TRABALHO DE GRADUAÇÃO

IMPLEMENTAÇÃO DE REDE AD HOC DE ROBÔS PARA USO INTEGRADO AO SISTEMA OPERACIONAL DE ROBÔS (ROS)

Jacqueline Cristina Lima de Souza

Brasília, dezembro de 2018

UNIVERSIDADE DE BRASÍLIA

Faculdade de Tecnologia
Curso de Graduação em Engenharia Elétrica

TRABALHO DE GRADUAÇÃO

IMPLEMENTAÇÃO DE REDE AD HOC DE ROBÔS PARA USO INTEGRADO AO SISTEMA OPERACIONAL DE ROBÔS (ROS)

Jacqueline Cristina Lima de Souza

Relatório submetido como requisito parcial de obtenção de grau de Engenheira Eletricista

Banca Examinadora

Professor Geovany Araújo Borges, ENE/UnB _____
Orientador

Professor Marcelo Menezes de Carvalho, ENE/UnB _____
Coorientador

Professor Adolfo Bauchspiess, ENE/UnB _____
Examinador interno

Mestrando Rodrigo Werberich, ENE/UnB _____
Examinador interno

Brasília, dezembro de 2018

FICHA CATALOGRÁFICA

DE SOUZA, JACQUELINE CRISTINA LIMA

Implementação de Rede *Ad Hoc* de robôs para uso integrado ao

Sistema Operacional de Robôs (ROS), [Distrito Federal] 2018.

xiii, 47p., 297 mm (FT/UnB, Engenheira, Elétrica, 2018). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. MANET 2. Robótica Móvel

3. ROS

I. Elétrica/FT/UnB II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

DE SOUZA, JACQUELINE CRISTINA LIMA, (2018) Implementação de Redes Ad Hoc de robôs para uso integrado ao Sistema Operacional de Robôs (ROS). Trabalho de Graduação em Engenharia Elétrica, Publicação FT. TG-$n^{\circ **}$, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 47p.

CESSÃO DE DIREITOS

AUTOR: Jacqueline Cristina Lima de Souza

TÍTULO DO TRABALHO DE GRADUAÇÃO: Implementação de Rede *Ad Hoc* de robôs para uso integrado ao Sistema Operacional de Robôs (ROS)

GRAU: Engenheiro ANO: 2018

Jacqueline Cristina Lima de Souza

Campus Darcy Ribeiro, SG-11, Universidade de Brasília.

70919-970 Brasília – DF – Brasil.

Dedicatória

Aos meus pais, irmãs, namorado e amigos.

Jacqueline Cristina Lima de Souza

Agradecimentos

Primeiramente gostaria de agradecer a DEUS que sempre esteve ao meu lado, me fortalecendo e não me deixando desistir. Obrigada Pai!

Gostaria de agradecer aos meus professores orientadores Geovany Araújo Borges e Marcelo Menezes de Carvalho. Me ensinaram e me ajudaram a trilhar esse caminho. O professor Geovany que me orientou e me ensinou em PIBICs também.

Agradeço meus pais, Odilon e Simone, por cuidarem de mim e se preocuparem comigo nessa caminhada que não foi nada fácil. Por sempre estarem ao meu lado mesmo nos dias mais difíceis e comemorarem comigo cada passo que deu certo. Sei que precisou de paciência para que eu alcançasse esse momento, mas nunca deixaram de acreditar em mim.

Minhas irmãs Sandy e Jéssyca que sempre se preocuparam com o melhor para mim, buscando me aconselhar e me mostrar o melhor caminho. Minha irmã Jéssyca, um exemplo de força e coragem que me acalmou e me aconselhou. Uma voz que busco ouvir nos momentos de aflição. Me recebeu em sua casa e fez dos meus dias melhores.

Meu namorado Herman que sempre esteve ao meu lado para me abraçar e afirmar que tudo daria certo. Esteve presente para ouvir cada medo, dúvida e para secar as lágrimas nos dias tristes. Comemorou comigo cada passo de sucesso. Me aguentou nos dias complicados em que as provas e as noites não dormidas me enlouqueciam. Entrou nesse curso comigo e saiu comigo sendo muito forte e aguentando muita pressão. Muito obrigada meu amor!

Agradeço também aos meus amigos que fizeram a diferença no curso: Antonio, Bruno, Guilherme, Jéssica, Jessica, Kássia, Lúcio, Lukas, Túlio. Foi um prazer conhecer cada um, estiveram junto comigo na maior parte desses difíceis anos. Me ajudaram a vencer muitas matérias, torceram por mim e me abraçaram também nos dias difíceis. A Jéssica (b1) que estava sempre disposta a ajudar, tirar dúvidas. A Jessica (b2) que sempre atendia meus telefonemas para que eu pudesse perturbá-la e passou pelas longas aulas de OI comigo, fazendo desses momentos mais significativos. Sem contar o laboratório de máquinas elétricas, longas tarde que a b2 amou passar comigo. O Lúcio com seus abraços consoladores. Agradeço a todos e não tenho dúvida de que cada um foi fundamental para que eu conseguisse me formar.

Quero agradecer a Priscilla e o Arthur que me apoiaram e me deram forças.

Também quero agradecer ao meu irmão Kenai, pronto para me ouvir e ajudar em todos os momentos. Me consolou e me encheu de esperanças, orou por mim e me fortaleceu. "Existe amigo mais apegado que um irmão" Provérbios 18:24.

Sou muito grata também ao Gabriel Araújo, do AMORA, que sempre me ajudou quando precisei, em cada PIBIC foi um professor. Me ajudou também em meu TCC, obrigada cara.

Jacqueline Cristina Lima de Souza

RESUMO

Este trabalho apresenta a implementação de uma rede ad hoc móvel de comunicação sem fio entre robôs do Laboratório de Robótica e Automação (LARA), para o desenvolvimento de aplicações, execução de tarefas e comunicação de alto nível entre os robôs, utilizando o sistema operacional de robôs (Robot Operating System – ROS). A rede escolhida para tal foi a *Ad Hoc* móvel, por ser descentralizada e independente de forma a facilitar a troca de informação entre as máquinas e oferecer flexibilidade. O objetivo é que o sistema operacional de robôs (ROS), funcione utilizando a rede ad hoc para se comunicar com outros robôs que estejam na rede, também trabalhando com o sistema operacional de robôs. Além disso, que consigam interferir nas tarefas desenvolvidas pelo outro. Para alcançar tais objetivos o trabalho foi dividido de forma a primeiro trabalhar na implementação e configuração da rede *Ad Hoc* e depois na parte de controle, no ROS, para descentralizar a sua forma de atuação e criar a informação a ser compartilhada. A rede ad hoc foi implementada utilizando o protocolo de roteamento OLSR (*Optimized Link State Routing*) e a integração do sistema de operação de robôs (ROS).

Palavras Chave: robótica móvel, redes ad hoc móveis, comunicação entre robôs, ros.

ABSTRACT

This work presents the implementation of a mobile ad hoc network of wireless communication between robots of the Robotics and Automation Laboratory (LARA), for application development, task execution and high-level communication among robots, using the robot operating system (Robot Operating System – ROS). The network chosen for this was mobile Ad Hoc, because it was decentralized and independent in order to facilitate the exchange of information between the machines and offer flexibility. The goal is for the robot operating system to work using the ad hoc network to communicate with other robots that are on the network, working with the robot operating system (ROS). Furthermore, that can interfere with the tasks performed by the other. In order to reach these objectives the work was divided in order to first work on the implementation and configuration of the Ad Hoc network and then the control part in the ROS to decentralize its way of acting and create the information to be shared. The ad hoc network was implemented using the OLSR routing protocol (Opti mized Link State Routing) and the integration of the robot operating system (ROS).

Keywords: mobile robotics, Mobile Ad Hoc networks, comunication between robots, ros.

SUMÁRIO

1 Introdução 5

1.1 Contextualização 5

1.2 Definição do problema 7

1.3 Objetivos do projeto 7

1.4 Resultados obtidos 7

1.5 Apresentação do manuscrito 8

2 Fundamentação 9

2.1 Introdução 9

2.2 TCP/IP (Transmission Control Protocol / Internet Protocol) 10

2.3 Mobile Ad Hoc Network (MANET) 11

2.3.1 Conceito 12

2.3.2 Características 13

2.3.3 Protocolo OLSR (*Optimized Link State Routing*) 14

2.4 Robot Operating System (ROS) 17

2.4.1 Distribuições 18

2.4.2 Pacotes 19

2.4.3 Definições e Aplicações 20

2.4.4 Sistema com mestres ROS único 23

2.4.5 Sistema com múltiplos mestres ROS 24

2.5 Interface de voz para robôs 27

2.5.1 *Hardware* 27

2.5.2 *Software* 27

2.5.3 Máquina de Estados Finitos 28

3 Desenvolvimento 30

3.1 Introdução 30

3.2 Robô e Computadores 30

3.3 Escolha do sistema operacional 31

3.4 Implementação da rede Ad Hoc móvel (MANET) 32

3.5 Teste de funcionamento da rede Ad Hoc móvel (MANET) 34

3.6 Implementação de multi-mestres ROS 37

3.7 Implementação da Interface de Voz para Robôs 37

4 Resultados 40

4.1 Introdução 40

4.2 Funcionamento da interconexão 40

Conclusões 45

Proposta de trabalhos futuros 45

FONTES 46

REFERÊNCIAS BIBLIOGRÁFICAS 47

LISTA DE FIGURAS

1.0 Estado da Califórnia, Estados Unidos, autoriza teste de carros autônomos (2018). Fonte[1].....6

2.1 Pilha de protocolos TCP/IP, cada camada representa um pacote de funções. Fonte [6].....10

2.2. Esquemático que representa uma rede com ponto central controlador. Fonte[2].....12

2.3 Esquemático que representa uma rede Ad Hoc. Fonte[3].....13

2.4 Esquemático que representa as vizinhanças dos dispositivos em uma rede ad hoc. Fonte[4].....13

2.6 Rede sendo inundada, sem a utilização da otimização MPR. Fonte[5].....16

2.7 Esquemático mostrando o *backbone* do MPR. Fonte[5].....16

2.8 O nó da câmera anuncia que tem a mensagem do tipo imagem. Fonte[7].....21

2.9 O nó da câmera publica no tópico, chamado de imagens. O nó de visualização de imagens anuncia que quer assinar o tópico imagens. Fonte[7].....22

2.10 O mestre leva o nó de visualização de imagens a assinar o tópico imagens. Fonte[7].....22

2.11 Sistema com mestre único. Fonte[8].....24

2.12 Sistema com múltiplos mestres. Fonte[8].....25

2.13 Sistema com múltiplos mestres ROS, cada computador com um mestre.....26

2.14 Máquina de estados desenvolvida a fim de determinar o modo de resposta do computador.....28

3.1 Aramis, robô móvel da família Pioneer modelo P3-DX da Mobile Robots. Fonte[9].....32

3.2 Página de criação e configuração da rede no Ubuntu 16.04.....33

3.3 Rede em funcionamento em um dos computadores.....33

3.4 Posicionamento físico da rede ad hoc.....34

3.5 Estação C do esquemático mostrado na figura 3.4.....35

3.6 Estação G do esquemático mostrado na figura 3.4.....36

3.7 Estação E do esquemático mostrado na figura 3.4.....36

3.8 Esquemático mostrando nós e tópicos referentes a interface de voz para os robôs e suas associações.....38

3.9 Esquemático destacando o nó *ear*, o tópico que assina e o tópico em que publica.....38

3.10 Esquemático destacando o nó *fsm*, os tópicos os quais ele assina e o tópico em que publica.....38

3.11 Esquemático destacando o nó *talker*, o tópico que assina e o tópico em que publica.....39

4.1 Computador Aramis detectando os mestres dos outros que foram nomeado de Jacqueline e

Asimo.....41

4.2 Computador Asimo detectando os mestres dos outros que foram nomeado de Jacqueline e Aramis.....41

4.3 Computador Jacqueline detectando os mestres dos outros que foram nomeado de Aramis e Asimo.....41

4.4 Terminal Asimo publicando no tópico da interface de voz que está em andamento no computador Jacqueline.....42

4.5 Terminal Aramis publicando no tópico da interface de voz que está em andamento no computador Jacqueline.....42

4.6 Imagem mostrando a comunicação entre os mestres de cada computador.....43

4.7 Imagem mostrando o terminal do computador Jacqueline respondendo aos comandos tanto da máquina Jacqueline quanto do Aramis e Asimo.....43

4.8 Nós, tópicos e serviços do sistema de forma completa.....43

4.9 Mostra um outro computador fornecendo uma entrada para a interface de voz.....44

Siglas

FSM	Finite State Machine
IP	Internet Protocol
ISO	International Organization for Standardization
MANET	Mobile Ad Hoc Network
MPR	MultiPoint Relay
OLSR	Optimized Link State Routing
OSI	Open Systems Interconnection
ROS	Robot Operating System
SSH	Secure Shell
TCP	Transmission Control Protocol

Capítulo 1

Introdução

1.1 Contextualização

Cada vez mais se tem buscado automatizar os robôs e deixá-los mais inteligentes e independentes com o objetivo de aprimorar os serviços prestados a humanos. E ao analisar o desenvolvimento humano percebeu-se que o trabalho em equipe acelera e facilita a execução de muitas tarefas, por isso cada vez mais implementações de sistemas com interconexão de máquinas têm surgido.

O primeiro passo para se obter sistemas onde as máquinas trabalham em equipe é desenvolvendo a comunicação entre elas, sem isso a máquina desconhece o que as outras a sua volta estão executando, podendo interferir de forma destrutiva na tarefa. Além disso, através da comunicação as máquinas podem pedir ajuda na realização de algum trabalho, que necessite de mais de uma mão de obra ou que precise ser executada mais rápida. Por isso, este trabalho aborda a comunicação entre robôs para viabilizar a execução de tarefas em equipe.

Várias aplicações já existem e estão sendo colocadas em prática e um ótimo exemplo são os carros autônomos que têm cada vez mais avançado para se tornar uma realidade nas ruas dos países desenvolvidos. O carro é agora uma formidável plataforma de sensor, absorvendo informação do meio (e de outros carros) e o equipando com motorista e infraestrutura para auxiliar a navegação segura, controle de poluição e gerenciamento de tráfego [Referência 1].

No estado da Califórnia, Estados Unidos, foi permitido o teste de carros autônomos implementados pela *Google* e testado apenas em ambientes controlados. Agora querem observar se quando colocado nas ruas em contato com todas as variáveis que o ser humano pode proporcionar no tráfego, age de forma eficiente e segura. O resultado esperado é mostrar que o carro autônomo consegue tomar decisões melhores e mais rápidas que as pessoas, por possuir mais informações através da sua conexão com os outros carros e de sua ampla tecnologia de sensores, tornando assim o trânsito mais seguro, diminuindo congestionamentos e a poluição. A figura 1.1 mostra os carros iniciando seus testes nas ruas da Califórnia.

Figura 1.1: Estado da Califórnia, Estados Unidos, autoriza teste de carros autônomos. (2018) - Fonte[1]

É extremamente necessário que os sistemas embarcados dos carros entejam interconectados, a fim de fornecer os resultados esperados. Pois sistemas isolados não contribuem para que o trânsito flua e nem para a segurança, por não haver compartilhamento de informações.

Visando tudo isso, a implementação de uma interconexão entre robôs se torna primordial para sua inteligência e traz vários desafios no âmbito do modo de desenvolvimento, pois o modo de desenvolvimento define a eficiência e a segurança da rede. A segurança é colocada não apenas no sentido de invasão por *hackers*, mas também no sentido de depender de fontes para seu devido funcionamento. As escolhas feitas para construir a base da comunicação entre as máquinas são de extrema importância, devendo ser construída de acordo com a aplicação.

A interconexão é implementada neste para viabilizar o trabalho em equipe de robôs móveis na execução de tarefas dentro do laboratório de robótica e automação (LARA), mais especificamente foi desenvolvido para serem guias do laboratório. E para validar a comunicação estabelecida entre as máquinas foi implementado um sistema de interface de voz para se obter um diálogo entre o robô e as pessoas, a fim de fornecer explicações a respeito dos projetos desenvolvidos no LARA (Laboratório de Robótica e Automação).

Apesar da aplicação de interface de voz para robôs que será mostrada, a interconexão entre os robôs abre um leque para várias outras aplicações por ter características que são primordialmente: a independência, a flexibilidade e a segurança. Essas características são devido a escolha da rede ad hoc, que não necessita de um ponto de acesso central, sendo cada robô um ponto de acesso da rede, uma vez conectados a ela. E também devido a escolha do protocolo de roteamento escolhido para a MANET (*Mobile Ad Hoc Network*), o OLSR (*Optimized Link State Routing*) e do ROS (Robot Operating System).

1.2 Definição do problema

Neste trabalho é abordado o problema de implementar um MANET (*Mobile Ad Hoc Network*) para o uso integrado ao ROS (*Robot Operating System*). De forma a deixar a rede de comunicação e suas aplicações as mais independentes possível. Isso foi proposto porque se os robôs então conectados à rede convencional, possui um ponto de acesso central, trocando dados através dela e se ocorre alguma instabilidade na rede, a interconexão entre os robôs é perdida. Com esta finalidade foi proposto utilizar a rede ad hoc, para descentralizar a interconexão e trazer mais flexibilidade para a rede.

O ROS (*Robot Operating System*) trabalha de forma centralizada por possuir um mestre e os escravos. Pensando do ponto de vista que o sistema como um todo só funciona enquanto o mestre estiver presente. Então é proposto que ao invés de o sistema como um todo possuir apenas um mestre, possuir vários.

O desenvolvimento foi feito utilizando dois computadores, atuando como robôs, e um robô móvel, visto que qualquer robô que tenha um computador embarcado com o sistema operacional Ubuntu pode adotar o sistema aqui desenvolvido.

1.3 Objetivos do projeto

O objetivo final é montar uma infraestrutura para uma rede de robôs autônoma, que poderá ser utilizada para alguma aplicação específica. De forma que o ROS (*Robot Operating System*) utilize o MANET (*Mobile Ad Hoc Network*) para estabelecer uma interconexão entre os mestres ROS (*Robot Operating System*) de outros dispositivos. Conseguindo então mapear todos os sistemas ROS (*Robot Operating System*) em funcionamento que participe da MANET (*Mobile Ad Hoc Network*). E por fim, conseguir ter acesso a todos os processos em funcionamento dos mestres vizinhos e interferir neles.

Dentre os objetivos está demonstrar o resultado por meio da implementação de uma interface de voz para robôs, utilizando o ROS (*Robot Operating System*) a fim de validar e demonstrar a interconexão estabelecida.

1.4 Resultados obtidos

Neste trabalho foi implementado uma rede ad hoc móvel (MANET, do inglês *Mobile Ad Hoc Network*) e instalado o protocolo de roteamento OLSR (*Optimized Link State Routing*) para atuar nesta rede. O protocolo de roteamento OLSR calcula os possíveis caminhos para comunicação

com os robôs da rede, fornecendo instantaneamente quais estão em seu raio de alcance e quais os melhores percursos para o envio de mensagens.

Foi também implementado a interconexão entre três computadores, atuando como robôs, via MANET (*Mobile Ad Hoc Network*) e a interconexão entre o sistema operacional de robôs (ROS, do inglês *Robot Operantig System*) de cada máquina. O ROS utilizou o MANET como rede comum para conseguir localizar os outros robôs, atuando com ROS, e dessa forma mapeou os mestres de cada robô. Após adicionar o mestre a sua rede de mapeamento o próximo passo foi fazer um acesso remoto.

Além disso, foi trabalhado o ROS para mudar a forma como originalmente ele trabalha, de forma centralizada com um mestre. Para isso foi necessário fazer com que o sistema tivesse vários mestres ao invés de apenas um, onde cada robô tinha seu próprio mestre.

Por fim foi implementada uma interface de voz para robôs em apenas uma das máquinas para testar e validar todo o sistema desenvolvido.

1.5 Apresentação do manuscrito

Este trabalho está organizado em cinco capítulos. O Capítulo 2 apresenta a fundamentação teórica para a compreensão dos conceitos básicos deste trabalho. Ele é dividido em quatro seções, a primeira explica a arquitetura do modelo TCP/IP (*Transmission Control Protocol / Internet Protocol*), a segunda à rede Ad Hoc, seu funcionamento, as ferramentas utilizadas para criar a rede e configurá-la. A terceira discorre sobre ROS e apresenta seu conceito, a distribuição escolhida, sua funcionalidade, pacotes utilizados. O capítulo 3 apresenta a integração de rede ad hoc de robôs ao ROS (*Robot Operating System*) por etapas e o funcionamento de cada uma das suas partes. No Capítulo 4 estão contidos os resultados da implementação, mostrando primeiramente a rede Ad Hoc funcionando e comprovando o seu funcionamento através de testes. E em seguida a integração funcionando juntamente com o ROS e seus resultados. E por fim, o Capítulo 5 contém as conclusões deste trabalho, assim como algumas propostas de trabalhos futuros.

Capítulo 2

Fundamentação

2.1 Introdução

Este capítulo visa apresentar os conceitos e bases teóricas utilizados e necessários para a implementação deste trabalho. Dessa forma, foi dividido em quatro seções abordando os seguintes temas: arquitetura do modelo TCP/IP (*Transmission Control Protocol* / *Internet Protocol*), rede ad hoc móvel (MANET) e protocolo de roteamento OLSR (*Optimized Link State Routing*), ROS (*Robot Operating System*) e interface de voz para robôs.

A primeira seção apresenta a arquitetura do modelo TCP/IP (*Transmission Control Protocol* / *Internet Protocol*), suas camadas e atuação. O TCP/IP é uma pilha de protocolos que será introduzida e é de extrema importância para entender o funcionamento da rede como um todo, porque tanto a rede ad hoc quanto o ROS a utilizam para estabelecer a interconexão.

A segunda seção apresenta a MANET (*Mobile Ad Hoc Network*) seu conceito e funcionamento, o motivo de se ter escolhido esse tipo de rede, quais os benefícios e a aplicação neste trabalho. Também apresenta o protocolo de roteamento utilizado na rede ad hoc, no caso o OLSR (*Optimized Link State Routing*), introduzindo o conceito de protocolo de roteamento, os tipos de protocolo existentes para essa rede, o funcionamento e objetivo do protocolo e por fim o motivo da escolha do OLSR.

A terceira seção apresenta o ROS (*Robot Operating System*), seu conceito e ampla utilidade ao facilitar o sistema de controle e automação de robôs. Também mostra suas distribuições, como ele atua e os pacotes escolhidos para a implementação deste trabalho. Em seguida são apresentadas duas modelagens da forma de funcionamento do sistema ROS em uma rede de múltiplos robôs. A primeira com apenas um mestre e a segunda com múltiplos mestres.

A quarta e última seção deste capítulo apresenta a interface de voz para robôs colocada neste trabalho para validar a implementação de interconexão dos robôs. Será descrita abordando três tópicos: o *hardware*, o *software* e a máquina de estados.

2.2 TCP/IP (*Transmission Control Protocol / Internet Protocol)*

A estrutura TCP/IP (*Transmission Control Protocol* / *Internet Protocol*), também conhecido como pilha de protocolos TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede. É conhecido como pilha de protocolos por possuir vários níveis de protocolos, sendo que cada nível utiliza as informações e serviços fornecidos pelas camadas abaixo dele, fazendo com que o nível vá subindo e vá se aproximando do usuário.

Como o TCP / IP é a pilha padrão de protocolo de rede na Internet, seu uso em redes móveis ad hoc é uma certeza. Além de alavancar um grande número de aplicativos, seu uso também permite integração direta com a Internet, quando disponível [Referência 5].

Essa pilha de protocolos é inspirada no modelo OSI (*Open Systems Interconnection*) que é a modelo referência da ISO (*International Organization for Standardization*), a qual visa padronizar produtos, processos, procedimentos e serviços a fim de facilitar uma comercialização internacional destes. Diferentemente da OSI que possui sete camadas, a TCP/IP possui quatro. Na figura 2.1 é possível observar as camadas do modelo TCP/IP.

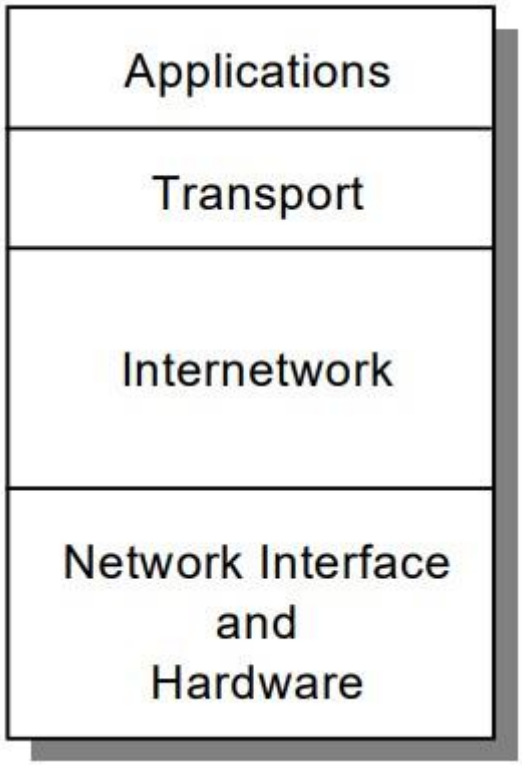

Figura 2.1: Pilha de protocolos TCP/IP, cada camada representa um pacote de funções. Fonte [6].

A explicação de cada de cada camada do TCP/IP está explicada abaixo e foi retirado da referência [Referência 6]:

- Camada de Aplicação: é o nível de maior abstração, mais perto do usuário. Utiliza os serviços fornecidos pelas camadas inferiores para atender protocolos e serviços

oferecidos ao usuário. É nesta camada que o ROS (*Robot Operating System*) e o protocolo de roteamento OLSR (*Optimized Link State Routing*) atuam.

- Camada de Transporte (TCP): A camada de transporte fornece a transferência de dados de ponta a ponta, fornecendo dados de um aplicativo para seu par remoto. Vários aplicativos podem ser suportados simultaneamente. O protocolo de camada de transporte mais utilizado é o Protocolo de Controle de Transmissão (TCP), que fornece entrega de dados confiável orientada por conexão, supressão de dados duplicados, controle de congestionamento e controle de fluxo.
- Camada de Internet: A camada de internetwork, também chamada de camada de internet ou camada de rede, fornece a imagem de "rede virtual" de uma internet (essa camada protege os níveis mais altos da arquitetura da rede física abaixo dela). Protocolo de Internet (IP) é o protocolo mais importante nesta camada. O IP fornece uma função de roteamento que tenta entregar mensagens transmitidas ao seu destino.
- Camada de interface de rede: A camada de interface de rede, também chamada de camada de link ou camada de link de dados, é a interface com o hardware de rede real. Essa interface pode ou não fornecer uma entrega confiável e pode ser orientada por pacote ou fluxo. Na verdade, o TCP / IP não especifica nenhum protocolo aqui, mas pode usar praticamente qualquer interface de rede disponível, o que ilustra a flexibilidade da camada IP [Referência 6].

O sistema operacional atua nas camadas mais baixas, fazendo com que o usuário se preocupe apenas com a camada de aplicação. E já oferece a estrutura para a implementação de uma rede como a ad hoc, porém os sistemas operacionais, por padrão, são configurados para atuar em redes convencionais, que possuem ponto de acesso. Mas apenas modificações na própria camada de aplicação são necessárias para criar a rede ad hoc.

2.3 *Mobile Ad Hoc Network* (MANET)

Os anos 90 assistiram a um rápido crescimento dos interesses de pesquisa em MANETs (*Mobile Ad Hoc Networks*). A infraestrutura e a natureza dinâmica dessas redes exigem um novo conjunto de estratégias de rede a serem implementadas para fornecer uma comunicação eficiente de ponta a ponta [Referência 3]. *Mobile Ad hoc Networks* (MANETs) são redes do tipo ad-hoc no qual os dispositivos apresentam mobilidade, estes dispositivos são denominados de nós móveis (NMs). Estas redes apresentam topologia dinâmica, devido à mobilidade dos nodos e a grande mudança em sua quantidade de nós na rede [Referência 4].

2.3.1 Conceito

Rede Ad Hoc Móvel (MANET, do inglês *Mobile Ad Hoc Network*) consiste em plataformas móveis, aqui colocadas como nós, que são livres para se mover arbitrariamente. É um sistema autônomo de nós móveis que pode operar isoladamente ou pode fazer interface com uma rede fixa. É equipada com transmissores e receptores sem fio, usando antenas que podem ser omnidirecionais (*broadcast*), altamente direcional (ponto-a-ponto), possivelmente dirigível, ou alguma combinação disso [Referência 8].

A ideia das redes Ad Hoc é de uma comunicação descentralizada entre máquinas sem necessitar de um ponto de acesso para conectar as estações. Os MANETs (*Mobile Ad Hoc Networks*) empregam a estrutura tradicional TCP / IP para fornecer comunicação de ponta a ponta entre os nós [Referência 3].

Na figura 2.2 é possível notar a estrutura de uma rede de ponto de acesso, onde todas as estações estão conectadas à central para poderem se comunicar com as outras. E a figura 2.3 apresenta a estrutura de uma rede ad hoc onde todas as estações estão interconectadas.

Figura 2.2: Esquemático que representa uma rede com ponto central controlador. Fonte[2]

Na figura 2.2 as estações são apenas clientes do servidor, já na figura 2.3 todas as estações são tanto clientes como servidores. Isso se dá porque também agem como roteadores se tornando ponto de acesso da rede ad hoc.

Figura 2.3: Esquemático que representa uma rede Ad Hoc. Fonte[2]

2.3.2 Características

Como já foi explicado na seção anterior, os nós da rede ad hoc são equipados com transmissores e receptores sem fio. Os transmissores têm um raio de alcance, este raio é definido pela potência mínima necessária para decodificação do sinal com sucesso. Ou seja, se o dispositivo A está a uma distância tal de B que ele consiga receber o sinal de B, o que significa que a potência recebida de A é minimamente a sensibilidade do sistema receptor de A. Então ele está na vizinhança de B.

A figura 2.4 mostra uma rede ad hoc com o raio de alcance dos transmissores das estações C, G e E. Os nós, ou estações, que estão dentro do alcance dos transmissores são vizinhos, se estiverem participando da rede. Os vizinhos do nó C são: A, B e G. Os vizinhos do nó G são: C e E. Os vizinhos do nó E são: G e F. A estação D não é vizinha de E porque apesar de estar em seu raio de alcance não participa da rede.

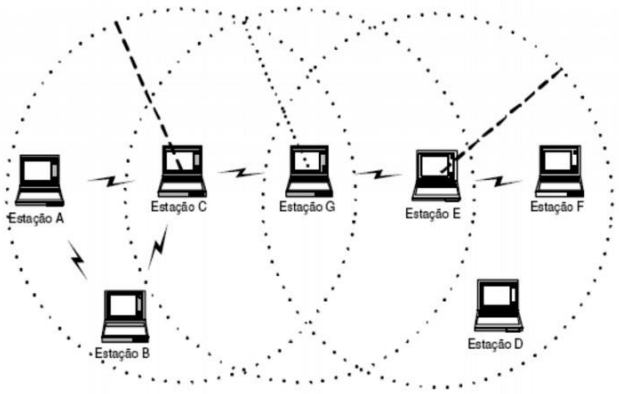

Figura 2.4: Esquemático que representa as vizinhanças dos dispositivos em uma rede ad hoc. Fonte[4]

As características mais importantes a serem citadas a respeito de MANETs, que têm como referência o [Referência 8], são:

i. Topologia dinâmica: os nós são livres para se moverem arbitrariamente. Assim, a topologia da rede, que é tipicamente *multihop* (múltiplos saltos), pode mudar de forma aleatória e rápida e pode consistir de ambos os *links*, bidirecionais e unidirecionais.
ii. Largura de banda contraída, *links* com capacidade variável: redes sem fio vão ter uma capacidade significativamente menor do que as com fio. Além disso, a taxa de transferência da comunicação sem fio, depois de contabilizar os efeitos de acesso múltiplo, desvanecimento, ruído, interferência etc. É frequentemente muito menor que a taxa máxima de transmissão de um rádio.
iii. Operação com restrição de energia: alguns ou todos os nós em um MANET podem contar com baterias ou outros meios de energia. Para esses nós, os critérios de projeto de sistema mais importantes podem ser a conservação da energia.
iv. Segurança física limitada: redes sem fio móveis são geralmente mais propensas a ameaças físicas de segurança do que redes cabeadas. Aumenta a possibilidade de espionagem, ataques de negação de serviço (interrompo o acesso de usuários autorizados a uma rede) precisam ser cuidadosamente considerados. Entretanto, existem técnicas de segurança de link, são geralmente aplicadas a redes sem fio para reduzir as ameaças.

2.3.3 Protocolo OLSR (*Optimized Link State Routing*)

Com todas as estações conectadas em uma rede ad hoc, podendo transmitir e receber dados, não é necessário que todas as mensagens cheguem a todas as estações, o objetivo é chegar apenas ao destinatário. Assim, com a finalidade de melhorar a transmissão de dados, deixar a rede mais eficiente e utilizar da melhor forma os recursos limitados, são desenvolvidos protocolos de roteamento. O protocolo de roteamento visa garantir que a mensagem sairá do remetente e chegará ao destinatário utilizando a melhor rota possível.

Esses protocolos podem ser classificados em três grupos diferentes: global / proativo, sob demanda / reativo e híbrido. Em protocolos proativos de roteamento, as rotas para todo o destino (ou partes da rede) são determinadas no início e mantidas usando um processo de atualização de rota periódica. Em protocolos reativos, as rotas são determinadas quando são

solicitadas pela origem usando um processo de descoberta de rota. Os protocolos de roteamento híbridos combinam as propriedades básicas das duas primeiras classes de protocolos em uma. Ou seja, ambos são reativos e proativos por natureza. Cada grupo tem um número de diferentes estratégias de roteamento, que empregam uma estrutura de roteamento plana ou hierárquica [Referência 3].

O protocolo escolhido neste trabalho, para atuar no MANET, foi o OLSR (*Optimized Link State Routing*). Ele foi projetado para funcionar de maneira completamente distribuída e não depende de nenhuma entidade central. E esta seção foi desenvolvida utilizando o [Referência 7] como principal referência.

O OLSR foi desenvolvido para MANET (*Mobile Ad Hoc Network*). É do tipo proativo e opera por tabela, ou seja, troca informações de topologia com outros nós da rede regularmente. Cada nó seleciona um conjunto de seus nós vizinhos como "*multipoint relays*" (MPRs). Dessa forma, somente os nós selecionados como tais são responsáveis pelo encaminhamento do tráfego de controle, destinado à difusão em toda a rede. Os MPRs fornecem um mecanismo eficiente para o tráfego de controle de inundação, reduzindo o número de transmissões necessárias.

Para entender melhor o funcionamento do protocolo de roteamento OLSR é importante introduzir os principais conceitos da terminologia utilizada:

- Nó: um roteador do MANET que implementa o protocolo OLSR.
- Interface OLSR: um dispositivo de rede de um MANET executando o OLSR. Um nó pode ter várias interfaces OLSR, cada interface atribuída a um endereço IP exclusivo.
- *Link*: par de interfaces OLSR (de dois nós diferentes) suscetíveis a ouvir um ao outro, ou seja, um pode receber tráfego do outro.
- Vizinhança do nó: um nó A é vizinho do nó B se o nó A pode ouvir o nó B, se enquadra na definição de vizinhança da rede ad hoc se ambos estiverem utilizando o protocolo de roteamento OLSR. Ou seja, se existe um *link* entre a interface OLSR do nó A e a interface OLSR do nó B.
- Salto vizinho: um nó ouvido por um vizinho.
- 2-salto vizinho: um nó ouvido pelo vizinho do vizinho.
- *Multipoint relay* (MPR): um nó é selecionado por seu vizinho de um salto para "retransmitir" todas as mensagens de difusão que recebe. Contanto que a mensagem não seja uma duplicata.
- *Multipoint relay selector* (MPR *selector, MS*): um nó que seleciona o seu vizinho de um salto, nó X, para ser o MPR, será chamado de *multipoint relay selecto*r do nó X.

A partir das figuras 2.5 e 2.6 é possível visualizar como diminui a quantidade de informação redundante a ser distribuída na rede ao se utilizar a otimização MPR. Primeiramente na figura 2.5 mostra uma rede disseminando as informações de controle sem a utilização da otimização MPR, inundando a rede. Na figura 2.6 mostra a mesma rede da figura 2.5, porém agora utilizando a otimização MPR. Dessa forma, apenas os nós escolhidos como MPRs passam as informações de controle para todos os seus vizinhos. A representação da figura 2.6 é chamada de *backbone* do MPR.

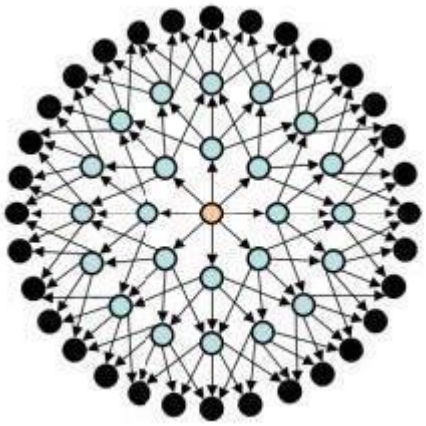

Figura 2.5: rede sendo inundada, sem a utilização da otimização MPR. Fonte[5]

.

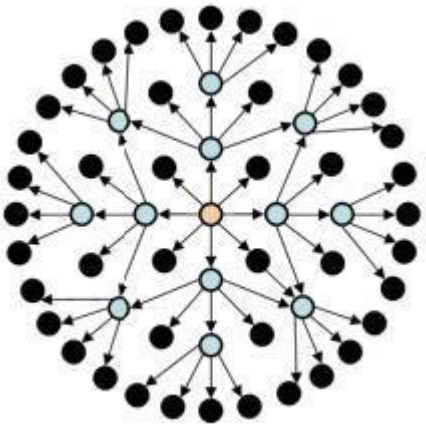

Figura 2.6: Esquemático mostrando o backbone do MPR. Fonte[5]

O protocolo OLSR é uma otimização do protocolo de roteamento *Link State* clássico adaptado para redes móveis ad hoc. Por isso herda a estabilidade de um algoritmo de estado de *link* e tem a vantagem de ter rotas imediatamente disponíveis quando necessário, devido à sua natureza

proativa. Usa o roteamento salto-a-salto, ou seja, cada nó usa suas informações locais para rotear os pacotes.

Para manter a rede atualizada o protocolo constantemente envia mensagens de controle para atualizar a vizinhança, as rotas e os custos. Porém, não requer a transmissão confiável de mensagens de controle, porque cada nó envia essas mensagens periodicamente e pode, portanto, sustentar uma perda razoável de algumas dessas mensagens.

O OLSR pode otimizar a reatividade para alterações topológicas, reduzindo o intervalo de tempo máximo para a transmissão periódica de mensagens de controle. Além disso, como o OLSR mantém continuamente rotas para todos os destinos na rede, o protocolo é benéfico para padrões de tráfego nos quais um grande subconjunto de nós está se comunicando com outro grande subconjunto de nós e onde os pares [*source*, *destination*] estão mudando com o tempo.

As informações de vizinhança, rotas e custos são disponibilizadas em uma tabela a qual o usuário tem acesso. Esta tabela será apresentada no capítulo 3, quando será implementado o protocolo.

Diferentemente do OLSR padrão da *Internet Engineerig Task Force* (IETF) que utiliza como métrica o número de saltos, necessários para chegar ao destinatário, para definir a melhor rota. O OLSR utilizado contém uma atualização que utiliza o *Expected Transmission Count* (ETX). O ETX calcula o número de transmissões e retransmissões necessárias para uma mensagem chegar ao seu destino. Essa modificação foi feita uma vez que se percebeu que o número de saltos não garantia uma melhor rota. Isso porque uma das conexões pode estar fraca e pode ser necessário enviar várias vezes a mensagem a fim de que chegue a seu destino.

2.4 *Robot Operating System* (ROS)

Robot Operating System (ROS) é um *framework* flexível para escrever *software* robótico. Ele contém coleção de ferramentas, bibliotecas e convenções que visam simplificar a tarefa de criar comportamentos de robôs complexos e robustos em uma ampla variedade de plataformas robóticas. Fonte[11]

O ROS tem uma estrutura de alto nível para modelagem e desenvolvimento de controle e automação de robôs juntamente com outras ferramentas. Isso faz com que ele seja exaustivamente explorado por empresas de tecnologia, universidades. O que é ótimo porque constantemente são implementados novos pacotes e ele continua a evoluir e avançar tecnologicamente.

Um *framework* cooperativo e gratuito, o ROS, permite que qualquer pessoa crie seu pacote e compartilhe, utilizando os pacotes já desenvolvidos ou criando outros completamente novos. Isso faz com que novas aplicações surjam com mais facilidade, uma vez que se pode usar pacotes novos juntamente com os que já estavam implementados. Entretanto, com o passar do tempo se os pacotes criados não forem mantidos pelo criador ou por uma terceira pessoa, fica sem suporte quando sair novas distribuições do ROS.

Um dos critérios no desenvolvimento do ROS foi a adoção de comunicação *peer-to-peer* (P2P) ao invés da comunicação centralizada. Em cenários onde múltiplos nós propagam informações coletadas por sensores para nós de processamento, o modelo P2P usualmente oferece melhor crescimento em escala e desempenho do que os modelos de comunicação centralizadas [Referência 9]. Quanto menos centralizado o sistema mais autonomia tem suas partes, o ROS já foi pensado para descentralizar e este trabalho propõe uma descentralização ainda maior, tanto na sua rede utilizada quanto na sua forma de atuação.

Com um tutorial amplo e detalhado, o ROS, é construído de forma a facilitar o acesso, onde qualquer pessoa pode aprender como utilizá-lo a qualquer momento. Porém, ter conhecimento de linguagens de programação e estrutura computacional facilita muito para aqueles que querem usufruir desse *framework*. As duas principais linguagens de programação que possui suporte é o C/C++ e o Python, possuindo até aplicação de programação no ROS, o *roscpp* e o *rospy*.

A seguir será explicado o funcionamento do ROS, bem como conceitos importantes para o entendimento deste. Para isto foram divididos em quatro tópicos: distribuições, pacotes, nós, definições e aplicações e por fim sistemas com múltiplos mestres ROS. As definições e as informações a respeito do ROS foram retiradas da sua página oficial: fonte[7].

2.4.1 Distribuições

As distribuições do ROS são conjuntos de pacotes do ROS, mais adiante será exposto o conceito de pacote. Um exemplo usado para entender melhor essa definição é que são semelhantes às distribuições Linux, como: Ubuntu, Mint, Debian, etc. O objetivo é garantir uma certa estabilidade para os desenvolvedores, assim sempre que uma nova distribuição é lançada os pacotes principais são mantidos, alguns *bugs* são melhorados ou arrumados e os desenvolvedores podem atualizar seus pacotes. Ou podem continuar mantendo e fazendo as devidas atualizações em seus pacotes continuando na distribuição antiga, de fato depende da aplicação.

Diversas vezes a distribuição escolhida para se trabalhar é a que já tem algum trabalho desenvolvido acerca do assunto, porque muitas vezes o pacote é criado, então surgem novas versões do ROS e os mantedores não atualizam levando para essas novas versões, então ao invés de gastar tempo entendendo o pacote e atualizando o mesmo é mais fácil apenas utilizalo para desenvolver o que se quer. Porém, ao escolher uma versão mais antiga é necessário avaliar se realmente vale a pena e se essa ainda possui suporte.

Basicamente o ROS não se responsabiliza em manter, atualizar e cuidar dos pacotes que os desenvolvedores criam, apenas garante que seu sistema tenha uma base completa e sem *bugs* para garantir que as pessoas possam criar *softwares* robóticos de qualidade e mantê-los.

Todo ano em maio é liberado uma nova distribuição, sendo que lançamentos em anos pares serão uma versão LTS (*long-term support*) com suporte de cinco anos e em anos ímpares são lançamentos normais com suportes por dois anos.

Abaixo estão listadas as distribuições do ROS e suas respectivas datas de lançamento:

- Box Turtle – 02 de março, 2010;
- C Turtle – 02 de agosto, 2010;
- Diamondback – 02 de março, 2011;
- Eletric Emys – 30 de agosto, 2011;
- Fuerte Turtle – 23 de abril, 2012;
- Groovy Galapagos – 31 de dezembro, 2012;
- Hydro Medusa – 04 de setembro, 2013;
- Indigo Igloo – 22 de julho, 2014;
- Jade Turtle – 23 de maio, 2015;
- Kinetic Kame – 23 de maio, 2016; ✦ Lunar Loggerhead – 23 de maio, 2017; ✦ Melodic Morenia – 23 de maio, 2018.

A distribuição utilizada neste trabalho foi a Kinetic Kame. Entretanto, já há implementações no ROS com redes ad hoc em distribuições mais antigas, Indigo Igloo. Ainda assim, foi decidido trabalhar com uma versão mais nova do ROS porque a antiga não foi atualizada, não tendo mais suporte. Também porque a implementação já feita não atenderia os objetivos propostos neste trabalho.

2.4.2 Pacotes

Pacotes são a unidade básica de organização do *software* ROS. Os *softwares* são organizados em pacotes o qual tem que conter nós, que será explicado adiante, uma biblioteca

independente, uma base de dados, arquivos de configurações e algo que constitua logicamente um módulo útil. O objetivo dos pacotes é garantir que o *software* tenha funcionalidade e seja fácil de ser mantido pelos desenvolvedores além de facilitar também que seja reutilizado por terceiros.

Estes também têm um papel importante na organização dos arquivos, possui comando exclusivos para achá-los, criá-los, checar o que estão fazendo, instalar, etc. E tudo isso pode ser feito através da linha de comando do terminal. Basicamente, tudo o que poderia ser feito pelo sistema de arquivos pode ser feito através do terminal.

Os pacotes base para a implementação deste trabalho foram desenvolvidos em versões mais antigas, porém, foram atualizados para as novas distribuições. Mais adiante serão expostos os pacotes utilizados para a implementação deste trabalho explicando seu funcionamento.

2.4.3 Definições e Aplicações

Antes de iniciar explicando as definições da linguagem utilizada no ROS é importante ressaltar que tanto no ROS quanto em MANET existe o termo nó. Porém, possuem definições diferentes.

Um nó é um executável que usa o ROS para se comunicar com outros nós, dentro de um pacote. Para se comunicarem utilizam bibliotecas cliente do ROS, essas bibliotecas podem ser implementadas usando as linguagens de programação C++ ou Python. Esses nós são usados para executar o que se quer, podendo fazer uso de várias bibliotecas e quando em conjunto podem formar uma rede complexa. O exemplo de um robô: um nó executa o controle da localização, um é responsável por documentar informações dos sensores, um outro executa o tratamento informações extraídas dos sensores, outro mostra os gráficos dessas informações, ou seja, cada um vai executar uma atividade.

Essa modelagem foi pensada para diminuir a complexidade do sistema, trabalhando de forma separada e conseguindo observar o que cada um faz e admitindo uma maior taxa de erros por serem mais isolados. Isso é uma vantagem, porque um processo não interfere no outro, sendo mais fácil, na maior parte das vezes, de identificar o problema e consertar.

O comando ROS conhecido para tratar de nós é o *rosnode*. E estes podem ser mapeados e executados através da linha de comando. Eles se encontram dentro de pacotes juntamente com as bibliotecas que farão uso.

Os tópicos são o meio pelo qual os nós utilizam para prestar serviço para outros nós, ou seja, é um prestador de serviços dos nós. Cada tópico tem cliente e servidor e um tópico pode ter vários servidores e vários clientes. O cliente invoca um serviço e o servidor utiliza o tópico para publicar / prestar o serviço e o nó cliente assina / subscreve aquele tópico para receber as informações. Dessa forma, sempre que há uma atualização no nó servidor ele atualiza o tópico e este assina o nó cliente. Um esquemático dessa prestação de serviço está mostrado na Figura 2.7.

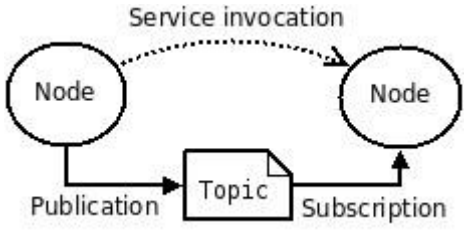

Figura 2.7: Esquemático da comunicação entre dois nós. Fonte[6]

Basicamente o tópico recebe o serviço e executa para o nó solicitante. A solicitação / resposta é feita por meio de um *Serviço,* que é definido por um par de mensagens : uma para a solicitação e outra para a resposta. Essas mensagens de solicitação e resposta são definidas em arquivos srv, que são compilados no código-fonte por uma biblioteca cliente do ROS. Existem outros tipos de mensagens que são as informações trocadas pelos nós e podem ser de vários tipos.

Com as definições de nó, tópico, publicar/assinar entende-se como o sistema é dividido dentro do ROS e uma vez dividido como trabalham se comunicando. O mestre gerencia os clientes e os servidores em tópicos e serviços, sua função é fazer com que os nós consigam se enxergar e se comunicar par a par (*peer-to-peer*). Para conseguir executar uma rede ROS é necessário que o mestre esteja sendo executado, se ele for parado quando algum código estiver sendo compilado, este também para. Pois além de fazer o gerenciamento dos nós e fazer com que exista a comunicação entre eles, o mestre também carrega componentes essenciais para o processamento dos pacotes. Todo sistema ROS possui mestre, o qual é executado pelo comando *roscore*. Nas imagens abaixo é possível observar como o mestre trabalha.

A Figura 2.8 mostra o ROS em funcionamento com dois nós, um referente as informações coletadas da câmera, no caso as imagens. E o outro nó, um aplicativo de visualização de imagens. O nó da câmera anuncia para o mestre que ela possui imagens.

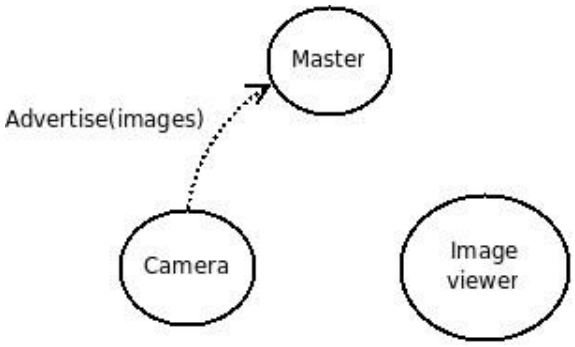

Figura 2.8: O nó da câmera anuncia que tem a mensagem do tipo imagem. Fonte[7]

Na Figura 2.9 o nó da câmera publica suas imagens no tópico, chamado de imagens. E o nó de visualização de imagens anuncia para o mestre que ele quer assinar o tópico imagens.

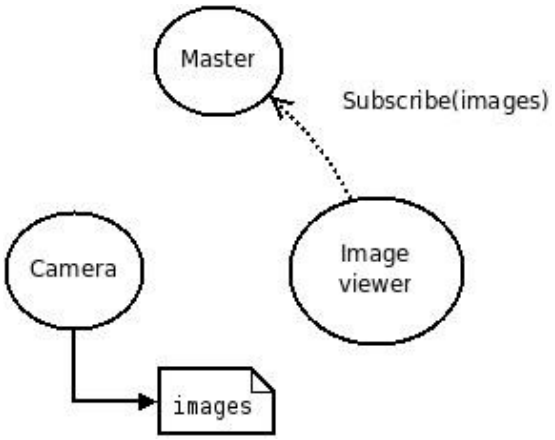

Figura 2.9: O nó da câmera publica no tópico, chamado de imagens. O nó de visualização de imagens anuncia que quer assinar o tópico imagens. Fonte[7]

Na Figura 2.10 o mestre leva o nó de visualização de imagens a assinar o tópico imagens. Assim o tópico imagens tem o servidor, nó da câmera, e o cliente, nó de visualização de imagens. O nó da câmera publica no tópico imagens e o nó de visualização de imagens assina esse tópico. A definição de mestre e seu funcionamento é muito importante para o entendimento deste trabalho.

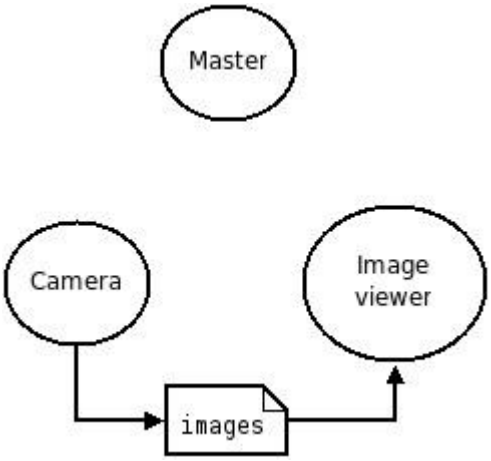

Figura 2.10: o mestre leva o nó de visualização de imagens a assinar o tópico imagens. Fonte[7]

2.4.4 Sistema com mestres ROS único

Pode-se adotar pelo menos duas modelagens da forma de funcionamento do sistema ROS em uma rede de múltiplos robôs. Uma será explicada nessa seção e a outra na seção seguinte.

A primeira modelagem é manter apenas um mestre no sistema, um computador atuando como mestre, e os outros atuando como servos. Ou seja, apenas um dos robôs é o mestre que comanda os outros trabalhando como nós no sistema ROS. Para isso, seria necessário utilizar o protocolo *Secure Shell* (SSH). A explicação referente a SSH tem como referência [Referência 11].

O SSH (*Secure Shell*) é um protocolo para acesso remoto seguro e outros serviços de rede em uma rede insegura. E consiste em três principais componentes:

- O Protocolo de Camada de Transporte [SSH-TRANS]: fornece autenticação, confidencialidade e integridade do servidor. Pode opcionalmente também fornecer compressão. A camada de transporte normalmente será executada em uma conexão TCP / IP, mas também pode ser usada sobre qualquer outro fluxo de dados confiável.
- O protocolo de autenticação do usuário [SSH-USERAUTH]: autentica o usuário do lado do cliente no servidor. Ele é executado sobre o protocolo da camada de transporte.
- O protocolo de conexão [SSH-CONNECT]: multiplexa o túnel criptografado em vários canais lógicos. Ele é executado sobre o protocolo de autenticação do usuário.

Os computadores que são os nós, nesta modelagem, são chamados *hosts* (hospedeiros), pois recebem uma extensão do computador principal, o mestre.

O ROS já possui implementações neste sentido, inclusive nos tutoriais é explicado como implementar esse sistema. Isso é muito utilizado quando se quer coordenar um robô móvel de um computador para conseguir obter e tratar os dados coletados pelo robô de forma mais eficiente, por exemplo.

A figura 2.11 mostra um esquemático de como seria essa implementação de um único mestre. No centro temos o computador número 1, o qual o *roscore* está sendo executado. E em volta outros computadores atuando como nós. As linhas pontilhadas na imagem representam o mestre gerenciando as outras máquinas. E a linha preta contínua representa os tópicos, serviços e ações dos nós.

Uma desvantagem desse modelo é que se o computador 1, por algum motivo, não participar mais dessa rede o sistema para de funcionar. Ou seja, cessa a comunicação entre os nós.

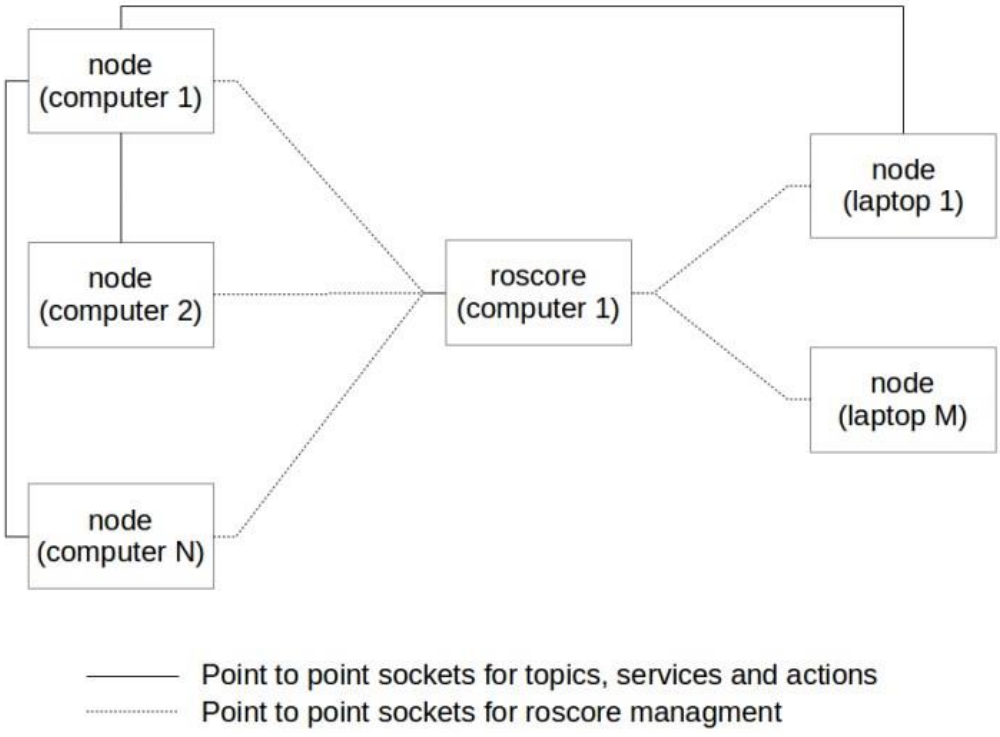

Figura 2.11: Sistema com mestre único. Fonte[8]

2.4.5 Sistema com múltiplos mestres ROS

A segunda modelagem, da forma de funcionamento do sistema ROS em uma rede de múltiplos robôs, seria ter múltiplos mestres presentes no sistema.

Esta seção tem como referência [Referência 14].

O desafio é estabelecer uma comunicação entre os mestres. A solução desse desafio foi implementada no pacote *multimaster_fkie* do ROS, desenvolvido por Alexander Tiderko[12]. A

imagem abaixo mostra a proposta do pacote *multimaster_fkie*, onde três grupos iguais ao da figura 2.11 estabelecem uma interconexão. Na figura 2.12 a linha vermelha representa os tópicos, serviços ou ações na rede comum. E a linha preta representa tópicos, serviços ou ações na rede local.

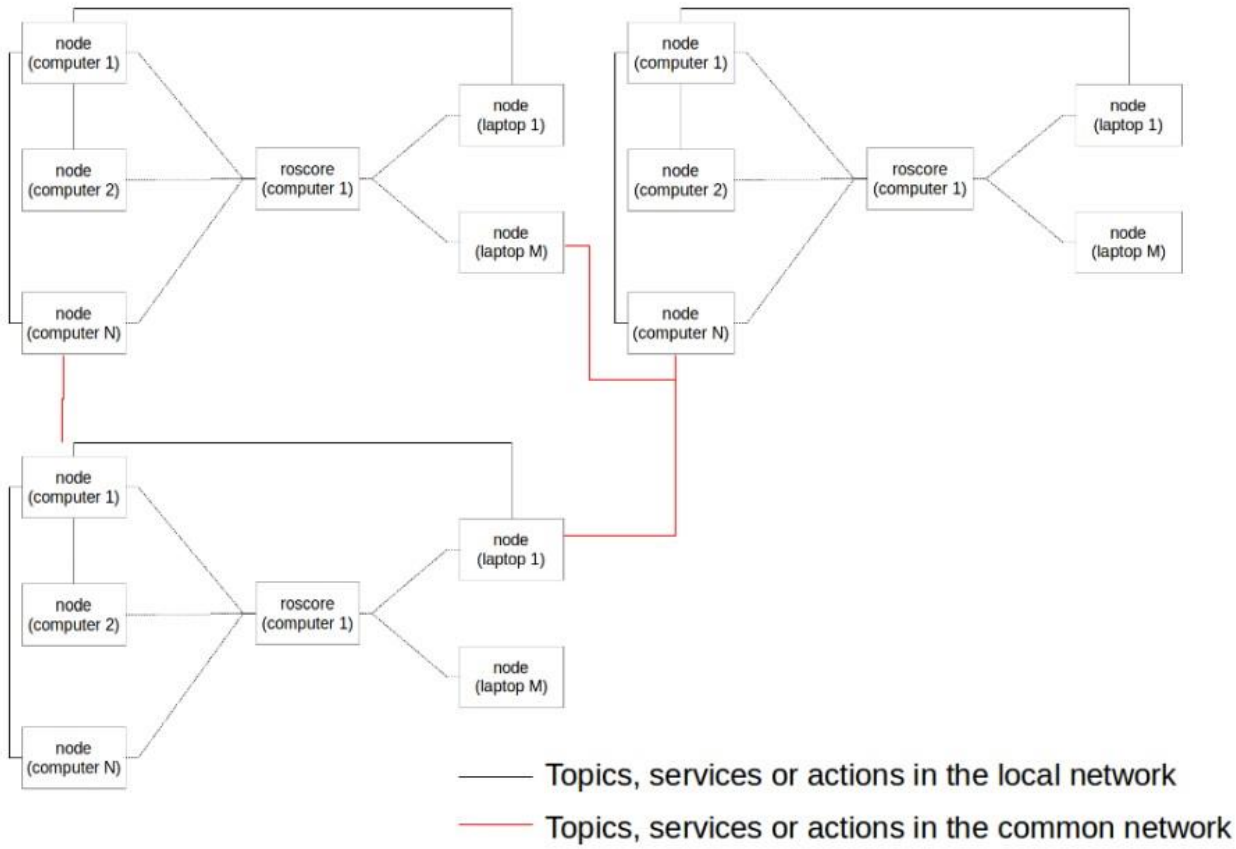

Figura 2.12: Sistema com múltiplos mestres. Fonte[8]

O *multimaster_fkie* consiste principalmente de dois nós: o *master_discovery* e o *master_sync*. Os principais recursos do nó *master_discovery* são resumidos a seguir:

- Periodicamente envia mensagens *multicast* à rede comum para avisar os outros mestres ROS de sua presença. E para detectar quaisquer outros mestres do ROS disponíveis na rede.
- Verifica o *roscore* local quanto há alterações na rede local e notifica todos os outros mestres do ROS, na rede comum, dessas alterações.

As principais características do nó *master_sync* são resumidas a seguir:

- Usa as informações fornecidas por outros nós *master_discovery* para registrar tópicos e serviços remotos no *roscore* local.

- As informações fornecidas pelos nós de *master_discovery* remoto também são usadas para atualizar as informações sobre tópicos e serviços.
- Pode ser configurado para selecionar quais hosts, tópicos e serviços serão sincronizados ou ignorados. Por padrão, todos os tópicos e serviços de todos os hosts são sincronizados, portanto, para reduzir a largura de banda necessária é possível sincronizar apenas os tópicos e serviços desejados.

Porém, a forma de projeto proposta neste trabalho é a de vários robôs com seu próprio mestre. Ou seja, nenhum robô atuando como nó. A figura 2.13 apresenta três computadores, cada um executando o *roscore,* e gerenciando nós que estão nas suas respectivas interfaces. A linha preta representa tópicos, serviços ou ações na rede local. E a linha vermelha representa tópicos, serviços ou ações na rede comum. É válido salientar que a rede comum utilizada é a rede Ad Hoc móvel.

O objetivo é estabelecer uma interconexão o mais independente possível. Isso é proposto para caso ocorra alguma instabilidade na rede, ou em alguma máquina, as outras continuem executando suas tarefas.

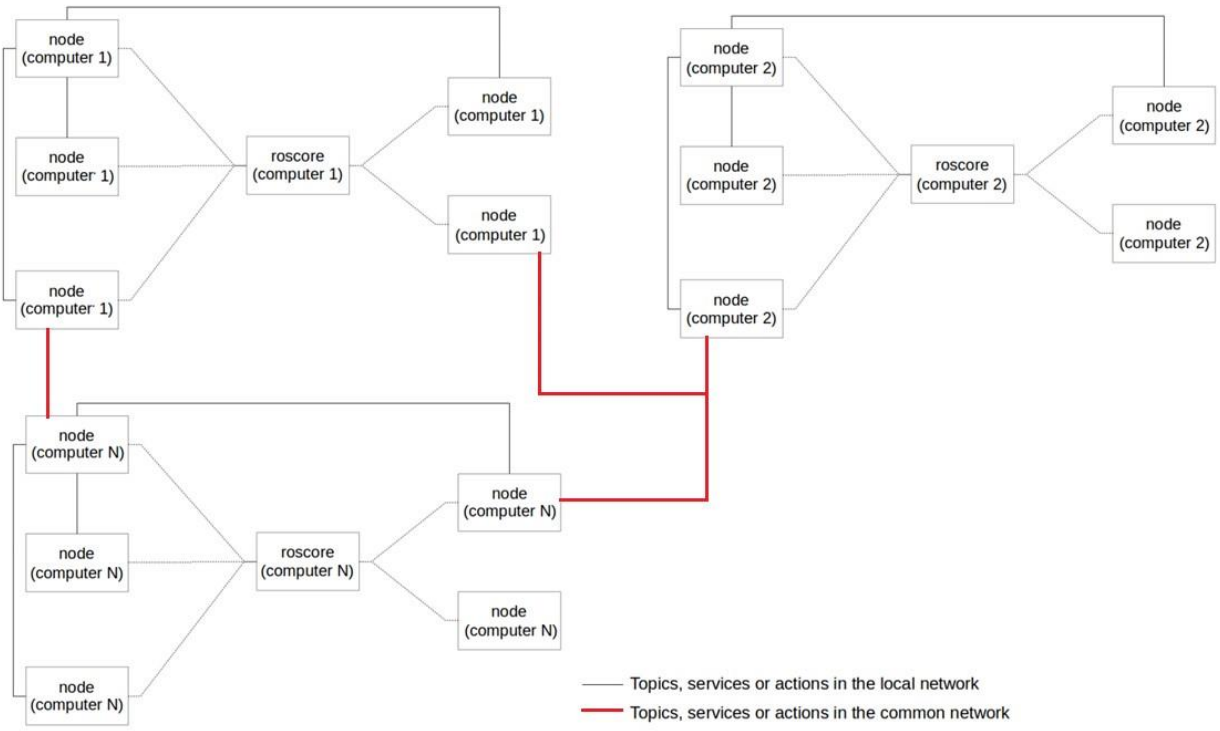

Figura 2.13: Sistema com múltiplos mestres ROS, cada computador com um mestre.

2.5 Interface de voz para robôs

Com o objetivo de validar o trabalho desenvolvido foi utilizada a interface de voz para robôs, a qual foi implementada utilizando ROS e uma máquina de estados, além de várias bibliotecas e programas. A seguir será explicado a estrutura dessa interface em três partes. A primeira abordará o *hardware* utilizado, a segunda o *software* e a terceira a máquina de estados.

2.5.1 *Hardware*

O *Notebook* utilizado foi o Lenovo com as seguintes especificações:

- Processador Intel® Core™ i5, 2.53 GHz;
- 4.00 GB de memória RAM;
- Sistema Operacional Ubuntu 16.04.

Foi selecionado um dos três computadores para ser implementado o sistema e os outros permanecerem sem nenhuma parte dessa interface.

O computador utilizado já possui saídas de som e microfone o que é necessário para a real comunicação com o robô. Porém, para diminuir os erros nos testes foi inserido entrada de texto para o usuário e saída de voz para o robô, durante o diálogo.

2.5.2 *Software*

A fim alcançar uma comunicação robô-humano utilizou-se de algumas bibliotecas e programas. Para reconhecimento de voz, reconhecimento de fala e síntese de voz. São elas: Festival e *PocketSphinx*.

Festival é uma biblioteca desenvolvida pela Universidade de Edimburgo para síntese de áudio. Biblioteca C++, multilíngue que provê a síntese de fala em inglês (britânico e americano), espanhol e galês. Implementa funções de alto nível, simples e práticas, para serem acopladas em outros códigos e sistemas. Também oferece uma interface cliente-servidor que permite que outros programas acessem suas funcionalidades. A língua escolhida para reprodução foi o inglês.

O *Sphinx* é uma biblioteca de reconhecimento de áudio desenvolvida pela Universidade Carnegie Mellon. O processo de reconhecimento usado pelo *CMU Sphinx* tem duas partes:

- *Split*: separa cada áudio em partes menores, por silêncio;
- *Matching*: cada áudio é equiparado com todas as possíveis palavras.

Isso significa que primeiro as frases, orações são separadas em palavras e depois cada sinal de voz, que seria de cada palavra, é comparada aos sinais contidos no banco de dados e o que tiver uma maior correspondência vai ser a utilizada. O modelo da língua inglesa provido pelo CMU *Sphinx* foi usado neste trabalho.

Através do ROS foi possível estabelecer uma comunicação entre bibliotecas. O funcionamento do sistema será detalhado no capítulo 3, na seção de implementação da interface de voz.

2.5.3 Máquina de Estados Finitos

Máquina de estados finitos (FSM, do inglês *Finite State Machine*) representa, neste trabalho, a lógica implementada do programa computacional para definir a saída de voz do robô a partir da entrada inserida. Ela só fica em um estado por vez e a mudança é feita a partir de uma transição pré-definida, sendo que cada estado possui uma ação e objetivo. Este tipo de FSM é denominada máquina de Mealy, porque sua saída depende apenas da entrada, independendo do estado atual.

Esta foi implementada para ser o "cérebro" do robô, fazer a tomada de decisão de como ele vai agir de acordo com a conversa. De fato, é uma forma muito engessada de diálogo onde a máquina espera do humano frases feitas para então retornar frases feitas também. Outras soluções de tomada de decisão foram propostas em estudos que continuaram este, porém foi decidido que este seria um melhor exemplo para a aplicação em questão. Na figura abaixo é possível ver o esquemático do funcionamento da máquina de estados, seus estados e possíveis conexões entre eles.

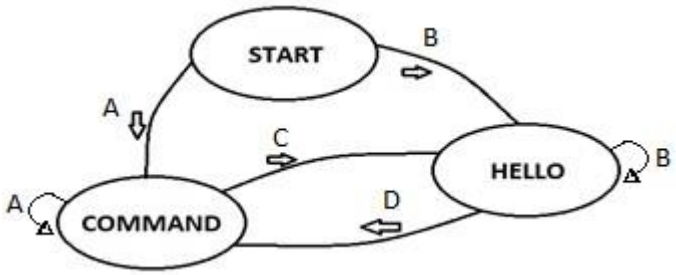

Figura 2.14: Máquina de estados desenvolvida a fim de determinar o modo de resposta do computador.

Abaixo estão listados os estados e suas descrições:

- START: estado inicial que apenas espera alguma entrada para determinar o próximo estado. Nele o robô se apresenta.

- HELLO: estado em que o robô faz perguntas para conhecer o seu “senhor”, quem ele vai servir.
- COMMAND: estado em que o robô recebe comandos e busca executá-los.

As transições A e D são iguais, se ele recebe alguma ordem da pessoa com quem está dialogando, então vai para o estado COMMAND. As transições B e C também são iguais, se ele recebe algum cumprimento do humano vai diretamente para o estado HELLO.

Por exemplo, se assim que a máquina é iniciada a pessoa insere a entrada de voz: olá Aramis. A transição ‘B’ acontece e a máquina vai para o estado HELLO. Porém, se a máquina for iniciada e a entrada de voz inserida for: fale sobre o AMORA (projeto do laboratório de robótica e automação - LARA). Então a transição ‘C’ ocorre e a máquina vai para o estado COMMAND. Se estiver no estado COMMAND e não for cumprimentado, continua neste estado aguardando por comandos, transição ‘A’. Caso seja, ele muda para o estado HELLO. Semelhantemente, se estiver no estado HELLO e não receber comandos, não muda de estado, transição ‘B’.

Capítulo 3

Desenvolvimento

3.1 Introdução

Este capítulo visa apresentar a implementação da rede ad hoc móvel (MANET) para uso integrado ao ROS. Utilizar pacotes ROS para alcançar uma comunicação entre mestres. Explicar o funcionamento das partes separadamente. E expor as decisões de projeto que foram tomadas ao longo do desenvolvimento. Além de mostrar o processo para se alcançar os resultados.

O processo para alcançar os resultados foi composto de cinco etapas:

1. Definir quais robôs ou computadores iriam ser utilizados.
2. Escolher o sistema operacional com o qual se iria trabalhar, o ROS neste trabalho não foi entendido como sistema operacional a partir das explicações expostas no capítulo 2.
3. Implementação e configuração da MANET e do protocolo de roteamento, escolhido para trabalhar com a rede.
4. Foram realizados testes de funcionamento da rede Ad Hoc a partir do protocolo de roteamento OLSR.
5. Implementação de multi-mestres ROS e seu funcionamento com a rede Ad Hoc.

Sendo assim, o capítulo é dividido em cinco seções. São elas: robôs e computadores, escolha do sistema operacional, implementação da rede ad hoc móvel (MANET), teste de funcionamento da rede ad hoc móvel (MANET), implementação de multi-mestres ROS.

3.2 Robô e Computadores

Os robôs para os quais o sistema foi pensado e desenvolvido são três da família Pioneer modelo P3-DX da Mobile Robots. Porém só um deles foi utilizado neste trabalho. Apelidado carinhosamente de Aramis, possui duas rodas, sonares, além de uma tela e Kinect anexados a sua estrutura metálica que é presa a sua parte superior. Aramis possui as seguintes especificações:

- Processador Intel® Celeron® CPU J1800 @ 2,41 GHz x 2;

- 64-bits;
- Sistema Operacional Linux Ubuntu 16.04.

Figura 3.1: Aramis, robô móvel da família Pioneer modelo P3-DX da Mobile Robots. Fonte[10].

O *Notebook* utilizado foi um Lenovo com as seguintes especificações:

- Processador Intel® Core™ i5, 2.53 GHz;
- 4.00 GB de memória RAM;
- Sistema Operacional Ubuntu 16.04.

E por fim, o outro computador utilizado tem as seguintes:

- Processador Intel® Core™ 2 Duo CPU E8200 @ 2,66 GHz x 2;
- 64-bits;
- Sistema Operacional Linux Ubuntu 16.04.

3.3 Escolha do sistema operacional

O sistema operacional (SO) utilizado nos três computadores foi o Linux Ubuntu 16.04 LTS, pois é o sistema com o qual o ROS tem mais afinidade e suporte. Essa afirmação é sustentada pelo fato de que todos os seus pacotes basicamente foram desenvolvidos no sistema operacional Linux. Além disso, a medida que o Linux ganha novas versões e se desenvolve o ROS o acompanha fazendo suas novas distribuições, com suporte e instalação para as novas versões.

Podem surgir dúvidas quanto ao fato do ROS ser um sistema operacional e de ser usado com outro sistema operacional, o Linux no caso. Mas pela hierarquia de camadas apresentadas no TCP/IP, capítulo 2, o sistema operacional é o que atua nas camadas mais baixas. Enquanto o ROS

atua na camada de aplicação, a camada mais alta. Considerando este fator, o ROS não atua como um sistema operacional.

O Linux é conhecido por ser mais voltado para desenvolvedores por possuir uma maior flexibilidade, ser totalmente gratuito e mais seguro de ameaças. E cada vez que uma nova versão é lançada fica mais amigável para cliente e surgem mais ferramentas de suporte.

Outro motivo para escolha do SO é a facilidade de se criar uma rede Ad Hoc nele, diferentemente do Windows se tem abertura não só para criar uma rede desse tipo como também configurar e mexer em suas configurações como potência de transmissão, o que foi muito importante pata atestar a funcionalidade dele.

3.4 Implementação da rede Ad Hoc móvel (MANET)

Como foi exposto, o sistema operacional atua nas camadas mais baixas da pilha de protocolos TCP/IP, executando os processos de baixo nível e disponibilizando para o usuário uma interface mais amigável. Assim, a partir do Linux Ubuntu a rede MANET foi criada.

O próprio Ubuntu já oferece a opção de rede ad hoc, ao ir nas opções para se criar uma rede Wireless. A criação da rede está mostrada na figura 3.2. As configurações necessárias foram apenas no endereço IP, máscara e *gateway*. Essa configuração foi necessária em todos os dispositivos uma vez que nenhum protocolo de endereçamento IP automático foi implementado para essa rede.

A rede foi criada em um dos computadores, em seguida este computador foi colocado para se conectar a ela. Na Figura 3.3 é mostrado essa conexão. Uma vez que um dispositivo está conectado, ele roteia a rede e as máquina que estiverem no alcance dessa rede conseguem captá-la e se conectar a ela. Vale ressaltar que nenhum código de segurança foi determinado, assim, basta o usuário escolher essa rede para se conectar.

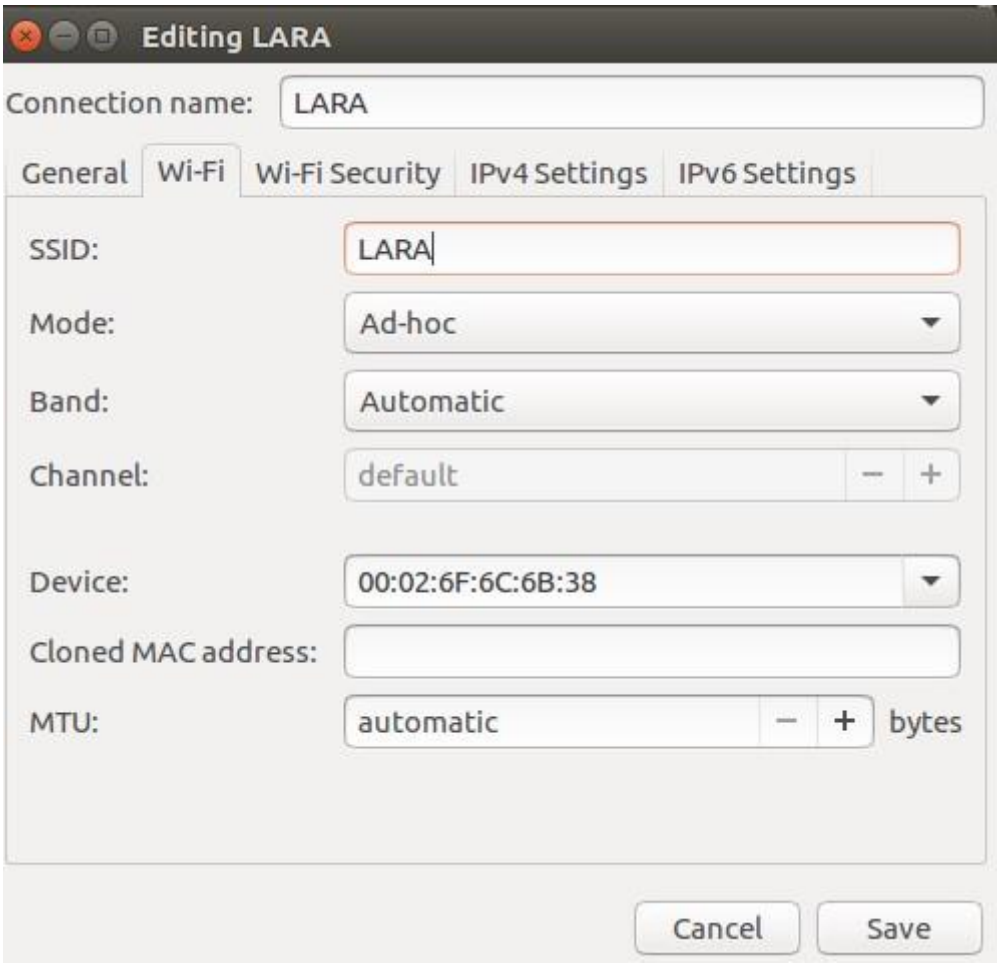

Figura 3.2: Página de criação e configuração da rede no Ubuntu 16.04

.

Figura 3.3: Rede em funcionamento em um dos computadores.

Após a criação da rede Ad Hoc e da configuração de todos os três computadores de forma a estarem participando desta, o protocolo OLSR foi instalado e o próximo passo foi configurá-lo para ele atuar da forma correta.

O protocolo de roteamento OLSR instalado não demanda muitas configurações para sua atuação na rede. A partir de sua página de configurações algumas mudanças foram feitas para incluir o mapeamento dos IPs, determinados para os computadores participantes da MANET. Em seguida foi necessário descobrir o nome da MANET, definida pelo Ubuntu, através do terminal para definir para o OLSR em qual rede ele atuaria.

3.5 Teste de funcionamento da rede Ad Hoc móvel (MANET)

A forma usada de garantir que a rede está funcionando como deveria é confirmar se os computadores realmente conseguem formar caminhos, identificar os MPRs corretamente, os saltos, a vizinhança, os links.

Com este objetivo foi testado a estrutura da figura 3.4, a qual mostra que as estações C e E devem estar na vizinhança da estação G. E G também deve estar em ambas as vizinhanças das estações C e E. Porém, a estação C não deve estar na vizinhança da estação E e nem a estação E deve aparecer na vizinhança do C.

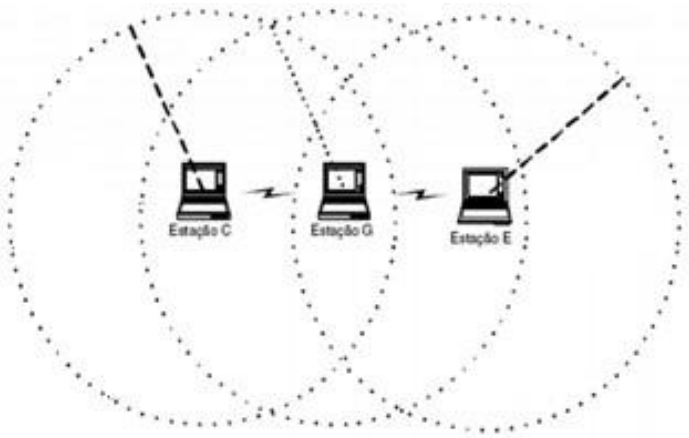

Figura 3.4: Posicionamento físico da rede ad hoc.

Para um melhor entendimento dos resultados, cada estação será associada a um endereço IP:

- Estação C: 192.168.0.102;
- Estação G: 192.168.0.103;
- Estação E: 192.168.0.101;

Quanto aos MPRs, todas as estações têm de ser neste caso, pois existe um único caminho possível. Ou seja, todos os nós vão enviar pacotes *broadcast*. Também é mostrado o número de MPRs que se encontram na rede como um todo, no caso 3.

Todos os nós devem ter um *link* entre eles, porque todos tem uma rota de comunicação estabelecida entre as suas interfaces OLSR.

Figura 3.5: Estação C do esquemático mostrado na figura 3.4.

A figura 3.5 mostra a estação C da figura 3.4. Em sua vizinhança consta apenas a estação G, como era esperado. Ela consta como um MPR da rede e tem acesso a estação E com dois saltos, utilizando a estação G como caminho.

Na figura 3.6 está a estação G da figura 3.4. Em sua vizinhança constam as estações C e E, também como era esperado. Ela consta como um MPR da rede e tem acesso a ambas as estações C e E, por um salto.

A figura 3.7 mostra a estação E da figura 3.4. Em sua vizinhança consta apenas a estação G, como o esperado. Ela consta como um MPR da rede e tem acesso a estação C com dois saltos, utilizando a estação G como caminho.

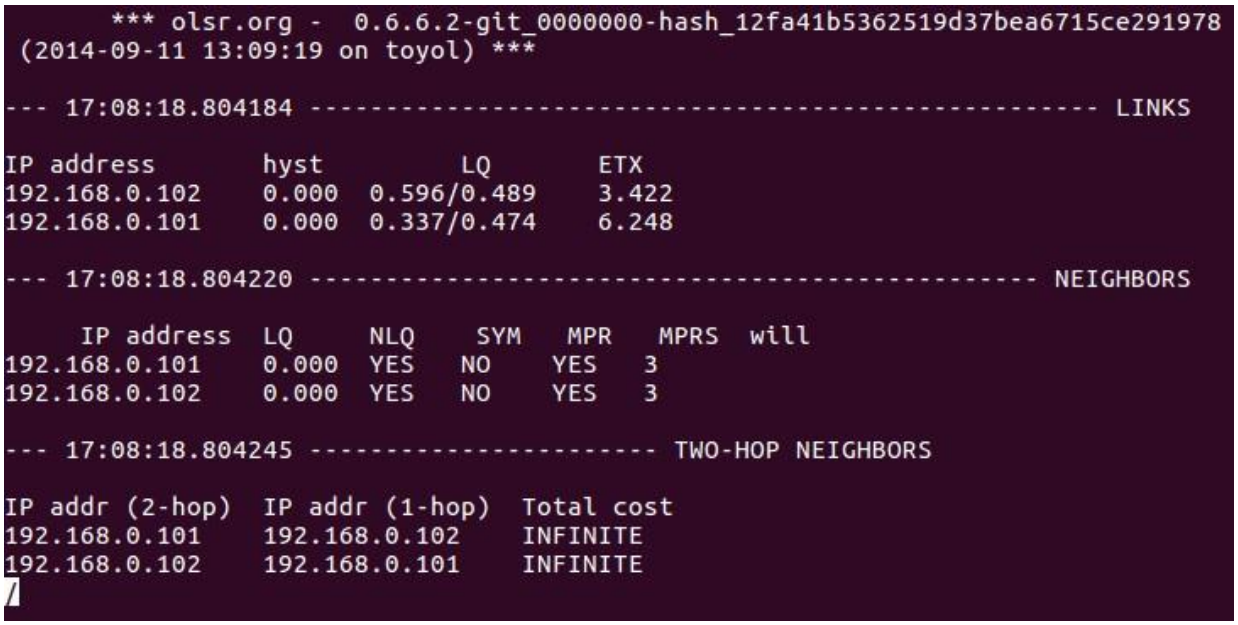

Figura 3.6: Estação G do esquemático mostrado na figura 3.4.

Figura 3.7: Estação E do esquemático mostrado na figura 3.4.

As figuras 3.5, 3.6, 3.7 mostram a rede Ad Hoc e o protocolo funcionando com plenitude, isso porque a configuração da figura 3.4 foi atendida com sucesso. Isso pode ser observado pela vizinhança de cada computador mostrado pelo protocolo e pelos saltos, ressaltando que ele só monta a tabela de conexão até dois saltos por só estarem três computadores na rede e existir a opção de no máximo dois saltos. Porém, se mais máquinas entrarem na rede a tabela vai aumentar mostrando todas as opções de saltos.

3.6 Implementação de multi-mestres ROS

O primeiro passo foi instalar o pacote *multimaster_fkie* do ROS, após isso foram feitas algumas mudanças em arquivos de configurações para que fosse habilitado o recurso *multicast*, que significa basicamente o envio de dados para múltiplos destinatários de forma simultânea.

O pacote permite um acesso remoto as outras máquinas e é através deste acesso que torna possível o mapeamento dos nós, além da possibilidade de assinar e publicar tópicos de outras máquinas. Não precisando fazer necessariamente um sistema onde um computador detém o sistema e as outros publicam nele, mas fazendo uma troca dinâmica entre todos da rede criando um sistema complexo.

Porém, para tornar toda essa comunicação possível, utilizou-se dos IPs determinados para adicionar ao arquivo de configuração, em cada máquina. E foram dados nomes aos IPs, para ao invés de o usuário receber as informações referentes à IPs, receber informações com o nome de cada máquina.

Após as alterações terem sido adicionadas ao arquivo de configuração, as máquinas então obtiveram a permissão para fazer o acesso remoto.

Posteriormente a instalação e configuração do pacote *multimaster_fkie* e criação e configuração da MANET, era necessário comprovar seu real funcionamento com uma aplicação. A aplicação proposta foi uma interface de voz para robôs que permite estabelecer um diálogo entre o humano e o robô.

3.7 Implementação da Interface de Voz para Robôs

A implementação e funcionamento da interface de voz serão explicadas com o auxílio da ferramenta *rqt_graph* do ROS. Esta ferramenta permite a visualização dos nós, tópicos, publicações e assinaturas do sistema em funcionamento.

A primeira etapa será explicar os nós da interface de voz , para em seguida mostrar suas conexões. Os nós e suas respectivas descrições:

- n__listener: a função deste nó é adquirir as informações de entrada, no caso é o que o humano insere na entrada de voz.
- fsm: este nó é o programa implementado utilizando a linguagem C, referente a máquina de estados finitos. Neste nó é definido a resposta do robô de acordo com a entrada recebida do tópico n__listener.
- talker: é o tópico que executa o programa de síntese de voz, ou seja, que fornece a saída de voz do robô em resposta a entrada.

A figura 3.8 apresenta a relação entre os principais nós e tópicos da interface de voz. As setas representam o fluxo de dados

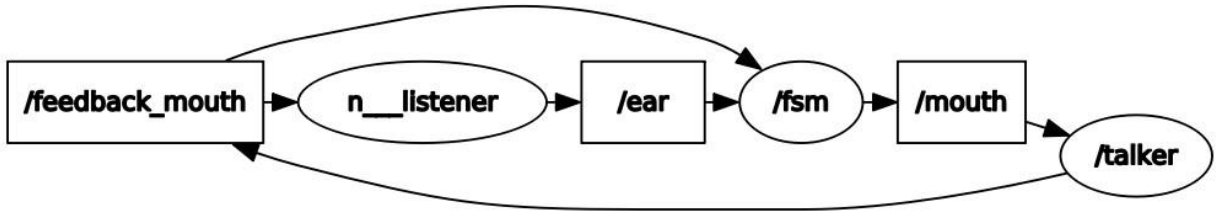

Figura 3.8: Esquemático mostrando nós e tópicos referentes a interface de voz para os robôs e suas associações.

A figura 3.9 apresenta o nó *n__listener*, mostrado em vermelho, o tópico que o nó assina, em azul, e o tópico no qual publica, em verde. O nó *n__listener* assina o tópico *feedback_mouth* para ter conhecimento de que a síntese de voz está sendo feita, porque enquanto o robô está falando ele não está ouvindo. Então, se houver alguma entrada de voz enquanto a síntese de voz está sendo feita, será ignorada. Publica no tópico *ear* a entrada do usuário.

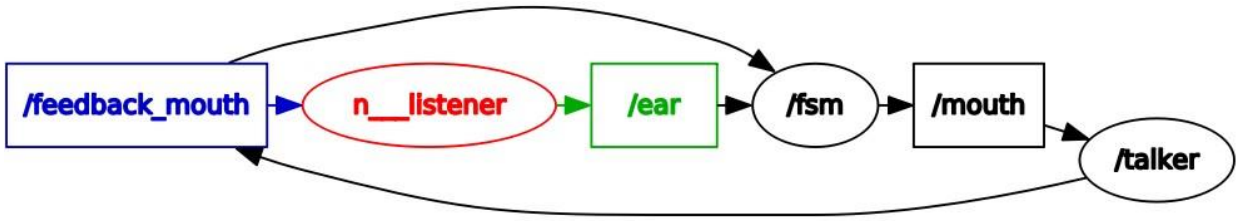

Figura 3.9: Esquemático destacando o nó ear, o tópico que assina e o tópico em que publica.

A figura 3.10 apresenta o nó *fsm*, mostrado em vermelho, os tópicos que o nó assina, em azul, e o tópico no qual publica, em verde. O nó *fsm* assina o tópico *feedback_mouth* para ter conhecimento de que a síntese de voz foi feita com sucesso para dar continuidade a máquina. Assina o tópico *ear* para ter acesso a entrada do usuário, o qual é fundamental para determinar o estado da máquina de estados.

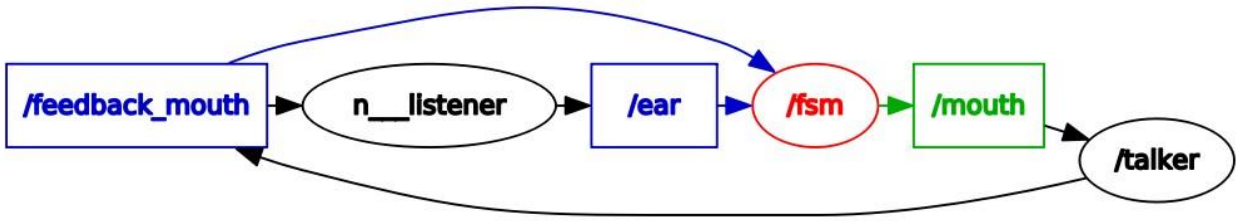

Figura 3.10: Esquemático destacando o nó fsm, os tópicos os quais ele assina e o tópico em que publica.

A figura 3.11 apresenta o nó *talker*, mostrado em vermelho, o tópico que o nó assina, em azul, e o tópico no qual publica, em verde. O nó *talker* assina o tópico *mouth* para obter as informações da síntese

de voz que será feita. Publica no tópico *feedback_mouth* as informação de que a síntese de voz está sendo feita e de que foi concluída.

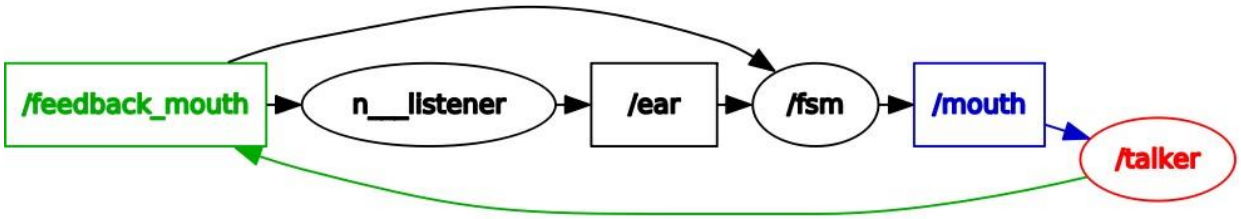

Figura 3.11: Esquemático destacando o nó talker, o tópico que assina e o tópico em que publica.

As figuras 3.8 ,3.9, 3.10 e 3.11 foram geradas através da ferramenta *rqt_graph* enquanto a interface de voz estava funcionando.

Capítulo 4

Resultados

4.1 Introdução

Esse capítulo apresenta o resultado de validação da implementação, mostrando que a camada de comunicação cooperativa funciona e de forma eficiente. Além de se comunicarem e poderem trocar informações da realização de suas tarefas os robôs também podem ajudar e interferir na tarefa do outro o que abre um leque imenso para futuras implementações.

4.2 Funcionamento da interconexão

O primeiro passo foi garantir o funcionamento da interface de voz, ou seja, fazer testes quanto ao reconhecimento de voz e a fala do robô e em seguida testar a máquina de estados. Importante relembrar que a máquina de estados que coordena o diálogo entre a máquina e o humano, pois ela é usada para definir as respostas do robô de acordo com a os dados de entrada da pessoa. Uma vez que o sistema estava funcionando foi compilado juntamente com o pacote *multimaster_fkie* do ROS, para isso primeiramente o mestre foi chamado em cada computador.

O funcionamento da interface de voz para os robôs foi mostrado juntamente com o funcionamento da implementação de redes ad hoc de robôs para o uso integrado ao sistema operacional de robôs.

Depois disso foi colocado o comando para que todos conseguissem mapear os mestres presentes na rede comum que no caso é a rede Ad Hoc. Nas figuras são mostradas as três máquinas mapeando os mestres vizinhos.

Na Figura 4.1 o computador nomeado de Aramis, detecta os mestres dos computadores nomeados como Jacqueline e Asimo. Na figura 4.2 o mestre do Asimo detecta os mestres da Jacqueline e do Aramis. E por fim, na figura 4.3 o mestre do Jacqueline detecta os mestres do Asimo e do Aramis.

```
aramis@aramis:~$ rosrun master_discovery_fkie master_discovery _mcast_group:=224
.0.0.1
[INFO] [1543358342.173903]: ROS Master URI: http://aramis:11311/
[INFO] [1543358342.217261]: Check the ROS Master[Hz]: 1
[INFO] [1543358342.218195]: Heart beat [Hz]: 0.02
[INFO] [1543358342.219185]: Active request after [sec]: 60
[INFO] [1543358342.220120]: Remove after [sec]: 300
[INFO] [1543358342.220854]: Robot hosts: []
[INFO] [1543358342.221603]: Approx. mininum avg. network load: 1.36 bytes/s
[INFO] [1543358342.264083]: Start RPC-XML Server at ('0.0.0.0', 11611)
[INFO] [1543358342.270231]: Subscribe to parameter `/roslaunch/uris`
[INFO] [1543358342.284541]: Create multicast socket at ('224.0.0.1', 11511)
[INFO] [1543358342.509575]: Detected master discovery: http://localhost:11611
[INFO] [1543358342.615115]: Added master with ROS_MASTER_URI=http://aramis:11311
/
[INFO] [1543358342.675745]: Detected master discovery: http://192.168.0.100:1161
1
[INFO] [1543358342.782536]: Added master with ROS_MASTER_URI=http://jacqueline:1
1311/
```

Figura 4.1: Computador Aramis detectando os mestres dos outros que foram nomeado de Jacqueline e Asimo.

```
asimo@asimo:~$ rosrun master_discovery_fkie master_discovery _mcast_group:=224.0.0.1
[INFO] [1543450336.528177]: ROS Master URI: http://localhost:11311
[INFO] [1543450336.544567]: Check the ROS Master[Hz]: 1
[INFO] [1543450336.544886]: Heart beat [Hz]: 0.02
[INFO] [1543450336.545101]: Active request after [sec]: 60
[INFO] [1543450336.545313]: Remove after [sec]: 300
[INFO] [1543450336.545527]: Robot hosts: []
[INFO] [1543450336.545741]: Approx. mininum avg. network load: 1.36 bytes/s
[INFO] [1543450336.569054]: Start RPC-XML Server at ('0.0.0.0', 11611)
[INFO] [1543450336.569753]: Subscribe to parameter `/roslaunch/uris`
[INFO] [1543450336.574729]: Create multicast socket at ('224.0.0.1', 11511)
[INFO] [1543450336.739983]: Detected master discovery: http://localhost:11611
[INFO] [1543450336.842578]: Added master with ROS_MASTER_URI=http://asimo:11311/
[INFO] [1543450337.690038]: Detected master discovery: http://192.168.0.101:11611
[INFO] [1543450337.724085]: Detected master discovery: http://192.168.0.102:11611
[INFO] [1543450337.795038]: Added master with ROS_MASTER_URI=http://jacqueline:11311/
[INFO] [1543450337.828922]: Added master with ROS_MASTER_URI=http://aramis:11311/
```

Figura 4.2: Computador Asimo detectando os mestres dos outros que foram nomeado de Jacqueline e Aramis.

```
jacqueline@jacqueline:~/catkin_ws$ rosrun master_discovery_fkie master_discovery _mcast_group:=224.0.0.1
[INFO] [1543450281.550759]: ROS Master URI: http://localhost:11311
[INFO] [1543450281.573432]: Check the ROS Master[Hz]: 1
[INFO] [1543450281.573861]: Heart beat [Hz]: 0.02
[INFO] [1543450281.574115]: Active request after [sec]: 60
[INFO] [1543450281.574415]: Remove after [sec]: 300
[INFO] [1543450281.574705]: Robot hosts: []
[INFO] [1543450281.574989]: Approx. mininum avg. network load: 1.36 bytes/s
[INFO] [1543450281.595633]: Start RPC-XML Server at ('0.0.0.0', 11611)
[INFO] [1543450281.596497]: Subscribe to parameter `/roslaunch/uris`
[INFO] [1543450281.603112]: Create multicast socket at ('224.0.0.1', 11511)
[INFO] [1543450281.769029]: Detected master discovery: http://localhost:11611
[INFO] [1543450281.872918]: Added master with ROS_MASTER_URI=http://jacqueline:11311/
[INFO] [1543450282.671416]: Detected master discovery: http://192.168.0.103:11611
[INFO] [1543450282.672822]: Detected master discovery: http://192.168.0.102:11611
[INFO] [1543450282.777277]: Added master with ROS_MASTER_URI=http://asimo:11311/
[INFO] [1543450282.781385]: Added master with ROS_MASTER_URI=http://aramis:11311/
```

Figura 4.3: Computador Jacqueline detectando os mestres dos outros que foram nomeado de Aramis e Asimo.

Através das Figuras 4.1, 4.2 e 4.3 é possível perceber que cada computador encontrou e adicionou a sua lista de mapeamento os mestres vizinhos da rede. E toda vez que um outro mestre se conectar a rede Ad Hoc criada, cada um vai detectar e adicionar a sua lista de mapeamento, porém para que isso seja possível um dos requisitos é que a nova máquina esteja na lista de *hosts* de cada computador.

Em seguida a interface de voz foi colocada para funcionar no computador nomeado como Jacqueline e uma vez funcionando seus tópicos foram mapeados nos outros computadores. Fazendo com que tudo que era feito na máquina Jacqueline era visto no computador Aramis e Asimo.

Isso já facilita muito em trabalhos em equipe, ter conhecimento do que o vizinho está fazendo e poder realizar alguma tarefa em conjunto ou se necessário substituí-lo. Porém, o objetivo final era que eles não só conseguissem ver o que o outro estava realizando como também pudesse interferir e fazer ou ajudar no trabalho. Para tornar isso possível era necessário que eles tivessem acesso para publicar no tópico desejado.

No caso colocado o tópico é o chamado *ear*, o qual é o utilizado para conversar com o robô e que aguarda o retorno do humano para responder, por isso que a assinatura é feita nele. Com os computadores sincronizados através do Asimo foi publicado no tópico *ear* e o Jacqueline respondeu com sucesso. A imagem 4.4 mostra a publicação no tópico.

```
asimo@asimo:~$ rostopic pub /ear std_msgs/String "data: 'TALK ABOUT AMORA'"
publishing and latching message. Press ctrl-C to terminate
asimo@asimo:~$ rostopic pub /ear std_msgs/String "data: MY NAME IS JACQUELINE'"
publishing and latching message. Press ctrl-C to terminate
```

Figura 4.4: Terminal Asimo publicando no tópico da interface de voz que está em andamento no computador Jacqueline.

Semelhantemente, foi feito no Aramis, utilizando o mestre dele para publicar no tópico *ear* que consta na máquina Jacqueline e mais uma vez obteve-se sucesso com a resposta do robô a interação.

```
^Caramis@aramis:~$ rostopic pub /ear std_msgs/String "data: 'TALK ABOUT AMORA'"
publishing and latching message. Press ctrl-C to terminate
aramis@aramis:~$ rostopic pub /ear std_msgs/String "data: 'HI ARAMIS'"
publishing and latching message. Press ctrl-C to terminate
```

Figura 4.5: Terminal Aramis publicando no tópico da interface de voz que está em andamento no computador Jacqueline.

Na Figura 4.6 observa-se no terminal do Asimo as publicações feitas nas Figuras 4.4 e 4.5, com o nome do respectivo robô. O que comprova que o acesso remoto foi feito com sucesso.

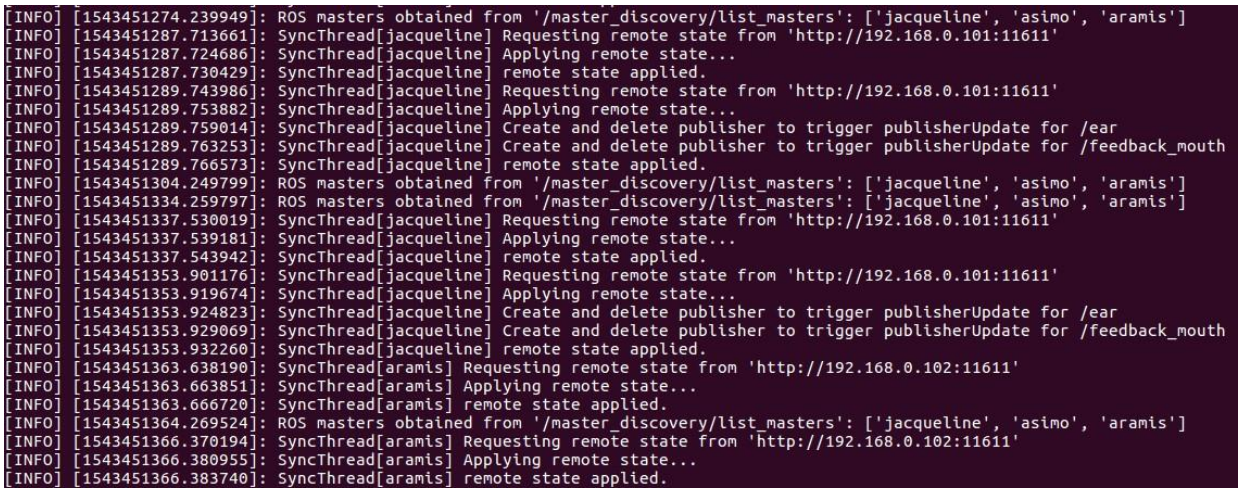

Figura 4.6: Imagem mostrando a comunicação entre os mestres de cada computador.

A Figura 4.7 mostra as mudanças de estados na máquina de estados a partir das iterações das três máquinas.

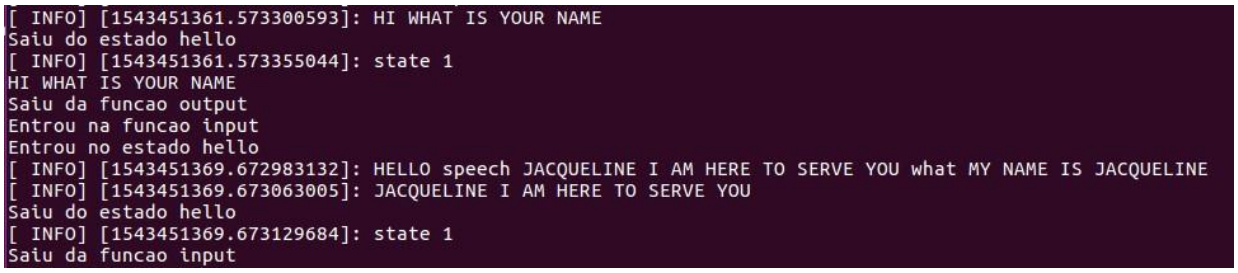

Figura 4.7: Imagem mostrando o terminal do computador Jacqueline respondendo aos comandos tanto da máquina Jacqueline quanto do Aramis e Asimo.

A Figura 4.8 mostra o pacote *multimaster_fkie* funcionando juntamente com a interface de voz para robôs. O pacote funciona separadamente da interface apenas interferindo quando alguma publicação é feita por parte de outro computador, o que não está ocorrendo na figura 4.8. O nós *master_discovery* publica as informações recebidas dos outros mestres no tópico *master_discovery/changes* e o nó *master_sync* assina esse nó para receber as informações e atualizações.

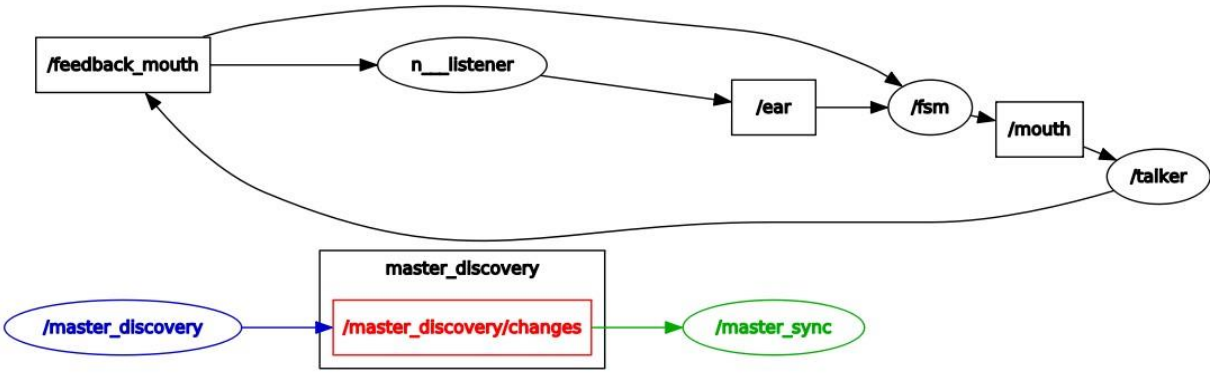

Figura 4.8: Nós, tópicos e serviços do sistema de forma completa.

A Figura 4.9 apresenta um nó que foi criado a partir de outro computador que publicou no nó *ear*. Ou seja, quando o comando de publicação no nó *ear* é enviado pelas outras máquinas, que não seja a principal que contém a interface de voz, como é mostrado nas figuras 4.4 e 4.5. É criado um nó na máquina Jacqueline que executa o comando e faz a publicação, no caso o nó *rostopic_6204_1544442726575*. Caso a entrada da máquina de estados venha da máquina Jacqueline a publicação será feita pelo nó *n__listener*.

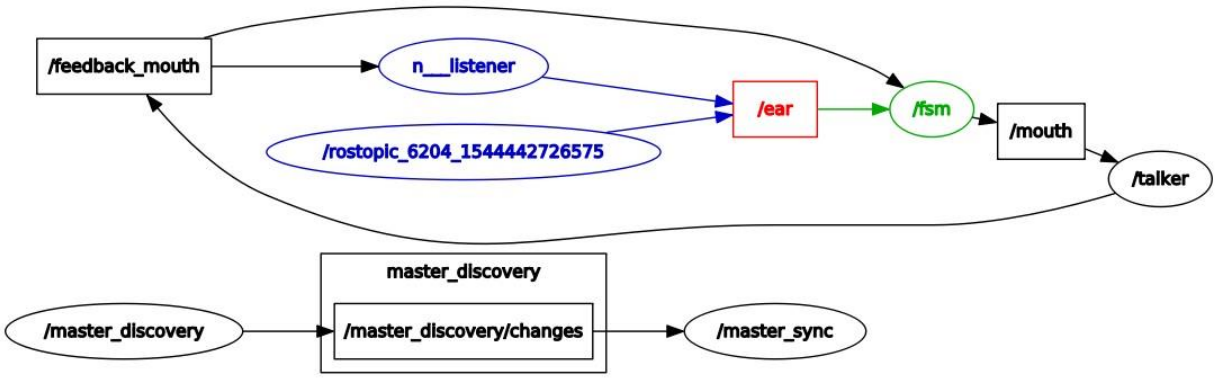

Figura 4.9: Mostra um outro computador fornecendo uma entrada para a interface de voz.

O que foi obtido como resultado foi uma infraestrutura para uma rede de robôs autônoma. Eficiente e independente que é gerado por cada terminal e que pode mapear os processos dos outros robôs presentes na rede, com o ROS em funcionamento. Além disso, podem interferir nesses processos, mesmo não possuindo a estrutura implementada no outro para realização destes. Isso pode ser explicado através da aplicação, pois o sistema foi montado e implementado apenas no terminal Jacqueline, nos outros dois apenas foi implementado o sistema para a rede Ad Hoc e pacotes ROS para a comunicação.

Conclusões

O trabalho proporcionou uma maior independência para os robôs, descentralizando ao máximo tanto a rede de comunicação quanto a parte de controle. Fazendo com que essa estrutura seja mais difícil de ser muito afetada, uma vez que seus processos ocorrem separadamente.

A rede MANET se mostrou eficiente e rápida, sem demonstrar atrasos de sincronização entre os computadores quando a comunicação foi estabelecida através dela. O protocolo de roteamento OLSR mostrou não ser necessário em uma rede em que há poucos nós. Isso foi comprovado porque o sistema como um todo foi testado com e sem a utilização do protocolo. E não foram notadas diferenças significativas nos resultados. A partir disso, concluiu-se que para uma aplicação dessa infraestrutura em que se tem poucos robôs não é necessário a utilização do protocolo de roteamento OLSR.

O ROS se mostrou uma ótima ferramenta para se trabalhar, muito flexível e fácil de ser utilizada.

O trabalho foi concluído com êxito uma vez que a infraestrutura para uma rede de robôs autônoma foi implementada com sucesso. Além disso, através dessa infraestrutura os robôs não só se comunicam entre si como também conseguem interferir no trabalho que o outro está realizando, sem a necessidade de ter todo o sistema do outro implementado nele. Tornando os trabalhos em grupo muito mais fáceis e possíveis de serem desenvolvidos.

Proposta de trabalhos futuros

As propostas de trabalhos futuros é aumentar a quantidades de robôs participantes dessa infraestrutura e checar sua eficiência. Testa-la com outros protocolos de roteamento e em áreas externas, adicionar obstáculos. Aumentar o fluxo de dados na MANET, colocar múltiplas interfaces voz para robôs juntos, por exemplo, para medir os atrasos.

FONTES

Fonte 1: https://www.b9.com.br/87344/california-autoriza-testes-de-carros-autonomos-em-viaspublicas/

Fonte 2: https://www.tecmundo.com.br/internet/2792-o-que-sao-redes-ad-hoc-.htm

Fonte 3: https://renatogbj.wordpress.com/2012/10/20/redes-de-sensores-sem-fio/

Fonte 4: https://paginas.fe.up.pt/~ee08059/olsr.php

Fonte 5: https://br.ccm.net/contents/285-o-que-e-o-protocolo-tcp-ip

Fonte 6: http://wiki.ros.org/ROS/Concepts

Fonte 7: http://wiki.ros.org/Master

Fonte 8: http://www.iri.upc.edu/files/scidoc/1607-Multi-master-ROS-systems.pdf

Fonte 9: https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html

Fonte 10: http://www.ros.org/about-ros/

Fonte 11: http://wiki.ros.org/multimaster_fkie

REFERÊNCIAS BIBLIOGRÁFICAS

[1] GERLA, Mario et al. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In: **Internet of Things (WF-IoT), 2014 IEEE World Forum on**. IEEE, 2014. p. 241-246.LEE, Eun-Kyu et al. Internet of Vehicles: From intelligent grid to autonomous cars and vehicular fogs. **International Journal of Distributed Sensor Networks**, v. 12, n. 9, p. 1550147716665500, 2016.DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping (SLAM): part I. *IEEE robotics & automation magazine*, IEEE, v. 13, n. 2, p. 99–110, 2006.

[2] LEE, Eun-Kyu et al. Internet of Vehicles: From intelligent grid to autonomous cars and vehicular fogs. **International Journal of Distributed Sensor Networks**, v. 12, n. 9, p. 1550147716665500, 2016.MATARIĆ, M. J. *The Robotics Primer*. Cambridge, Massachussetts: The MIT Press, 2007.

[3] ABOLHASAN, Mehran; WYSOCKI, Tadeusz; DUTKIEWICZ, Eryk. A review of routing protocols for mobile ad hoc networks. **Ad hoc networks**, v. 2, n. 1, p. 1-22, 2004.

[4] M. Mauve, J. Widmer, and H. Hartenstein, "A Survey on Position-Based Routing in Mobile Ad-Hoc Networks," IEEE Network Magazine, vol. 15, no. 6, pp. 30–39, November 2001.

[5] HOLLAND, Gavin; VAIDYA, Nitin. Analysis of TCP performance over mobile ad hoc networks. **Wireless Networks**, v. 8, n. 2-3, p. 275-288, 2002.

[6] PARZIALE, Lydia et al. **TCP/IP tutorial and technical overview**. IBM Redbooks, 2006.

[7] CLAUSEN, Thomas; JACQUET, Philippe. **Optimized link state routing protocol (OLSR)**. 2003.

[8] MACKER, Joseph. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. 1999.

[9] JACQUET, Philippe et al. Optimized link state routing protocol for ad hoc networks. In: **Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International**. IEEE, 2001. p. 62-68.

[10] QUIGLEY, Morgan et al. ROS: an open-source Robot Operating System. In: **ICRA workshop on open source software**. 2009. p. 5.

[11] YLONEN, Tatu; LONVICK, Chris. **The secure shell (SSH) protocol architecture**. 2005.

[12] KRAMER, J.; SCHEUTZ, M. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, v. 22, p. 101–132, 2007.

[13] ANDRE, Torsten; NEUHOLD, Daniel; BETTSTETTER, Christian. Coordinated multi-robot exploration: Out of the box packages for ROS. In: **Globecom Workshops (GC Wkshps), 2014**. IEEE, 2014. p. 1457-1462.

[14] JUAN, Sergi Hernandez; COTARELO, Fernando Herrero. Multi-master ros systems. **Institut de Robotics and Industrial Informatics**, 2015.

[15] HAX, Vinícius Alves et al. ROS as a middleware to Internet of Things. **Journal of Applied Computing Research**, v. 2, n. 2, p. 91-97, 2013.