculos são

Figura 5: Fronteiras calculadas com pacote *explore_lite* em azul

registrados, gerando um novo mapa com todos os pontos alcançáveis pelo robô, como o mostrado na Figura 6. Para analisar se um determinado ponto é livre de obstáculo, foi utilizado o costmap [18], que pode ser resumido como uma versão do mapa original onde a área dos obstáculos é inflada, criando uma região de segurança e reduzindo o risco de colisões para o robô. Ele utiliza uma função de custo que varia entre 0 e 255 para representar o risco de uma colisão no mapa, sendo 255 o risco máximo, ou seja, o obstáculo de fato. Portanto, antes de utilizá-lo, o mapa foi binarizado considerando a faixa de valores livres consideradas pela documentação do *ROS*.

Figura 6: Comparação entre o costmap binarizado (esquerda) e o mapa com os pontos alcançáveis (direita)

O próximo passo é procurar por regiões não visitadas pelo robô nesse mapa de pontos alcançáveis. No caso, como para a simulação do WiFi é necessário antes ter o mapa inteiro pronto para calcular as atenuações por conta das paredes, essa verificação não causa efeitos, visto que toda a área do mapa ainda é considerada não visitada. No entanto, essa verificação permite que, no caso de um robô real, seja possível aproveitar a área já coberta pela primeira exploração, só visitando os pontos que ainda faltam. Para definir como essa região será percorrida, considerando os limites dessa área, são definidos grids no eixo

x e y com espaçamento de 12 células, o que equivale a 60 cm. Em seguida, todos os pontos onde as linhas do grid se cruzam são verificados e os pontos marcados como alcançáveis são adicionados na lista de objetivos do robô. A Figura 7 mostra um exemplo dessa estratégia sendo aplicada, onde as linhas em azul indicam os grids e os pontos em vermelho indicam os objetivos que farão parte do caminho percorrido pelo robô.

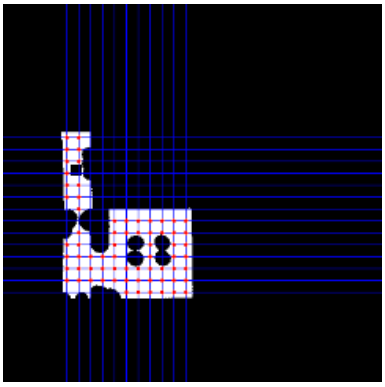

Figura 7: Pontos adicionados ao caminho a ser percorrido pelo robô

Finalmente, para criar um caminho a partir dessa lista de objetivos, os pontos são reordenados de forma que o próximo objetivo seja o ponto ainda disponível mais próximo do objetivo anterior. Por fim, o robô obtém esse caminho e executa cada um dos objetivos. Caso um dos pontos esteja inacessível por alguma mudança no ambiente e o robô não consiga alcançá-lo, o ponto é descartado e o robô segue para o próximo objetivo.

4.1.5 Simulação do Sinal Wi-Fi

O principal objetivo do projeto é a capacidade de criar mapas de calor de redes sem fio em ambientes fechados, então uma forma de simular esses sinais era essencial para que fosse possível validar o sistema. Para isso o modelo da *Free Space Path Loss* (FSPL) foi utilizado. Em telecomunicações a FSPL é a atenuação da energia de rádio entre os pontos de alimentação de duas antenas em uma linha de visão livre de obstáculos [19]. O FSPL depende de dois parâmetros: o primeiro é a frequência dos sinais de rádio; o segundo é a distância de transmissão sem fio. A fórmula a seguir [20] pode refletir a relação entre eles.

$$FSPL(dB) = 20log10(d) + 20log10(f) + K \quad (1)$$

Nessa fórmula, d é distância, f a frequencia e K uma costante que depende das unidades usadas para d e f. Se d é medido em metros e f em MHz a constante K tem o valor de -27.55. Utilizando essa fórmula juntamente com a fórumla de *fade margin*[21], conseguimos chegar a uma equação que liga a distância em metros com o nivel do sinal (FSPL) em dBm:

$$d(m) = 10^{\frac{FSPL+27.55-20log(f)}{20}} \quad (2)$$

Com essa equação, a posição do robô e a posição do roteador podemos facilmente calcular a força do sinal onde o robô está, mas como dito anteriormente essa fórmula não leva em consideração obstáculos, por isso um pouco de atenuação é adicionada quando paredes são encontradas entre a posição do robô e a posição do roteador. Dessa forma o robô é capaz de fazer uma simulação simplificada de redes sem fio.

4.1.6 Criação do Heatmap

O nó responsável pelo *heatmap* é responsável por duas partes principais: o tratamento de imagem recebida e acrescentar os dados de medições de Wi-Fi para a criação do *heatmap*. A imagem do ambiente explorado é recebida em formato PGM e os dados das medições são salvos em um arquivo CSV. O caminho de ambos é recebido por um tópico *ROS*.

O tratamento de imagem consiste inicialmente no formato em que os outros pacotes retornam a imagem, pois a resolução era reduzida e havia todo o espaço do ambiente de simulação que nem sempre é utilizado. Assim a imagem é cortada e redimensionada. Também, para exibir o resultado do *heatmap*, é necessário que a imagem seja transparente na parte interna do ambiente explorado. Todos os tratamentos de imagens foram feitos usando o pacote Python *Pillow* [22].

Como a criação de *heatmaps* já é um problema conhecido e amplamente difundido, partimos do uso de um *script Python* simples, o wifi-heatmap [23], para avaliar as entradas de dados necessárias e modificações para o problema proposto. Devido ao código ter 10 anos e ser baseada em uma versão legada do *Python*, foi necessário fazer mudanças fundamentais para possibilitar o funcionamento dele.

O pacote de acesso ao CSV usado originalmente no script foi descontinuado, assim foi susbstituido pelo pacote *Pandas* [24]. Nele foram feitas as transformações de matrizes para alimentar os outros pacotes. Foram usados os pacotes *Scipy* [25] e *Numpy* [26] para fazer interpolação com função linear dos dados de medição do Wi-Fi. O pacote *Matplotlib* [27] foi usado para a exebição do gráfico final do *heatmap*.

Por fim, o resultado final da imagem em PNG é publicado em um tópico *ROS* em que o nó de comunicação de servidor está inscrito.

4.1.7 Comunicação com o Servidor

A comunicação do robô com o servidor foi desenvolvida como sendo um dos nós do nosso sistema. As responsabilidades desse nó podem ser divididas em três:

1. Receber do servidor a informação de que o robô deve iniciar uma nova movimentação e avisar ao nó de exploração, por meio de uma publicação em um tópico, que ele deve iniciar.

2. Receber a resposta do nó de exploração que, por meio de inscrição em um tópico, que a exploração início e enviar essa informação ao servidor.

3. Receber do nó responsável por criar os mapas de calor, por meio de inscrição em um tópico, onde os mapas foram armazenados e enviar ao servidor as imagens e as informações desse último *survey*.

Para que a comunicação com o servidor possa ser realizada o *Wherebot* deve estar conectado a uma rede, então o robô também é responsável por rodar um pequeno servidor web responsável por disponibilizar ao usuário uma página de configuração, que pode ser vista na Figura **??**, onde ele pode passar os dados de uma rede sem fio para o robô se conectar.

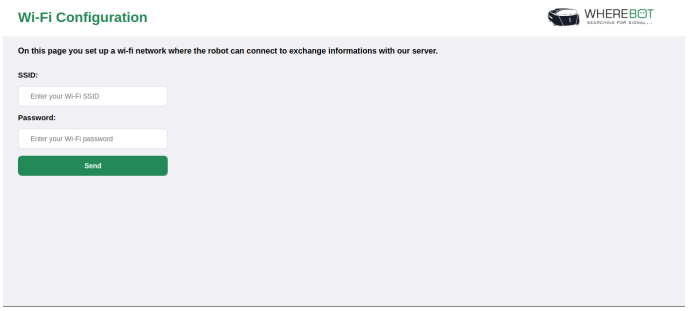

Figura 8: Página para configuração de uma rede sem fio.

4.2 Servidor

O servidor foi todo desenvolvido utilizando a linguagem de programação *JavaScript* e consequentemente o ambiente *NodeJS* que tem como principal função a execução de código *JavaScript* fora do navegador, ou seja, no *back-end*. Para dar início ao projeto a biblioteca *Express* para *NodeJS* foi utilizada, ela fornece um conjunto robusto de recursos para criação de todas funcionalidades necessárias em um servidor web.

4.2.1 Banco de Dados

Devido a sua grande versatilidade o banco de dados *NoSQL MongoDB* foi utilizado, a Figura 9 apresenta o diagrama de entidade e relacionamento do banco de dados gerado, esse banco foi criado utilizando o pacote *mongoose* disponível para *NodeJS* que permite a criação da estrutura e o manuseamento dos dados utilizando JavaScript.

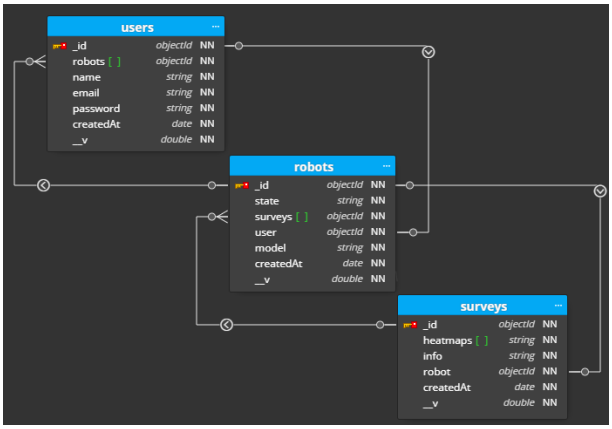

Figura 9: Pontos adicionados ao caminho a ser percorrido pelo robô

O banco de dados é composto de três tabelas, a tabela de *users* que guarda todos os dados referentes ao usuários cadastrados no banco e tem uma ligação de um para muitos com a tabela *robots* que guarda os dados de todos os robôs cadastrados no sistema e essa pro sua vez também tem uma ligação de um para muitos com a tabela de *surveys* que guarda as informações de todos os *site surveys* realizados.

4.2.2 Rotas

A arquitetura utilizada na criação do servidor é conhecida como Transferência Representacional de Estado (sigla em inglês REST), ou seja, nosso servidor é composto de várias rotas que podem ser acessadas por requisições HTTP, sendo que essas requisições devem ser dos tipos GET, POST, PUT ou DELETE.

No total nosso servidor é composto por nove rotas que são responsáveis por criar, ler, atualizar ou remover dados do nosso banco de dados, todas essas rotas e suas responsabilidades podem ser verificadas na Figura 10. Um ponto importante é que algumas dessas rotas são restritas, dessa forma os dados dos usuários não podem ser acessados facilmente.

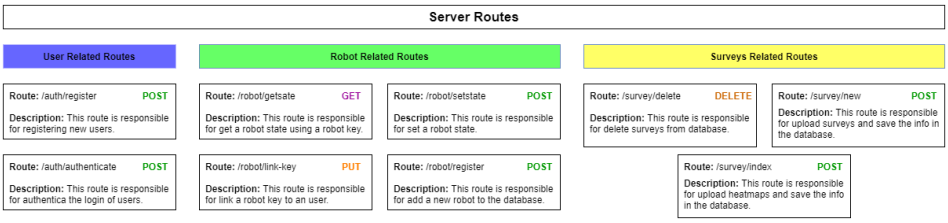

Figura 10: Pontos adicionados ao caminho a ser percorrido pelo robô

4.2.3 Segurança e Autenticação

Para manter a segurança do sistema o uso de um token JWT (JSON Web Token) foi implementado no servidor, JWT é um método padrão da indústria para realizar autenticação entre duas partes por meio de um token assinado que autentica uma requisição web. Esse token é um código em Base64 que armazena objetos JSON com os dados que permitem a autenticação da requisição. Uma vez que os dados enviados pelo cliente na tela de login tenham sido autenticados no servidor, este cria um token JWT assinado com um segredo interno e enviará este token de volta ao cliente. Provido com o token autenticado, o cliente possui acesso agora a rotas da aplicação que são restritas.

4.2.4 Hospedagem

Para que o servidor esteja facilmente acessível aos clientes ele deve ser hospedado na nuvem, para isso a hospedagem do servidor é dividida em duas partes. Primeiro o banco de dados *MongoDB* foi hospedado nos serviços web da *Amazon*, além disso as imagens de mapas de calor que são enviadas ao servidor também são guardadas nos serviços da Amazon. Já o servidor em sim foi hospedado na plataforma *Heroku* que é compatível com aplicações feitas utilizando *NodeJS* e contém uma ferramenta de *deploy* automático ligado ao *github*, ou seja, sempre que o código for alterado no *GitHub* o servidor é automaticamente atualizado.

4.3 Aplicação Web

Antes de iniciar o desenvolvimento da página web, que é a principal fonte de interação com o usuário, vários passos foram realizados. Primeiramente um esboço de fluxo de navegação e um esboço das telas foi construído, dessa forma foi possível concluir quais dados eram necessários em cada tela e quais desses dados seriam mostrados ao usuário. Após isso foi possível a criação de uma versão final do fluxo de navegação assim como de um design para cada uma das telas presentes no fluxo, ambos podem ser vistos na Figura 11.

Com as rotas do servidor já configuradas, o fluxo de navegação finalizado e o design do projeto feito o desenvolvimento da aplicação web pode ser iniciado. As telas de login e registro basicamente fazem requisições para o servidor tanto para autenticar, no caso do login, quanto para cadastrar um novo usuário, no caso do registro. A página principal tem um funcionamento um pouco mais complicado, ela é responsável por fazer as seguintes requisições ao servidor:

- Requisitar os dados de todos robôs e *surveys* associados a um usuário.
- Requisitar o início de um *survey* a um dos robôs pertencentes ao usuário.
- Requisitar a associação de um novo robô ao usuário.

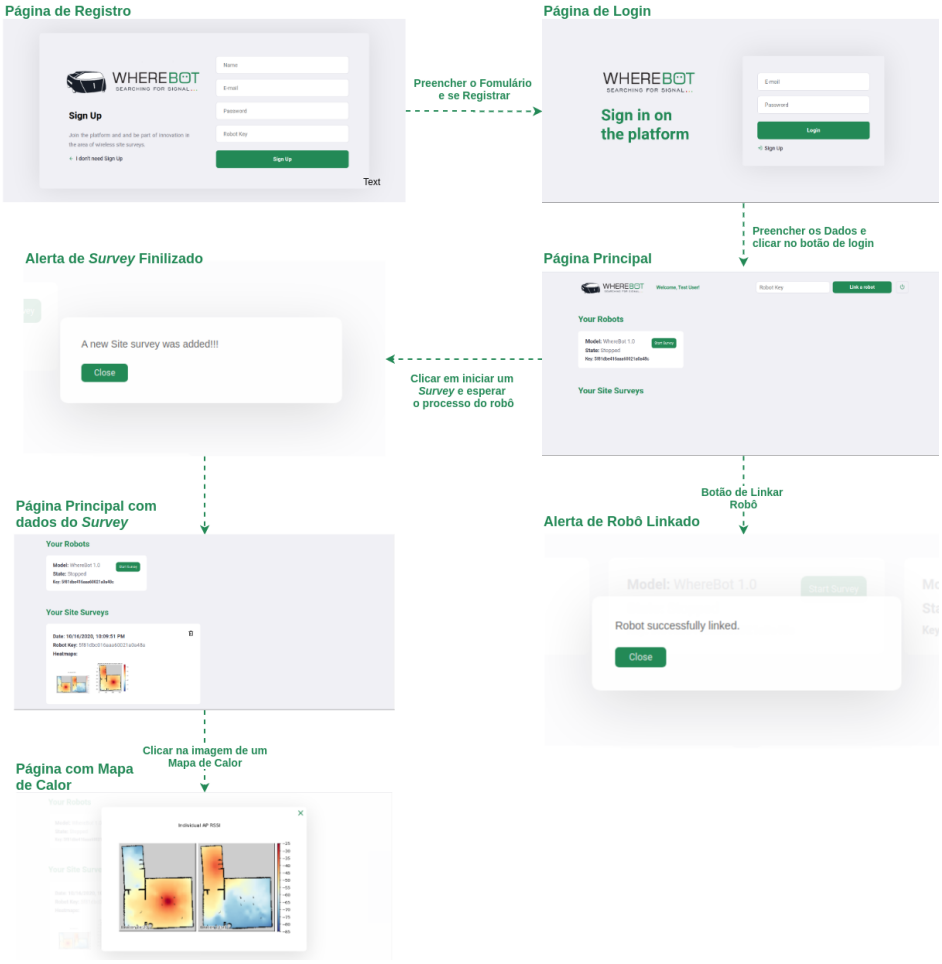

Figura 11: Fluxo de navegação da página web.

Além disso ela deve receber uma notificação quando um *survey* é finalizado, para isso utilizamos um cliente *socket* que se conecta ao servidor assim que o usuário acessa sua página principal, com essa conexão assim que o servidor registra um novo *survey* uma notificação é enviada através do *socket* e uma notificação é mostrada na tela.

5 Resultados e Conclusões

5.1 Testes e Resultados

Para que o funcionamento básico do sistema pudesse ser comprovado, foram feitos os seguintes testes:

- Um novo usuário de testes foi registrado e executou o login na aplicação web.
- O usuário de testes associou um wherebot a sua conta.
- O usuário de testes inicio a navegação do wherebot utilizando a aplicação web e recebeu o resultado do site survey.
- O usuário de testes exluiu as infomações do survey realizado.
- O modelo simulado do robô realizou o mapeamento de três ambientes diferentes, enquanto os dados de rede sem fio eram simulados.
- O modelo simulado do robô realizou o mapeamento de um ambiente enquanto desviava de alguns obstáculos adicionados durante a execução.

5.2 Problemas e Desafios

Durante o desenvolvimento do projeto foram encontradas algumas dificuldades. A primeiro delas foi a falta de experiência com o *ROS* e também com simulações, o que trouxe a necessidade de aprender diversos novos conceitos para a implementação do projeto. Além disso, houve dificuldade em se definir como o problema da segunda navegação, a responsável por cobrir a área do ambiente, seria abordado. Não foram encontrados muitos pacotes na comunidade que implementassem um algoritmo para esse problema, e os encontrados estavam há muito tempo sem suporte e sequer funcionaram nos testes realizados. Por fim, quando optou-se fazer uma implementação própria, surgiu a questão de como explorar uma região de forma estruturada. Em um primeiro momento foram testados o ponto dessa região mais distante do robô, o mais próximo e até o centróide da região. No entanto, isso criava uma exploração desordenada que, apesar de cumprir o objetivo, precisava de muito tempo para se completar.

Outros dois problemas encontrados estão associados à primeira exploração do ambiente. Inicialmente, quando o robô era colocado em uma posição u objetos ao seu redor, a estratégia adotada pelo pacote utilizado calculava como objetivo a própria posição do robô, o levando a ficar parado. Isso trouxe a necessidade de se alterar o código fonte desse pacote criando uma estratégia alternativa para esse caso. O segundo problema ocorre pois quando o robô é colocado na mesma situação anterior, ele passa a depender unicamente do uso de odometria uma vez que não há objetos dentro do limite do *LiDAR*, fazendo com que

o mapa gerado não esteja correto ou, em casos mais graves, a exploração tenha problemas para ser finalizada.

5.3 Conclusão

O desenvolvimento do projeto trouxe a necessidade da integração de várias áreas de conhecimento adquiridas durante o curso, tanto na parte de software como de hardware, na parte de desenvolvimento de aplicação para o embarcado, o desenvolvimento do servidor, o desenvolvimento da aplicação web, além da integração entre todos esses componentes. Além disso o desenvolvimento proporcionou o aprendizado de novas ferramentas e tecnologias, como o ROS, o Gazebo e o sistema de aplicações em nuvem da Amazon. Por mais que alguns problemas tenham sido encontrados durante o decorrer do desenvolvimento, eles foram solucionados graças ao trabalho em equipe e ao planejamento inicial que levou em consideração essa possibilidade. Com isso viu-se a real necessidade de um cronograma com horas de folga e uma análise de riscos com alternativas para contornar os problemas sem comprometer o tempo de entrega e o orçamento do projeto. Houve a necessidade de adaptar o cronograma inicial para um período mais curto e, por isso, a equipe replanejou todas as atividades necessárias. O tempo total previsto no início do projeto pelo cronograma, incluindo a folga de 30%, era de 366 horas, e o tempo total realmente gasto foi de 352 horas.

Referências

[1] Lu Tan and Neng Wang. Future internet: The internet of things. 2010.

[2] B. Dionísio and W. Daniel. Análise comparativa entre rede cabeada e rede wireless. 2015.

[3] Wirelessdead spots - what are they and how to combat them `https://www.digitalairwireless.com/articles/blog/wireless-dead-spots-what-are-they-and-how-combat-them`.

[4] Why is a wireless site survey needed? `https://www.sonicwall.com/support/knowledge-base/why-is-a-wireless-site-survey-needed/170503717715599/`.

[5] ROS. `https://www.ros.org`.

[6] Turtlebot3 Burger. `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`.

[7] NodeJS. `https://nodejs.org/en/`.

[8] Gazebo. `http://gazebosim.org`.

[9] MongoDB. https://www.mongodb.com.

[10] Amazon Web Services. https://aws.amazon.com/pt/.

[11] Amazon. https://www.amazon.com.

[12] ReactJS. https://pt-br.reactjs.org.

[13] HEROKU. https://www.heroku.com/.

[14] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The SLAM problem: A survey. *Frontiers in Artificial Intelligence and Applications*, 184(1):363–371, 2008.

[15] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[16] Jiri Horner. https://wiki.ros.org/explore_lite.

[17] Eitan Marder-Eppstein. https://wiki.ros.org/navigation.

[18] Eitan Marder-Eppstein, David V. Lu, Dave Hershberger. https://wiki.ros.org/costmap_2d.

[19] Mohammad Rafiqul Islam, Syad Kamrul; Haider. Sensors and low power signal processing, (2010 ed.).p.49.

[20] Walter Debus. Rf path loss transmission distance calculations, 2006. Download: https://www.researchgate.net/profile/Sourabh_Bharti3/post/How_can_I_find_sensing_range_of_sensor_node_based_on_its_energy/attachment/59d63e39c49f478072ea8f3e/AS%3A273767446581257%401442282653853/download/path-loss-calculations.pdf.

[21] O. Katircioğlu, H. Isel, O. Ceylan, F. Taraktas, and H. B. Yagci. Comparing ray tracing, free space path loss and logarithmic distance path loss models in success of indoor localization with rssi. 2011.

[22] Pillow. https://python-pillow.org/.

[23] Beau Gunderson. Pacote wifi-heatmap. https://github.com/abntex/limarka.

[24] Pandas. https://pandas.pydata.org/.

[25] Scipy. https://www.scipy.org/.

[26] Numpy. https://numpy.org/.

[27] Matplotlib. https://matplotlib.org/.