PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO

GABRIEL MACEDO COUTO

GABRIEL POZZA MACHADO

Suporte à Formação de Enxame de Robôs

Porto Alegre, Brasil

17 de novembro de 2014

Gabriel Macedo Couto
Gabriel Pozza Machado

Suporte à Formação de Enxame de Robôs

Volume Final de Trabalho de Conclusão II apresentado como requisito parcial para a obtenção de título de Bacharel em Ciência da Computação pela Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul.

Pontifícia Universidade Católica do Rio Grande Do Sul
Faculdade de Informática
Ciência da Computação

Orientador: Edson Ifarraguirre Moreno

Porto Alegre, Brasil
17 de novembro de 2014

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
MCL	*Monte Carlo Localization*
ROS	*Robot Operating System*
SLAM	*Simultaneous localization and mapping*
TCP	*Transmission Control Protocol*

SUMÁRIO

1 INTRODUÇÃO

Não é novidade a existência de tecnologias que possibilitam a construção de robôs. Para muitas pessoas, quando você fala de robôs, elas pensam em máquinas muito grandes, como as utilizadas em linhas de montagem. Máquinas de tamanho volumoso, estacionárias e construídas para um propósito específico, capazes de concluir seus objetivos com muita precisão. Porém, essa visão está desatualizada. A humanidade já constrói e comercializa[1] robôs de diversos tamanhos que conseguem se mover em ambientes desconhecidos e cumprir simples ou complexas tarefas com sucesso. Um robô é todo e qualquer dispositivo mecânico criado pelo homem que pode se movimentar de maneira autônoma[2].

Sabemos que há muitas situações onde é necessário explorar ambientes desconhecidos. Considere a seguinte situação, onde um esquadrão militar precisa encontrar um caminho entre um ponto A e B que seja seguro para passagem, atravessar esse caminho, e surpreender o inimigo. No caminho direto entre esses dois pontos há um campo minado. O inimigo está perto, e controla posições de vantagem nos flancos. Logo, desviar do campo minado não é uma opção. A estratégia desse esquadrão é encontrar um caminho seguro através do campo minado antes que o inimigo perceba, para então dar continuidade ao que foi planejado e surpreender o inimigo. Sem conhecer a distribuição das minas terrestres, encontrar o caminho pode ser perigoso, se feito humanamente. Para realizar este mapeamento poderia ser utilizado um robô. O problema em se utilizar apenas um único robô, para esse exemplo, é que este não apresentará uma resolução em um tempo hábil, pelo fato de estar sozinho. Em uma situação crítica como essa, o tempo de resolução do problema é muito importante. O esquadrão militar não pretende ficar parado por muito tempo tão perto do inimigo.

Ao tentar resolver o mesmo problema do campo minado, ao utilizar dois robôs que não cooperam entre si para fazer a descoberta no ambiente, seria intuitivo dizer que o ganho de tempo no mapeamento seria desprezível. Isso acontece devido ao fato dos robôs estarem trabalhando individualmente. Os dois robôs iriam procurar o caminho de maneira separada e não trariam ganho de desempenho para a resolução do problema, ou seja, o tempo de resolução seria, em média, o mesmo que utilizar um robô apenas. O mesmo se repete se tentarmos resolver o mesmo problema com três robôs que não cooperam entre si. Assumimos, então, que esse problema exige, no mínimo, cooperação entre os robôs para resolver a tarefa em um tempo hábil. Seria lógico, também, pensar que existe uma curva onde, até certo ponto, quanto mais robôs trabalhando cooperativamente, menor o tempo para atingir a resolução do problema. Então, para resolver esse problema de maneira ótima, precisaremos de vários robôs que trabalham em conjunto. Mas como os

robôs iriam cooperar para resolver o problema?

A base de qualquer cooperação está na comunicação. A comunicação pode multiplicar as capacidades e aumentar a eficiência dos robôs[3]. Existem várias maneiras de comunicação, as quais são categorizadas de duas maneiras: implícita ou explícita. A comunicação implícita consiste na ausência da comunicação direta de informações, onde o ambiente é utilizado para passar essa informação, tal como vários animais utilizam através de feromônios em rastro. A comunicação explícita acontece quando a troca de informações é passada diretamente para os outros robôs, através de antenas, luzes ou gesticulação. Supondo que determinamos a maneira que os robôs iriam se comunicar, o próximo passo estaria na coordenação desses robôs.

Como idealizamos utilização de vários robôs pequenos e simples capazes de comunicar entre si, iremos chamar esse sistema multi-robótico de enxame de robôs. Enxame de robôs é uma abordagem no segmento da robótica para coordenar um grande número de robôs, inspirada em insetos que trabalham em cooperação[4]. Um enxame de robôs possui muito em comum com uma colônia de formigas ou um enxame de abelhas: Nenhum indivíduo no grupo é muito inteligente ou complexo, porém, juntos, conseguem resolver tarefas difíceis[5]. Em robótica de enxame, os robôs são relativamente simples e baratos. Em grupo, essas máquinas simples cooperam para resolver um problema que normalmente seria resolvido por outro robô muito mais complexo e caro. Uma vantagem da robótica de enxame é a tolerância a falhas, onde quando um dos robôs do grupo para de funcionar, os outros robôs continuarão a resolver o problema sem o robô perdido. Outra vantagem da robótica de enxame é a escalabilidade, devido ao baixo custo desses robôs, o tamanho do enxame pode ser aumentado ou diminuído conforme necessário.

A nossa solução hipotética para o problema do campo minado descrito antes, seria colocar grupos de robôs em uma formação que maximizasse a área mapeada pelo grupo. E com esse grupo em formação, iniciar o mapeamento de um caminho seguro para a travessia dos pontos A e B. Com a formação correta, seria possível não apenas aumentar a área mapeada, mas também aumentar largura do caminho considerado seguro, e por consequência a vazão de soldados para fazer a travessia.

Porém, essa solução é nada além de hipotética, devido ao fato de ser composta por suposições. De outro ângulo, ela é apenas uma vertente para outros problemas. Pois mesmo assumindo que os robôs consigam se movimentar de maneira autônoma e comunicar entre si, para fazer a formação dos robôs seria necessário que os robôs tenham alguma noção de localização. O problema de localização[6] é outro ponto crítico para que tudo isso funcione, pois os robôs não conhecem o ambiente. Eles precisam saber onde eles precisariam ir(relacionado a posição atual), para então poder entrar em formação.

Em um enxame de robôs, a comunicação precisa ser a menor possível[3], não apenas pela simplicidade das máquinas utilizadas, mas também porque a tendência de um enxame

de robôs é trabalhar com muitas máquinas, e fica difícil com que elas consigam completar suas tarefas enquanto precisam lidar com uma grande quantidade de comunicação. Podemos dizer então que a comunicação é outro ponto crítico para a formação do enxame.

O que pretendemos com esse o projeto é propor uma solução para suporte à formação de um enxame de robôs. Para conseguir isso, faremos uma utilização das soluções atuais para os problemas de comunicação e localização. Por fim, também iremos oferecer uma camada que encapsule a comunicação, localização e a solução, que dê suporte à formação do enxame. Como os recursos disponíveis para o nosso projeto são limitados, nosso foco é criar soluções através de um ambiente simulado, utilizando o software ROS.

O software *Robot Operating System* (ROS) é um framework para sistemas Unix, utilizado para facilitar o desenvolvimento de sistemas robóticos[7]. Nele será estabelecido uma arquitetura básica para robôs, os quais serão utilizados nas simulações do enxame. Nesta ferramenta iremos testar e validar diversos algoritmos para localização e métodos para resolver os problemas de comunicação, para então oferecer o suporte à formação.

1.1 OBJETIVOS

Esse projeto tem como alvo propor uma solução para suporte à formação de um enxame de robôs, em um ambiente simulado. Os objetivos estratégicos desse trabalho são: *(a)* ter domínio dos conceitos para modelagem de sistemas robóticos; *(b)* conhecer abordagens de comunicação para um enxame de robôs; *(c)* compreender os mecanismos para um robô se localizar em um ambiente desconhecido.

Os objetivos específicos desse projeto são derivados dos objetivos estratégicos: Para atingir o objetivo *(a)* será necessário fazer o levantamento de ferramentas para modelagem de sistemas robóticos; Para atingir o objetivo *(b)*, será necessário fazer um levantamento sobre abordagens de comunicação para enxames de robôs; Para atingir o objetivo *(c)* será necessário fazer um levantamento de algoritmos de localização móvel existentes para robôs, e compreender o funcionamento desses algoritmos.

1.2 ORGANIZAÇÃO DO DOCUMENTO

O restante do documento está organizado da seguinte forma: no Capítulo 2 abordaremos a fundamentação teórica, apontando alguns conceitos relacionados ao trabalho; no Capítulo 3 será apresentada, mais detalhadamente, a proposta do projeto em si; no Capítulo 4, apresentaremos uma série de testes realizados e os dados obtidos em cada simulação e, por fim, no Capítulo 5 está a conclusão do grupo sobre o projeto e demais declarações.

2 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo, exploram-se aspectos vinculados aos temas explorados neste trabalho, tais como: (i) robôs, (ii) algoritmos de localização e (iii) Robot Operating System. Na seção 2.1, é explorado a definição de robôs e seus conceitos básicos de movimentação e localização. Na seção 2.2 explora-se algoritmos diferentes que foram utilizados para determinar a localização de um robô em um ambiente. Na seção 2.3 é descrita a comunicação de processos através do ROS, seus utensílios que auxiliam em "localização e mapeamento simultâneos"(SLAM), como também navegação.

2.1 ROBÔS

Em 1979, *Robot Institute of America* definiu robô como sendo um manipulador reprogramável, multifuncional desenhado para mover materiais, partes, ferramentas, ou dispositivos especializados através de diversos movimentos programados para performar uma variedade de tarefas[8]. De maneira mais genérica, é uma máquina que desempenha uma função de maneira a substituir um agente vivo[9].

Robôs são especialmente desejáveis para certos tipos de trabalho porque ao contrário dos humanos: nunca ficam cansados; eles conseguem resistir condições físicas que são desconfortáveis ou perigosas; eles podem operar em ambientes sem ar; eles não se entediam com repetições; e eles não se distraem das tarefas que lhe são dadas. Os robôs da atualidade possuem sistemas sensoriais avançados que processam informações e aparentam funcionar como se tivessem um cérebro. O que acontece, na verdade, é que o "cérebro" é alguma forma de inteligência artificial através de um programa de computador que roda dentro do robô. Os sensores permitem que o programa que roda no robô observe as condições do ambiente à sua volta, e então decidir um curso de ações a tomar, baseado nessas condições.

Um robô pode incluir diversos componentes: (i) efetores: braços, pernas, mãos, pés, rodas, etc; (ii) sensores: dispositivos que fazem algum tipo de leitura do ambiente, tal como câmeras, lasers, detectores de luz, calor, etc; (iii) computador: o cérebro que contém os algoritmos que controlam os robôs; (iv) equipamento: ferramentas que podem auxiliar o robô em sua tarefa. As características que diferem um robô de um maquinário comum são que: robôs normalmente funcionam por conta própria, são sensíveis ao ambiente, adaptam-se as variações do ambiente ou aos erros de sua performance e são orientados às tarefas.

De acordo com o propósito desse trabalho, vamos nos focar nos conceitos necessários para movimentação e localização dos robôs. Para conseguir a efetiva movimentação de

robôs, há dois tipos de informação necessários: perceptual e odométrica[10]. A informação perceptual é o que o robô enxerga. Essa informação pode ser os dados de medida de distância, por um sensor laser, por exemplo. A informação odométrica são dados que informam o quanto e para onde o robô se movimentou desde que foi ativado. Isso pode ser alimentado com informações de controle do motor, ou giroscópios e acelerômetros. Ambas informações juntas podem ser utilizadas para que o robô conheça o ambiente a sua volta, saiba por onde ele andou nesse ambiente, e com isso, montar um mapa que auxilie na sua navegação.

2.2 ALGORITMOS DE LOCALIZAÇÃO

Resolver o problema da localização do robô é um ponto-chave para o desenvolvimento desse trabalho, já que é essencial para a formação. Existem diversas abordagens para se obter uma localização no ambiente, tais como: *(a) Monte Carlo Localization; (b) Markov Localization; (c) Kalman Filter.*

Os algoritmos *(a)* e *(c)* são implementações baseadas no conceito de *filtro de Bayes*, que consiste em um conceito abstrato que utiliza probabilidade para estimar estados, de maneira recursiva. A ideia por trás do filtro de Bayes consiste em resolver o problema de estimar um estado x a partir de dados obtidos através da medição de sensores[11].

2.2.1 MONTE CARLO LOCALIZATION

Filtro de partículas que realiza uma estimativa de todos os lugares em que um robô pode estar em um determinado momento. Dado o mapa do ambiente do robô, a medida em que este se desloca, as partículas se agrupam nas posições que possivelmente ele pode estar. Através de dados sensoriais obtidos pelo robô, é possível formar este conjunto de partículas[12].

Por exemplo, vamos supor que em um determinado ambiente existe um robô no qual não se conhece sua posição atual. Este robô se desloca alguns metros e reconhece, através de sensores, que ao seu lado existe uma porta. Neste estado do tempo, todos os lugares do mapa em que houverem portas haverá uma concentração maior de partículas do que em outros lugares. Isto pode ser chamado de *crença*, porque representa os lugares nos quais o robô acredita estar[10]. A medida em que o robô se desloca pelo ambiente, os conjuntos de partículas se unificam e a crença da sua posição aumenta, até o momento em que se obtém um único conjunto de partículas em determinado ponto do mapa, apontando assim sua verdadeira localização atual.

2.2.2 MARKOV LOCALIZATION

A localização por *Markov* é um algoritmo probabilístico que possuí características semelhantes ao algoritmo MCL. A diferença está na representação das probabilidades. Enquanto MCL representa suas estimativas através de um conjunto de partículas, este algoritmo mantém uma distribuição probabilística de todos os possíveis lugares nos quais um robô acredita estar[13].

2.2.3 KALMAN FILTER

Algoritmo recursivo que estima o estado em algum momento do tempo e obtém feedback, através da medição de ruído. Este algoritmo é composto por dois processos[14]: *a)* atualização de tempo, onde é realizada uma estimativa sobre o próximo estado; *b)* atualização da medição, onde a estimativa realizada é corrigida e as informações sobre o estado são atualizadas.

2.3 ROBOT OPERATING SYSTEM

A principal função do ROS é servir como um *middleware* de comunicação entre diversos processos essenciais para o funcionamento do robô. O ROS oferece uma aplicação onde processos podem publicar informações em tópicos, ou, também, se inscrever em outros tópicos para ler as publicações colocadas lá por outros processos. Assim, diversos processos podem comunicar entre si, e executar suas diferentes tarefas para fazer com que o sistema como um todo atinja um mesmo objetivo.

2.3.1 ESTRUTURAS BÁSICAS

A comunicação no ROS é feita, basicamente, utilizando conceito semelhante aos grafos. Os processos que utilizam o ROS para se comunicar com os outros são chamados de *Nodos*. Cada nodo pode se inscrever ou publicar em *Tópicos* diferentes, definindo o sentido do vértice, que indica o sentido do fluxo de mensagens. Cada tópico possui a definição das *Mensagens* daquele tópico, que servem para definir o tipo dos campos utilizados para publicação. Essa estrutura foi feita para funcionar unidirecionalmente, onde um ou mais processos publicam seus dados, e outros processos interessados podem consumir esses dados à vontade.

Esse é um processo assíncrono que funciona principalmente sobre TCP/IP (camada de transporte TCPROS), ou até uma opção UDP (camada de transporte UDPROS), porém essa última só está disponível para processos em C++. A camada de transporte UDP é de baixa latência e suscetível à perdas. Os nodos são obrigados a implementar

TCPROS, e caso elas implementem outras camadas, negociam qual camada a ser utilizada em runtime. Por ser assíncrono, essa metodologia de comunicação possui suas limitações.

O ROS fornece, também, uma estrutura chamada *Serviços*, que é, simplificadamente, uma API para fazer *Remote Procedure Calls* (RPC), que permitem que processos façam requisições e esperem pela resposta. Essa é uma forma de comunicação que é utilizada para mensagens críticas, pois permite conexão permanente entre dois processos, fornecendo resultados com uma performance superior.

Adicionalmente, o ROS também possui um *Servidor de Parâmetros*, que permite que alguns processos guardem configurações que podem ser vistas e modificadas globalmente, caso seja necessário. Sendo assim, é possível alterar algumas configurações facilmente através desse servidor, e às vezes, um processo pode até ajustar as configurações de outro processo.

O processo mestre do ROS provê o serviço de registro e consulta de nomes e endereços para as trocas de mensagens do ROS. Esse processo precisa estar rodando para que todos os sistemas baseados o ROS consigam se encontrar e comunicar.

A organização dos softwares que interagem através do ROS é feita através de Pacotes. Cada pacote pode conter bibliotecas, nodos, *datasets*, arquivos de configurações ou qualquer coisa que seja útil empacotado junto O ROS oferece, também, um repositório onde a comunidade pode disponibilizar pacotes livremente, e até consultar e fazer download pacotes onde determinada função já está implementada por outra pessoa.

2.3.2 SLAM E NAVEGAÇÃO

O ROS, através de seu repositório de pacotes, oferece diversas implementações de algoritmos que podem agilizar no desenvolvimento de aplicações para robôs. Tendo um modelo robótico funcional para o ROS, é possível utilizar esses pacotes prontos, de maneira que o robô poderá usufruir de localização, mapeamento e navegação com pouquíssima implementação.

Assumindo que o robô publica seus dados de odometria e percepção em seus respectivos tópicos, e que o robô controla seus motores de acordo com dados publicados em um tópico para movimentação. Então é possível utilizar um pacote disponível de SLAM, junto de arquivos de ajuste fino do processo para que faça uso desses tópicos corretamente. O processo que controla o SLAM irá fazer os cálculos necessários e publicar seus resultados(tais como mapa, mapa de custos e localização do robô) em outros tópicos, também configurados nos arquivos de ajuste fino.

Adicionalmente, pode ser utilizado outro pacote integrado ao SLAM. Existem pacotes responsáveis pela navegação autônoma de um robô. Basta que no ajuste fino do pacote de navegação, seja configurado para ele ler os tópicos alimentados pelo processo

que calcula o SLAM. Esse pacote de navegação deve possuir um tópico de *goal* (objetivo), onde uma aplicação de terceiro pode publicar uma posição(coordenada), que esse processo de navegação irá calcular o melhor caminho(caso exista) para o robô chegar a essa posição. Em seguida, o processo de navegação irá publicar no tópico de controle de movimentos do robô as informações necessárias para atingir esse percurso. Ao mesmo tempo em que o robô segue seu percurso, os sensores e os outros processos estão sempre gerando dados novos que podem alimentar os mapas ou ocasionar mudanças de percurso.

2.3.3 SIMULAÇÃO

Como esse trabalho se foca no uso de enxame de robôs na tentativa de se obter uma formação, seria importante que houvesse muitos robôs à disposição. Entretanto, esse não é o caso. Para conseguir fazer os testes necessários no desenvolvimento desse trabalho, tudo será feito, principalmente, em ambientes simulados. O ROS, quando instalado por completo, é integrado a alguns simuladores. Dentre eles, o simulador mais completo, mais utilizado, e melhor bem integrado ao ROS, é o Gazebo.

O Gazebo fornece simulação em tempo real de um mundo com modelos robóticos bem completos que são afetados pela física. O Gazebo é responsável por cálculos de atrito, colisões, e basicamente tudo que envolve a simulação da situação real. Para utilizar algum robô simulado com o Gazebo, é necessário ter um modelo robótico e processos que fazem a ponte entre o ROS e o Gazebo.

O modelo robótico é um arquivo, normalmente no formato SDF(Simulator Description Format), mas também aceita um formato antigo chamado URDF(Universal Robotic Description Format). Esse modelo robótico possui as informações de dimensões, peso, coeficientes de atrito, juntas, sensores, e todas as outras características físicas do robô. Através desse arquivo que o simulador fará os cálculos de como esse robô se comportará no mundo.

Os processos que fazem a ponte entre o ROS e o Gazebo são processos que se inscrevem ou publicam em certos tópicos do ROS, de maneira a servir de ponte entre os ambientes. Para controlar o motor de um robô que está sendo simulado no Gazebo, pode ser criado um processo que se inscreve em tópicos de velocidade para cada motor, e a cada mensagem lida sobre a velocidade do motor, esse processo irá aplicar as forças necessárias nas juntas do robô no Gazebo. Essas forças de rotação são recebidas pelo Gazebo, se traduzindo no movimento do robô. Caso o modelo robótico possua sensores, também seria necessário ter outro processo que publiquem em tópicos do ROS as informações de sensores que estão sendo geradas pelo simulador Gazebo.

3 DESCRIÇÃO DO PROJETO

Neste capítulo, descreveremos o projeto implementado, detalhando a sua arquitetura e as funcionalidades alcançadas. Como previamente apontado, o projeto teve por finalidade propor uma solução para suporte à formação de um enxame de robôs, em um ambiente simulado.

Desta maneira, e com as fundamentações apresentadas previamente, o projeto está composto principalmente de quatro elementos, quais sejam: (i) um modelo robótico, (ii) um modelo de mundo para simulação, (iii) SLAM e navegação com o modelo robótico, (iv) implementação de um processo que permita que os robôs simulados entrem em formação. O primeiro serve de base para construir os outros componentes do projeto, por isso foi escolhido o modelo Pioneer-3AT, por razões detalhadas na Seção 3.2. O segundo é um arquivo no formato SDF que descreve um ambiente, com paredes, portas, e até os robôs utilizados na simulação, também detalhado na Seção 3.2. O terceiro é composto diversos arquivos, que determinam o comportamento e tópicos pelos quais os pacotes de SLAM e Navegação irão trabalhar, para cada robô, detalhados na Seção 3.3. Finalmente, o quarto é a maior implementação do projeto, detalhada na Seção 3.4.

A Figura 1 demonstra a disposição em camadas da arquitetura de diversos componentes que serão utilizados no projeto. Consideramos que a camada mais baixa é a primeira, e a camada mais alta é a última. O primeira camada da imagem é o Gazebo, que servirá de simulador para os testes. O segunda camada da arquitetura é o ROS, que serve como centro de comunicação entre os diversos processos utilizados no projeto. A camada que está entre a primeira e a segunda, é “Ponte Pioneer-3AT” é um processo descrito na Seção 3.2, que é responsável por integrar os dados do robô entre o ROS e o Gazebo. Note que SLAM, Navegação e Formação são camadas que estão no mesmo nível, e que todas elas são dependências do ROS para comunicar entre si.

Figura 1 – Arquitetura do projeto.

O elemento "Formação" da última camada da Figura 1, representa o processo que este trabalho se comprometeu em implementar, no qual o usuário poderá especificar uma formação, e que dado um número de robôs, e seus respectivos tópicos de *goal*(objetivo), o nosso processo trabalhará para levar os robôs até as diversas localizações desejadas até que a formação escolhida seja reconhecida.

Nas próximas seções deste capítulo, descreveremos detalhadamente os componentes que compõem o projeto, as suas respectivas características, funcionalidades e a relação entre esses componentes, conforme apresentado na Figura 1.

3.1 INSTALAÇÃO DO AMBIENTE

3.1.1 PREPARAÇÃO DOS REPOSITÓRIOS

Para fazer a instalação do ROS, supomos que o ambiente já possui a versão 14.04 do Ubuntu previamente instalado. É possível instalar o ROS em outras versões do Ubuntu[referencia necessaria], porém, não iremos cobrir o mesmo para simplificar as instruções de instalação do ambiente. Para iniciar a instalação é necessário adicionar primeiro o repositório do ROS na lista de fontes de software do sistema. Para fazer isso, é necessário utilizar o seguinte comando:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
 trusty main" > /etc/apt/sources.list.d/ros-latest.list'
```

Após acrescentar os fontes do ROS, é necessário ajustar as chaves:

```
$ wget https://raw.githubusercontent.com/ros/
 rosdistro/master/ros.key -O - | sudo apt-key add -
```

Para fazer a instalação do Gazebo, também é necessário preparar os repositórios. Os comandos são semelhantes ao do ROS, porém, apontando para os endereços correspondentes ao software:

```
$ sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu
 trusty main" > /etc/apt/sources.list.d/gazebo-latest.list'
```

Após acrescentar os fontes do ROS, é necessário ajustar as chaves:

```
$ wget http://packages.osrfoundation.org/gazebo.key -O - |
 sudo apt-key add -
```

3.1.2 INSTALAÇÃO DOS PACOTES

Antes de iniciar a instalação, é preciso atualizar a lista local do repositório de softwares:

```
$ sudo apt-get update
```

Para realizar a instalação do ROS, vamos fazer a instalação completa. Dado que neste trabalho são utilizados vários pacotes que estão disponíveis automaticamente na versão completa do software:

```
$ sudo apt-get install ros-indigo-ros-base
 ros-indigo-slam-gmapping ros-indigo-navigation
 ros-indigo-robot-state-publisher ros-indigo-rqt-common-plugins
 ros-indigo-rqt-capabilities ros-indigo-rqt-controller-manager
 ros-indigo-rqt-robot-steering ros-indigo-rviz
 ros-indigo-xacro ros-indigo-yujin-ocs
 ros-indigo-gazebo4-ros-pkgs git
```

Este último comando irá levar algum tempo até ser concluído, dado que o ROS consiste de um grande conjunto de softwares, que são os pacotes do ROS com o prefixo *ros-indigo*, que indicam a versão do ROS para qual foram desenvolvidos: o pacote ros-base é a unidade básica do ROS, contendo os componentes essenciais para seu funcionamento; o pacote slam-gmapping é uma implementação do [algoritmo] para o ROS[citação]; *navigation* é um pacote com várias ferramentas básicas de navegação, o que inclui o *move_base*, que será responsável pela navegação dos robôs no ambiente, e também o *amcl*, implementação do *Monte Carlo Localization*; robot-state-publisher é um pacote que realiza a leitura do estado de cada junta do robô, e publica sua posição atual no tópico geral de posições relativas chamado tf; rqt-common, rqt-capabilities, rqt-controller-manager e rqt-robot-steering são um conjunto de pacotes que permitem criar interfaces gráficas simples para servirem de entrada de dados, através de tópicos, no ROS(tal como uma interface para controlar a velocidade do robô, nesse caso); rviz é um pacote que inclui o software rviz, o qual serve para visualizar dados graficamente; xacro é um processador de macros para XML, utilizado para construir definições de robôs extensas, em XML, através de um arquivo "XML" simplificado com macros; do yujin-ocs o que nos interessa é o cmd-vel-mux, um multiplexador de tópicos de velocidade que permite que diversas aplicações interajam com o mesmo robô, com suas devidas prioridades; gazebo4-ros-pkgs é um conjunto de pacotes que instala o gazebo4 em conjunto com pacotes integradores do ROS.

Após a instalação de todas as dependências do projeto, é necessário colocar o ROS e o Gazebo no PATH de execução do ambiente.

Para colocar o ROS no PATH de execução do bash:

```
$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
```

Para configurar o Gazebo, basta coloca-lo no PATH de execução:

```
$ echo "source /usr/share/gazebo/setup.sh" >> ~/.bashrc
```

Por último, é necessário evetivar as modificações no PATH:

```
$ source ~/.bashrc
```

3.1.3 PREPARANDO PARA A EXECUÇÃO

Antes de seguir com a execução do projeto, é necessário preparar o sistema de pendências do ROS e configurar o espaço de trabalho. Para iniciar o sistema de dependências do ROS:

```
$ sudo rosdep init

$ rosdep update
```

O ROS utiliza um software chamado catkin para gerenciar o ambiente de trabalho e os pacotes em desenvolvimento. Para iniciar o desenvolvimento, precisamos iniciar esse ambiente, de maneira a acomodar os pacotes nos quais estamos trabalhando; Para criar um espaço de trabalho no catkin, utilize os seguintes comandos no shell:

```
$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/src

$ catkin_init_workspace

$ cd ..

$ catkin_make

$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc

$ source ~/.bashrc
```

3.1.4 INSTALAÇÃO DO PACOTE *ROS-PIONEER-3AT* E *SUPORTEFORMACAO* NO ROS

Para fazer a instalação do modelo Pioneer-3AT no ROS, vamos utilizar a versão open-source do modelo. Primeiro, vamos entrar na pasta do espaço de trabalho e então baixar o pacote do Pioneer-3AT.

```
$ cd ~/catkin_ws/src

$ git clone https://github.com/dawonn/ros-pioneer3at.git
```

Para fazer a instalação do pacote *suporteformacao*, que contém todas as configurações descritas aqui nesse trabalho, no ROS, basta apenas ter a pasta suporteformacao no espaço de trabalho também:

```
$ cd ~/catkin_ws/src

$ git clone https://github.com/gmcouto/suporteformacao.git
```

Por último, é necessário compilar ambos pacotes para poder utilizar o projeto:

```
$ cd ~/catkin_ws/

$ catkin_make
```

3.2 PIONEER-3AT E SIMULAÇÃO NO GAZEBO

Para o propósito desse trabalho, o modelo robótico escolhido é o Pioneer-3AT, que é um robô multipropósito. Ele serve para funcionar em lugares abertos secos, e em lugares fechados. É um robô de 4 rodas e 4 motores em que pode ser acoplado diversos tipos de sensores[15]. Um dos fatores mais importantes que contribuiu para a escolha deste modelo é que o ROS pode ser facilmente instalado no computador interno, que há na versão completa do robô. Também é interessante para o projeto que o laboratório de robótica da Faculdade de Informática da PUCRS também possui uma unidade deste robô, o que acrescenta uma possibilidade de fazer testes reais com o trabalho aqui apresentado.

O Pioneer-3AT já é um modelo bastante utilizado em estudos que envolvem mapeamento, navegação, monitoramento, ressonância, visão, manipulação e muitos outros comportamentos[15]. Devido a popularidade desse robô, foi possível ter acesso a um projeto de código aberto que possui diversos ativos que podem contribuir para que esse trabalho possa ser concluído em tempo hábil, dentre os quais são: (i) um processo ponte Gazebo-ROS; (ii) modelo robótico pronto, no formato SDF; (iii) arquivos de configuração para SLAM e navegação; (iv) scripts de exemplo prontos, incluindo uso de scanner de distância laser Hokuyo, como pode ser visto na Figura 2.

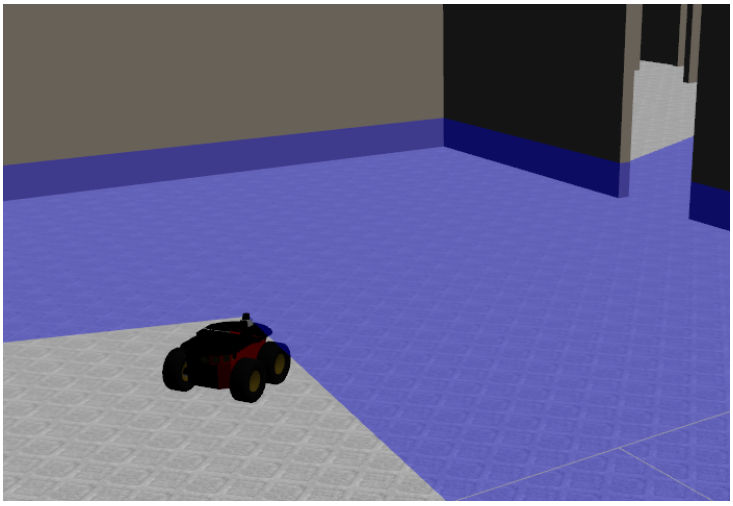

Figura 2 – Pioneer-3AT com sensor laser Hokuyo no Gazebo.

O mundo em que a simulação será realizada consiste em um cenário vazio composto por portas, janelas e paredes, como pode ser visto na Figura 3. Este ambiente foi criado devido ao fato do projeto possuir a liberdade de abstrair a complexidade de fabricar um mundo próprio para o Gazebo. O que é necessário fazer, para adaptar o mundo aos nossos casos de uso, é apenas ajustar no arquivo SDF a localização de cada robô simulado, para uma sala do mundo a qual seja propícia para os testes.

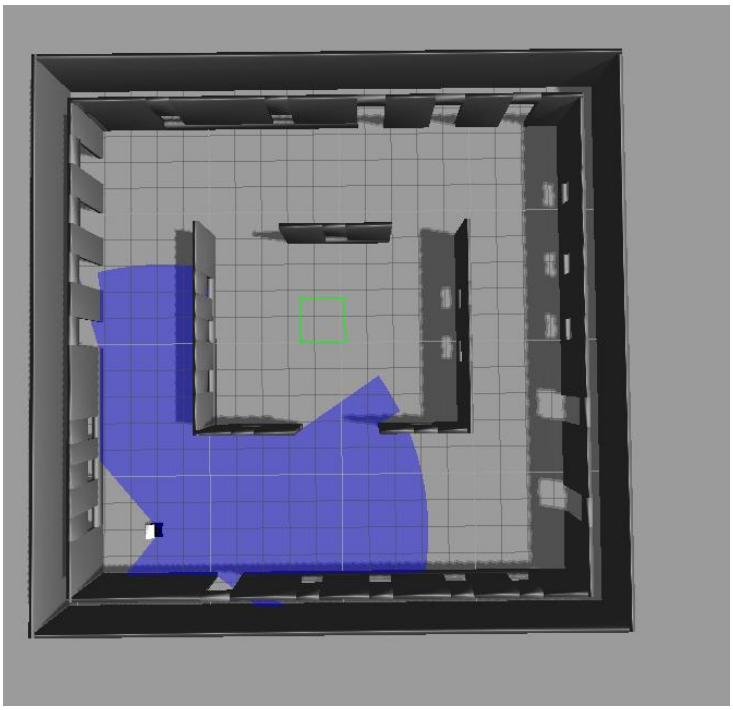

Figura 3 – Cenário onde os testes serão realizados.

3.3 SLAM E NAVEGAÇÃO

É possível resolver o problema de SLAM de muitas maneiras. As abordagens mais populares consistem na utilização de MCL ou *Kalman Filter* e suas variações. A abordagem por MCL pode ser utilizada através do pacote *amcl* do ROS, na qual apresenta um mapa do ambiente pronto. *Kalman Filter* é uma solução eficiente para SLAM[16] e pode ser utilizada através do pacote *robot_pose_ekf* , para estimar a posição do robô.

Durante os testes feitos, foi utilizado o pacote *slam_gmapping* para SLAM, e para navegação foi utilizado o *move_base*. Ambos foram escolhidos porque são parcialmente implementados pelo pacote open-source do Pioneer-3AT. O pacote *move_base* é o principal pacote da pilha de navegação oficial do ROS. Enquanto o *slam_gmapping* é apenas uma das opções de localização e mapeamento do ROS, possui grande vantagem em ser um pacote que une o mapeamento e localização em um só. Como pode ser visto na Figura 4, o Pioneer-3AT consegue mapear um ambiente desconhecido, e desviar de obstáculos de forma inteligente quando se usa o pacote *move_base*.

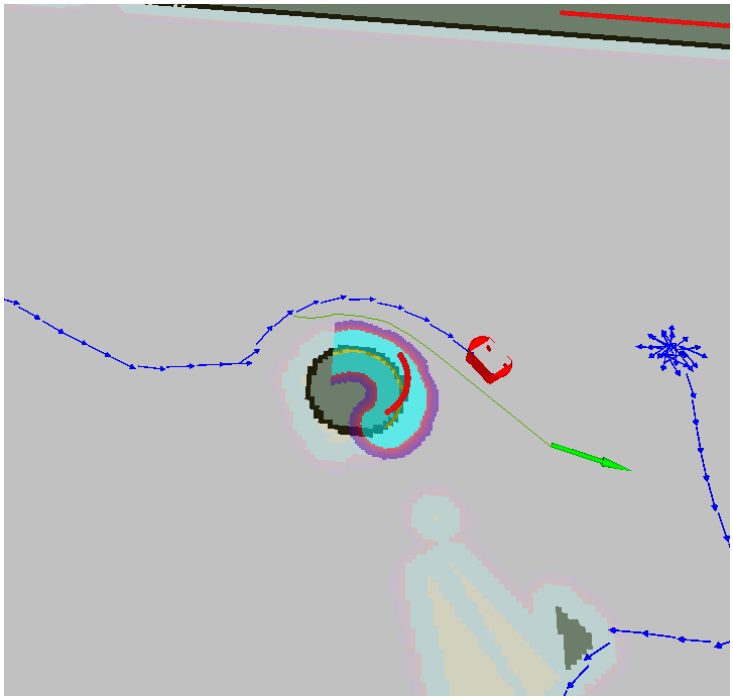

Figura 4 – "Visão" do Pioneer-3AT com *slam_gmapping*, enquanto desvia de obstáculo para chegar ao objetivo.

Existem muitos arquivos de configuração de ambos pacotes, nos quais precisam ser estudados caso exista a necessidade de mudar o comportamento do SLAM ou da Navegação. Para replicar o procedimento para múltiplos robôs, seria necessário ter cópias de arquivos de configuração para cada robô, porque nesses arquivos também estão dispostos os tópicos utilizados por esses processos, que são individuais de cada robô.

3.4 IMPLEMENTAÇÃO

Assumindo que estamos executando uma simulação na qual todos os robôs sabem sua localização atual, e que cada robô está configurado adequadamente com um pacote de navegação. Além de existir um caminho possível entre os robôs, o elemento que falta para obter a formação dos robôs é apenas um comando, para cada robô, que ajusta o objetivo como sendo exatamente a respectiva posição da formação desejada.

O foco deste projeto consistiu em como conseguir montar uma simulação, atingir a condição adequada para formação e implementar o software que emita os comandos finais para obter a formação. Porém, para conseguir gerar os comandos de formação, foi necessário: (i) saber a formação utilizada, (ii) atribuir cada robô para alguma coordenada de acordo com a formação, (iii) converter a coordenada da formação para as coordenadas do mapa, (iv) e o caminho dos tópicos de *goal* de cada um dos robôs. Tendo todos esses dados em mãos, é possível enviar para cada robô, a posição correta na qual deseja-se que eles estejam, em uma ordem que eles não se tornem obstáculo para os próximos robôs chamados. O pacote de navegação irá fazer o *path panning* necessário para que a formação seja atingida.

O entregável desse projeto consiste primeiramente em um pacote para o ROS, contendo os processos necessários para resolver os problemas que foram propostos a serem resolvidos por este projeto. Este pacote contém inclusive os arquivos de configuração básicos para o robô Pioneer-3AT, de maneira que os nossos testes possam ser reproduzidos com o menor esforço possível. Adicionalmente, foi construída uma ferramenta que automatiza a réplica dos arquivos de configuração para múltiplos robôs.

3.4.1 BREVE ROTEIRO DE DESENVOLVIMENTO

O pacote *suporteformacao* disponibilizado por esse trabalho é o foco do desenvolvimento feito nesse trabalho. Nessa seção será ilustrado o processo usado para replicar nossos arquivos.

O primeiro passo é a criação do pacote no catkin. Para criar o pacote, entre na pasta `src` do seu espaço de trabalho e digite:

```
$ catkin_create_pkg suporteformacao
```

Perceba que a pasta suporteformacao foi criada, dentro dela há o arquivo package.xml e o arquivo CMakeLists.txt. O arquivo package.xml pode conter o nome, descrição, autores e dependências do pacote. Já o arquivo CMakeLists descreve as bibliotecas que devem ser usada na compilação do pacote. Ambos arquivos já possuem uma descrição detalhada sobre como completar os arquivos com as informações necessárias, por isso, não

iremos nos aprofundar nesse nível de detalhe.

O segundo passo para o desenvolvimento do pacote foi a criação do mundo no gazebo. Para criar o mundo no gazebo, primeiro é necessário criar um ambiente para os robôs navegarem, basta abrir a interface do gazebo, ir no menu *Edit*, selecionar o item *Building Editor*. O Gazebo possui uma interface bastante intuitiva para desenvolver "construções", podendo adicionar paredes e portas através dos botões *Add Wall* e *Add Door*. Por isso não vamos cobrir nesse documento os detalhes para construir um ambiente. Quando concluído, é possível salvar o modelo da construção no formato SDF. Um formato derivado do XML criado para descrever ambientes e objetos para simuladores robóticos, visualizadores e sistemas de controle. Originalmente desenvolvido como parte do Gazebo, ele foi pensado com aplicações robóticas científicas em mente. A especificação do formato é encontrada no site [http://sdformat.org/spec].

O terceiro passo o desenvolvimento do projeto é compreender e implementar os arquivos básicos que descrevam como o robô funciona na simulação. Cada Pioneer-3AT na simulação, de acordo com nosso projeto, é composto por diversos componentes. Para facilitar a compreensão, dividimos esses componentes entre seis categorias: (a) hardware; (b) controle e tranformação; (c) localização e mapeamento simultâneos; (d) monte-carlo localization; (e) navegação; (f) visualização.

O hardware é responsável em garantir que a simulação do robô esteja integrada entre o ROS e o Gazebo por completo, e que as funcionalidades do robô sejam acessíveis(como a funcionalidade de andar). Essa categoria é composta pelos componentes: (i) um nodo do ROS que faz ponte entre o Gazebo do ROS, trazendo a Odometria do Gazebo para o ROS, e enviando as mensagens de velocidade do ROS para o Gazebo; (ii) um nodo do ROS que publica os estados das juntas do robô no ROS; (iii) nodo que faz a ponte do Gazebo para o ROS com as leituras de scanner laser. Os processos se chamam, respectivamente: sf_gazebo_bridge, joint_state_publisher e sf_gazebo_laserscan.

A categoria controle e transformação é a categoria responsável por publicar a transformação geométrica entre diversos planos de coordenadas (coordenada relativa) e também responsável por receber o controle de velocidade dos robôs. Essa categoria é composta por: (i) nodo que publica as coordenadas relativas dos componentes do robô no tópico tf (tópico geral que contém todas coordenadas relativas aos planos no ROS); (ii) nodo que publica a coordenada estática entre o scanner laser e o centro do robô no tf(usado para o gmapping); (iii) nodo que gerencia diversos tópicos de velocidade, de acordo com a prioridade, e publica apenas o controle devido(assim é possível usar vários sistemas de controle simultaneamente). Os processos se chamam, respectivamente: state_publisher, static_transform_publisher e yocs_cmd_vel_mux.

A categoria de localização e mapeamento simultâneos é responsável por fazer a localização e mapeamento simultâneo no robô. Essa categoria é composta por apenas um

nodo, que faz a leitura dos dados de laser, e tenta construir um mapa, enquanto fornece dados de localização do ambiente. O processo em questão se chama gmapping.

A categoria monte-carlo localization é uma alternativa ao gmapping, que serve para disponibilizar um mapa previamente construído, e fornecer uma localização aproximada para os robôs. Essa categoria é composta por dois nodos: (i) servidor do mapa para cada robô; (ii) implementação do monte-carlo localization para o ROS, que usa o mapa e os dados do laser para dar uma localização aproximada dos robôs. Os processos se chamam, respectivamente: map_server e amcl.

A categoria navegação é responsável por fornecer autonomia de navegação do robô por todo ambiente conhecido. Essa categoria é composta apenas pelo nodo conhecido como move_base, que faz leituras de odometria e de coordenadas relativas, e como resultado, tenta gerar um caminho até um objetivo especificado. O processo em questão se chama move_base, porém, é composto por diversos subprocessos.

A categoria de visualização é responsável por gerar uma interface que permite a visualizações ou controle de fácil uso. Essa categoria é composta por dois nodos: (i) o controlador de velocidade; (ii) visualizador da visão do robô. Os processos se chamam, respectivamente: rqt_robot_steering e rviz.

Para fazer os componentes funcionarem em conjunto, bastou criar arquivos no formato *roslaunch* [http://wiki.ros.org/roslaunch/XML] de forma que inicializem todos os nodos ao mesmo tempo. Os arquivos utilizados em nosso projeto se baseiam nos exemplos existentes no pacote pioneer-3at, apenas com adaptações nos endereços dos tópicos de cada instância dos nodos, para que o ambiente multi-robô funcione.

Os arquivos roslaunch estão na pasta launch do pacote, e estão divididos da mesma maneira que as categorias descritas para o robô. A categoria (a) está representada pelo arquivo gen_hardware.launch; (b) pelo arquivo gen_tf_and_control.launch; (c) pelo arquivo gen_gmapping.launch; (d) pelo arquivo gen_amcl.launch; (e) pelo arquivo gen_move_base.launch; (f) pelo arquivo gen_ui.launch.

O prefixo "gen" utilizado nesses arquivos serve apenas para indicar que eles são arquivos genéricos e funcionam com argumentos que especificam o nome do robô. Por exemplo, evitando a criação de várias cópias semelhantes dos arquivos. Entretanto, alguns dos componentes descritos anteriormente precisam de arquivos de texto de configurações adicionais para funcionar. Os arquivos de configuração precisam ser separados por robô porque os arquivos de texto possuem endereços de tópicos específicos de cada robô, ou coordenadas do início da simulação (tal como o amcl).

O quarto passo para o desenvolvimento do pacote, foi a criação do arquivo básico de inicialização da simulação. Esse arquivo seria o primeiro arquivo roslaunch a ser executado, antes de toda simulação. Esse arquivo se chama setup.launch, e é responsável por iniciar

o servidor do gazebo com o nosso mundo, e, também, publicar informações básicas do modelo robótico para o ROS, no tópico robot_description. Esse arquivo responsável por isso se chama setup.launch.

Por último, foi necessário fazer as implementações que o nosso projeto propôs, e outras implementações que se fizeram necessárias ou úteis ao longo do período de desenvolvimento. Entre esses, estão: (i) check_topics, responsável por analisar se a odometria e o scanner laser estão funcionando perfeitamente; (ii) generate_robot_files, responsável por criar os arquivos de configuração de todos robôs, a partir de um pequeno arquivo de entrada com a listas de robôs desejados e uma pasta com configurações genéricas(que será replicado para cada robô, substituindo o nome correto); (iii) create_formation, responsável por gerar o arquivo que descreve a formação a ser usada; (iv) robot_formation, incunbido de esperar uma coordenada do mapa, e então enviar objetivos para os robôs executarem a formação na localização escolhida. Há um script chamado suporteformacao.launch que automatiza a execução do check_topics e do robot_formation para que a utilização desses processos seja facilitada durante a simulação.

3.4.1.1 COMANDOS BÁSICOS DO ROS

O ROS possui vários comandos para interação:

`roscore` - processo principal do ROS. É preciso para o sistema de tópicos funcionar.

`roslaunch` - um processo de automação de execução de scripts, que inclui parâmetros e outras opções. Esse processo também executa o roscore, quando necessário. Os arquivos de entrada seguem o formato roslaunch.

`rosrun` - ferramenta de fácil acesso a todos os executáveis do pacote.

`roscd` - um *change directory*, com atalho para os pacotes do ROS.

`rostopic` - processo que ajuda a visualizar os tópicos e as mensagens trafegando através do sistema.

`rosclean` - usado para apagar os logs do ROS.

3.4.1.2 COMANDOS BÁSICOS DO GAZEBO

gzclient - inicia a interface gráfica do gazebo.

gzserver - inicia o servidor do gazebo.

gz - processo de acesso, modificação e leitura da execução do Gazebo. Usado para analisar tópicos, modificar física, entre outros.

3.4.1.3 COMANDOS BÁSICOS DO CATKIN

catkin_init_workspace - cria os arquivos básicos do espaço de trabalho catkin_make - usado para compilar os projetos. Deve ser executado da raiz do workspace. catkin_create_pkg - cria a pasta e esqueleto de um pacote do ROS.

3.4.1.4 TÓPICOS DO ROS

Existem muitos tópicos envolvidos numa simulação, e eles podem variar de acordo com os processos que estão sendo usados na simulação. Os tópicos são, por exemplo, endereços de sala de bate-papo. Só que nessas salas, alguns processos estão apenas querendo ouvir a conversa, e outros apenas querendo falar. Cada tópico é estritamente definido por um formato de mensagens. Como há muitos tópicos, os que ganham destaque são os mais relacionados ao nosso projeto:

/clicked_point - coordenada de um ponto no mapa. Usado para definir onde a formação será feita.
/robot01/laserscan - dados do laser que estão sendo obtidos através da tradução Gazebo-ROS.
/robot01/amcl_pose - tópico com a coordenada do robô, segundo o monte-carlo localisation.
/robot01/cmd_vel - tópico onde devem ser publicados os comandos de velocidade, pelo multiplexador de velocidades.
/robot01/odom - dados de odometria que estão sendo obtidos através da tradução Gazebo-ROS.
/robot01/map - mapa gerado pelo SLAM, ou por um servidor de mapa pronto.
/robot01/move_base_simple/goal - tópico onde deve ser publicado o objetivo do robô.
/tf - tópico geral onde todos os componentes da simulação são publicados com suas respectivas coordenadas em relação a outro plano, como uma hierarquia.

Em um ambiente normal existe dezenas ou centenas de tópicos no ROS. Apenas para exemplificar, aqui está a lista completa de tópicos para a simulação de um único robô:

/clicked_point

/clock

/gazebo/link_states

/gazebo/model_states

/gazebo/parameter_descriptions

/gazebo/parameter_updates

/gazebo/set_link_state

/gazebo/set_model_state

/robot01/amcl_pose

/robot01/cmd_vel

/robot01/initialpose

/robot01/joint_states

/robot01/laserscan

/robot01/map

/robot01/map_metadata

/robot01/move_base/cancel

/robot01/move_base/cmd_vel

/robot01/move_base/feedback

/robot01/move_base/goal

/robot01/move_base/odom

/robot01/move_base/result

/robot01/move_base/status

/robot01/move_base_simple/goal

/robot01/nodelet_manager/bond

/robot01/odom

/robot01/particlecloud

/robot01/ps3joy/cmd_vel

/robot01/robot01_cmd_vel_mux/active

/robot01/robot01_cmd_vel_mux/parameter_descriptions

/robot01/robot01_cmd_vel_mux/parameter_updates

/robot01/rqt/cmd_vel

/robot01_amcl/parameter_descriptions

/robot01_amcl/parameter_updates

/robot01_move_base/NavfnROS/plan

/robot01_move_base/NavfnROS/potential

/robot01_move_base/TrajectoryPlannerROS/cost_cloud

/robot01_move_base/TrajectoryPlannerROS/global_plan

/robot01_move_base/TrajectoryPlannerROS/local_plan

/robot01_move_base/TrajectoryPlannerROS/parameter_descriptions

/robot01_move_base/TrajectoryPlannerROS/parameter_updates

/robot01_move_base/current_goal

/robot01_move_base/global_costmap/costmap

/robot01_move_base/global_costmap/costmap_updates

/robot01_move_base/global_costmap/footprint

/robot01_move_base/global_costmap/inflation_layer/parameter_descriptions

/robot01_move_base/global_costmap/inflation_layer/parameter_updates

/robot01_move_base/global_costmap/obstacle_layer/parameter_descriptions

/robot01_move_base/global_costmap/obstacle_layer/parameter_updates

/robot01_move_base/global_costmap/obstacle_layer_footprint/footprint_sta
mped

/robot01_move_base/global_costmap/obstacle_layer_footprint/parameter_de
scriptions

/robot01_move_base/global_costmap/obstacle_layer_footprint/parameter_up
dates

/robot01_move_base/global_costmap/parameter_descriptions

/robot01_move_base/global_costmap/parameter_updates

/robot01_move_base/global_costmap/static_layer/parameter_descriptions

/robot01_move_base/global_costmap/static_layer/parameter_updates

/robot01_move_base/local_costmap/costmap

/robot01_move_base/local_costmap/costmap_updates

/robot01_move_base/local_costmap/footprint

/robot01_move_base/local_costmap/inflation_layer/parameter_descriptions

/robot01_move_base/local_costmap/inflation_layer/parameter_updates

/robot01_move_base/local_costmap/obstacle_layer/parameter_descriptions

/robot01_move_base/local_costmap/obstacle_layer/parameter_updates

/robot01_move_base/local_costmap/obstacle_layer_footprint/footprint_stamped

/robot01_move_base/local_costmap/obstacle_layer_footprint/parameter_descriptions

/robot01_move_base/local_costmap/obstacle_layer_footprint/parameter_updates

/robot01_move_base/local_costmap/parameter_descriptions

/robot01_move_base/local_costmap/parameter_updates

/robot01_move_base/parameter_descriptions

/robot01_move_base/parameter_updates

/rosout

/rosout_agg

/tf

/tf_static

3.4.2 APLICAÇÕES IMPLEMENTADAS

Durante o desenvolvimento desse projeto, além de todos os arquivos de configuração, hierarquia de diretórios e scripts de lançamento de arquivos, implementamos quatro programas para nos ajudar a atingir o resultado esperado. Eles são: (i) generate_robot_files; (ii) check_topics; (iii) create_formation; (iv) robot_formation. Nas próximas subseções, vamos descrever brevemente o funcionamento dessas aplicações.

3.4.2.1 GENERATE_ROBOT_FILES

Esse processo foi criado na intenção de facilitar a criação dos arquivos de configuração de diversos robôs iguais. Como cada robô precisa ter o próprio arquivo de configuração, é possível imaginar que seria vantajosa a existência de uma pasta com um "template" de configurações.

De maneira resumida, o processo apenas cria cópias do template em uma localização estipulada, substituindo os nomes dos robôs, e colocando a coordenada iniciais nos lugares corretos. Esse é o processo principal a usar o arquivo robots.txt, que contém a lista dos robôs e suas coordenadas x e y, respectivamente, em cada linha.

Esse programa recebe como argumentos uma pasta que sirva de template, a pasta destino onde serão colocados os robôs, uma lista de robôs com suas coordenadas iniciais, um caminho de lançamento do roslaunch do robô (que é relativo a raiz do pacote), e o caminho de um arquivo roslaunch resultante, para iniciar a simulação.

3.4.2.2 CHECK_TOPICS

Esse processo foi criado com o intuito de verificar se a simulação já está preparada para iniciar a formação. A necessidade de criar esse processo se originou devido a um bug no Gazebo, onde nem sempre a subscrição de um tópico do gazebo está chamando o método *callback*. Ou seja, aleatoriamente os tópicos de odometria ou dos dados do scanner laser de determinado robô acabam por não receber nenhum dado. Sem isso, outros processos não funcionam corretamente, impedindo de fazer a simulação.

Para criar esse processo, é necessário apenas se inscrever no tópico do ROS de odometria e do scanner laser de cada robô, e monitorar se algum dado estava sendo recebido. A lista de robôs utilizados é através do robots.txt.

3.4.2.3 CREATE_FORMATION

Uma das tarefas que fazia parte do escopo inicial do projeto, esse processo serve para criar um arquivo de texto que contem os robôs e suas coordenadas cartesianas em uma formação. As coordenadas são independente de contexto, porque será utilizada apenas a diferença entre a maior e menor coordenada de cada eixo, em referência ao ponto escolhido para a formação durante a simulação.

Dentre as formações escolhidas para implementar, implementamos 4 formações: em linha no eixo X, em linha no eixo Y, em forma de V, e circular. Resolvemos considerar que cada robô da lista seria representado em uma célula de tamanho x1, no eixo x, e tamanho y1, no eixo y. Assim, considerando o tamanho dessas células, calculamos a distância de cada célula do robô de acordo com o formato desejado para formação. Por exemplo, em uma formação em forma de V, com três robôs, onde cada célula tem tamanho 1x1, a lista de formação resultante poderia ser:

r1	0	0
r2	-1	-1
r3	1	-1

Onde a formação resultante, representada "graficamente" seria como por exemplo:

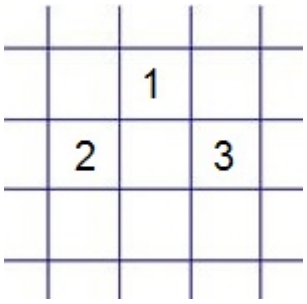

Figura 5 – Exemplo de formação em V.

O programa também considera o ângulo desejado que foi passado por parâmetro, onde na quarta coluna do arquivo resultante, é colocado o angulo desejado para que o robô se posicione. Ou seja, a rotação no eixo z. A formação circular é a única que despreza o valor angular passado por parâmetro, porque achamos interessante que essa formação tenha o próprio cálculo do ângulo, de acordo com a posição do robô no círculo.

3.4.2.4 ROBOT_FORMATION

A maior ambição desse projeto, que é obter a formação autônoma dos robôs simulados. Esse processo utiliza o arquivo de formação gerado pelo create_formation para estipular objetivos para cada robô no mapa de acordo com um ponto escolhido no mapa. Basicamente o processo se inscreve no tópico /clicked_point e espera por um ponto ser clicado no rviz. Quando ele recebe um ponto no mapa, ele abre o arquivo de formação, e centraliza as coordenadas da formação de acordo com seus valores máximos e mínimos para cada eixo, e então soma a coordenada gerada com o ponto escolhido.

Com as coordenadas resultantes, o programa envia para o primeiro robô da lista seu objetivo e espera que o robô esteja a uma distância d deste objetivo para seguir para o próximo robô. Durante as execuções, percebemos que distâncias muito longas faziam com que os robôs efetuassem deslocamento mais cedo, aumentando a possibilidade de colisão entre si, e com distâncias muito curtas a simulação demorava mais porque os robôs precisavam estar muito perto dos objetivos para que o próximo fosse enviado.

3.5 EXECUÇÃO

Antes de iniciar a simulação, ou fazer uso de qualquer processo, é importante sempre efetuar a compilação do pacote caso os arquivos fonte tenham sido modificados. Para fazer isso, basta rodar o comando catkin_make à partir da raíz do espaço de trabalho catkin.

Para utilizar o pacote suporteformacao, primeiro é necessário definir os nomes e as coordenadas dos robôs a serem utilizadas. O arquivo config/robots.txt, em um exemplo

com 4 robôs, poderá conter algo como:

robot01 -7 -7
robot02 -6 -7
robot03 -7 0
robot04 -6 1

Cada robô possui uma coordenada x e y, respectivamente, de acordo com a localização inicial desejada. Para garantir que o robô inicie em uma zona livre de colisões, o teste pode ser feito no Gazebo previamente para análise das coordenadas.

Após o ajuste dos robôs a serem utilizados, deve-se executar o comando:

```
roslaunch suporteformacao generate_robot_files.launch
```

Esse comando irá criar os arquivos dos robôs em suas respectivas localizações, recomenda-se apagar o conteúdo da pasta robots, a não ser que o usuário esteja incluindo arquivos adicionais em seus robôs. Após a confirmação do término, é possível encerrar o processo sem problemas.

Com os arquivos de configuração e lançamento devidamente criados, é necessário criar uma formação, através do comando:

```
roslaunch suporteformacao create_formation.launch
```

Esse processo é responsável por criar um arquivo chamado formation.txt, que contém o nome dos robôs, e as coordenadas de suas posições na formação. Após a confirmação do término, é possível encerrar o processo sem a ocorrência de problemas.

Finalmente, o ambiente está apto a iniciar a execução através do comando:

```
roslaunch suporteformacao start.launch
```

Esse processo é um script para iniciar o arquivo setup.launch, cada um dos robôs utilizados na simulação, e, por fim, o arquivo suporteformacao.launch. Esse script irá preparar toda simulação, todos robôs, e também a implementação da formação. Basta aguardar na tela a confirmação de que todos os tópicos estão OK, que será possível enviar uma localização para a formação.

Como cada robô possui uma visualização diferente, a simulação não se inicia com nenhuma interface. Porém é possível iniciar as interfaces manualmente. Para a visualização e controle de um robô específico, é necessário executar o seguinte comando para o robô "robot01":

roslaunch suporteformacao gen_ui.launch robot_name:=robot01

Esse comando irá abrir o controle de velocidade do "robot01" e também a interface de visualização rviz, que mostra tuodo que o robô em questão está vendo. Nessa interface, também é possível dar um goal específico para o robô(ferramenta *2D Nav Goal*), ou enviar

um ponto do mapa onde será feita a formação (ferramenta *Publish Point*).

Para obter uma visualização completa do ambiente, é necessário abrir a interface do gazebo, que pode ser obtida pelo comando:

```
gzclient
```

4 RESULTADOS

As analises apresentadas a seguir foram feitas com base na execução do algoritmo *amcl*. Isto se explica pelo fato de que, para utilizar o algoritmo *gmapping* primeiramente seria necessário se movimentar pelo mapa para então designar o objetivo para os robôs, o que acaba desviando do escopo principal deste projeto.

Para execução dos testes foi utilizado um computador AMD Phenom II X4, que possui 3 GB de memória RAM e 4 processadores. Com este computador foi possivel a execução de testes com no máximo 8 robôs em um único ambiente. A partir desta quantidade, devido ao elevado número de informações a serem processadas para cada robô, se torna inviável a execução de testes para computadores que possuem configuração semelhante. Para cada caso de teste foram analisadas as seguintes informações: (i) fator de tempo real; (ii) tempo de simulação; (iii) tempo real; (iv) número de iterações realizadas.

4.1 CASO DE TESTE 1 - CENÁRIO COM UM ROBÔ

Nesta situação, foi inserido um robô em uma determinada posição do mundo, e o objetivo é que o robô se desloque até o centro do ambiente. A observação esperada neste caso de teste é que o robô faça uma varredura, a fim de conhecer sua posição, para então

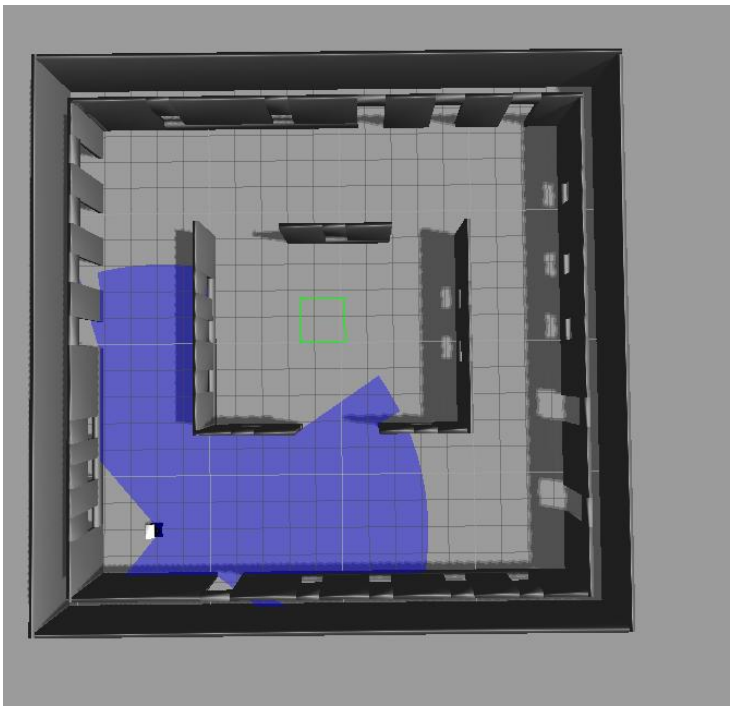

Figura 6 – Cenário de testes com um robô.

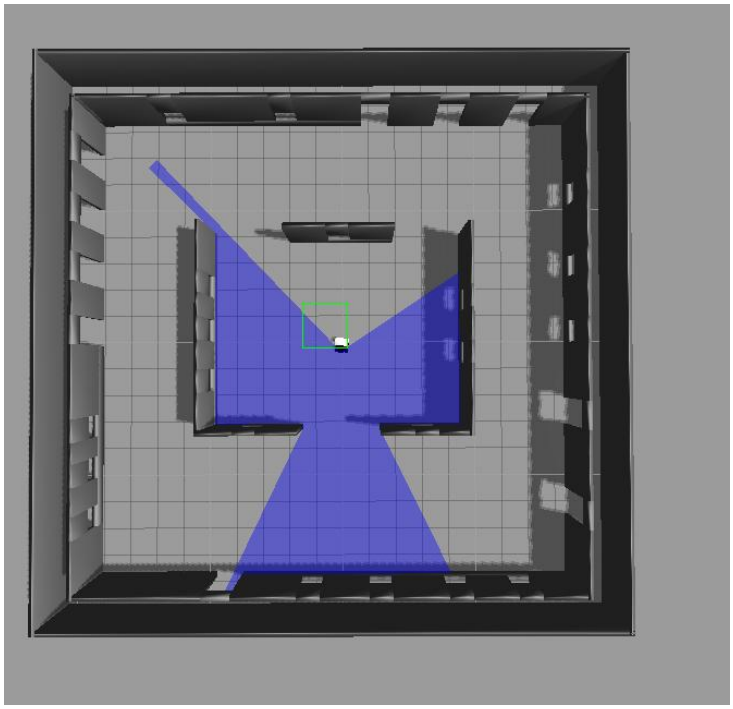

Figura 7 – Estado final do Caso de teste 1 após deslocamento do robô.

efetuar o deslocamento. Em outras palavras, o objetivo deste caso de teste consiste em explorar a localização de um determinado robô no ambiente.

Resultados obtidos:

- Fator de tempo real: 0.88
- Tempo de simulação: 00:04:30
- Tempo real: 00:03:22

4.1.1 ANÁLISE DO CASO APRESENTADO

O robô inicia com seu mapa preenchido. Logo, inicia conhecendo sua posição e para onde deve se deslocar. A medida em que efetua o deslocamento, o robô faz leituras com seu laser e as particulas se reordenam, recalculando a possível posição do robô através da amostragem de partículas.

4.2 CASO DE TESTE 2 - CENÁRIO COM DOIS ROBÔS

São inseridos dois robôs nesta situação e o comportamento esperado é que cada robô reconheça a posição, na qual o outro robô se encontra, como um obstáculo. Portanto, deverá recalcular a rota a ser traçada. Após o reconhecimento, o esperado é que os robôs se desloquem para o centro do ambiente, um após o outro, e fiquem alinhados. Neste caso de teste é explorado, além da localização, um início de formação, no qual será abordado com mais detalhes ao longo dos casos de teste efetuados.

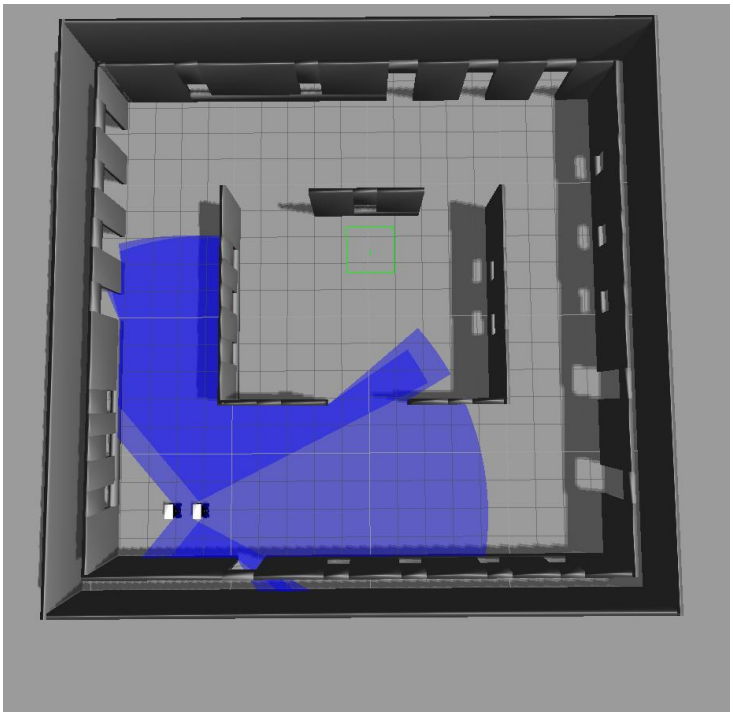

Figura 8 – Cenário de testes com dois robôs.

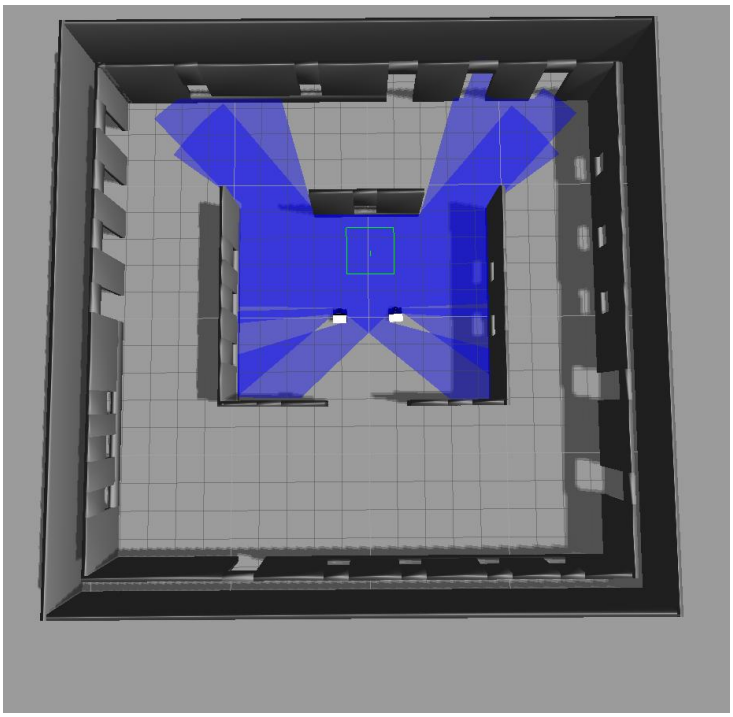

Figura 9 – Estado final do Caso de teste 2 após deslocamento dos robôs.

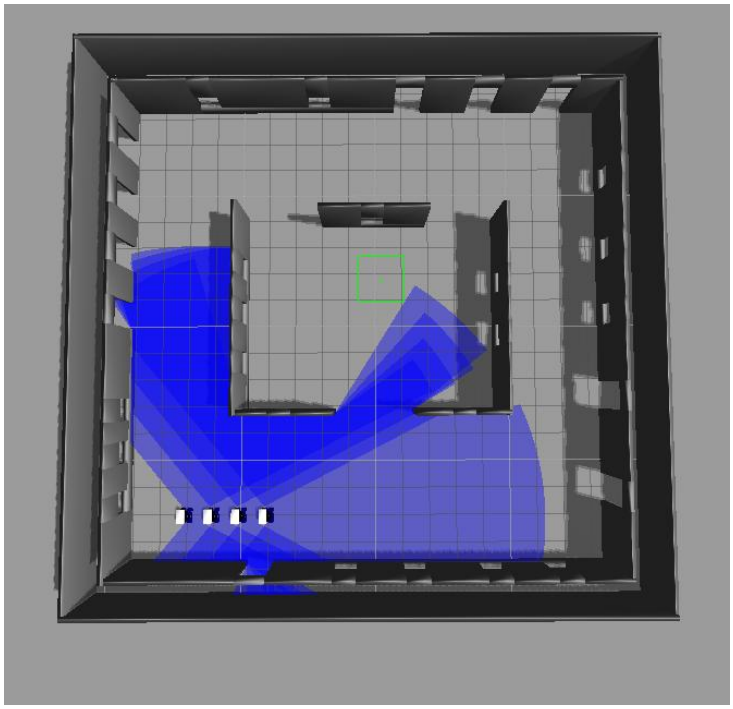

Figura 10 – Cenário de testes com quatro robôs.

Resultados obtidos:

- Fator de tempo real: 0.64
- Tempo de simulação: 00:02:30
- Tempo real: 00:04:23

4.2.1 ANÁLISE DO CASO APRESENTADO

Ambos os robôs iniciam com seu mapa preenchido. Porém, o primeiro robô identifica que ao seu lado existe um obstáculo apenas no momento em que o laser atinge o segundo robô (comportamento observado no item anterior). Ao detectar o obstáculo, o robô se desloca para o ponto desejado e o fluxo deste caso de teste acontece como descrito anteriormente.

4.3 CASO DE TESTE 3 - CENÁRIO COM QUATRO ROBÔS

São inseridos quatro robôs nesta situação e o resultado esperado é que a formação seja construída no centro do ambiente. Neste caso de teste será explorada a organização dos robôs e o tempo necessário para que entrem em formação. Note que, como nos casos de testes anteriores, as observações sobre localização foram analisadas com sucesso. Logo, para este caso de teste e os casos seguintes, não possuirá relevância.

Resultados obtidos:

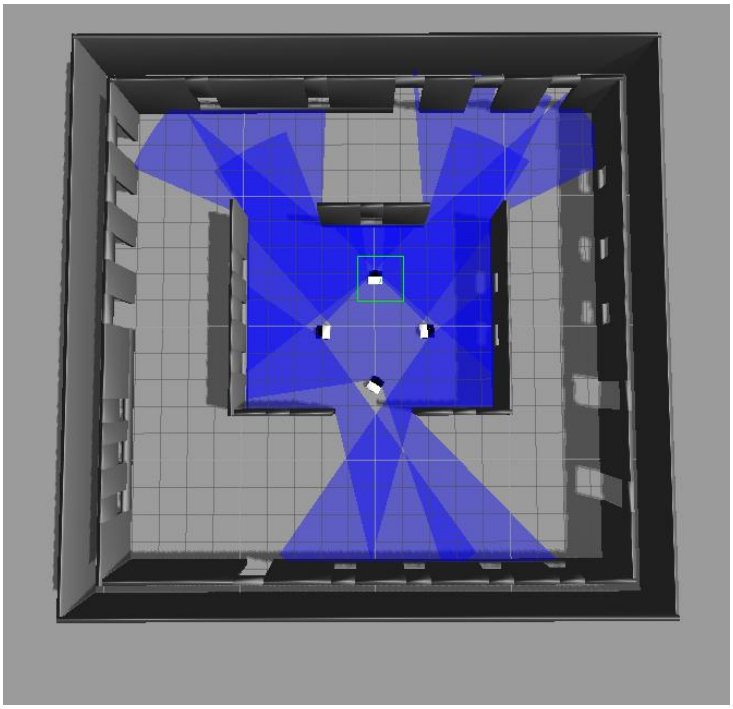

Figura 11 – Estado final do Caso de teste 3 após deslocamento dos robôs.

- Fator de tempo real: 0.21
- Tempo de simulação: 00:03:08
- Tempo real: 00:14:32

4.3.1 ANÁLISE DO CASO APRESENTADO

Cada robô identifica como obstáculo os robôs restantes, porém, não se torna necessário efetuar paradas para rotacionar e analisar o ambiente devido ao fato destes robôs possuírem o mapa do ambiente. A medida em que cada laser identifica um obstáculo, o robô, por conhecer o mapa do ambiente, mantém seu deslocamento desviando do obstáculo visualizado.

4.4 CASO DE TESTE 4 - CENÁRIO COM SEIS ROBÔS

Nesta situação é esperado que uma formação em círculo seja construída no centro do ambiente, sem a ocorrência de colisões. Assim como no caso de teste anterior, neste caso de teste será explorada a organização dos robôs e o tempo necessário para que entrem em formação.

Resultados obtidos:

- Fator de tempo real: 0.09
- Tempo de simulação: 00:03:27
- Tempo real: 00:28:04

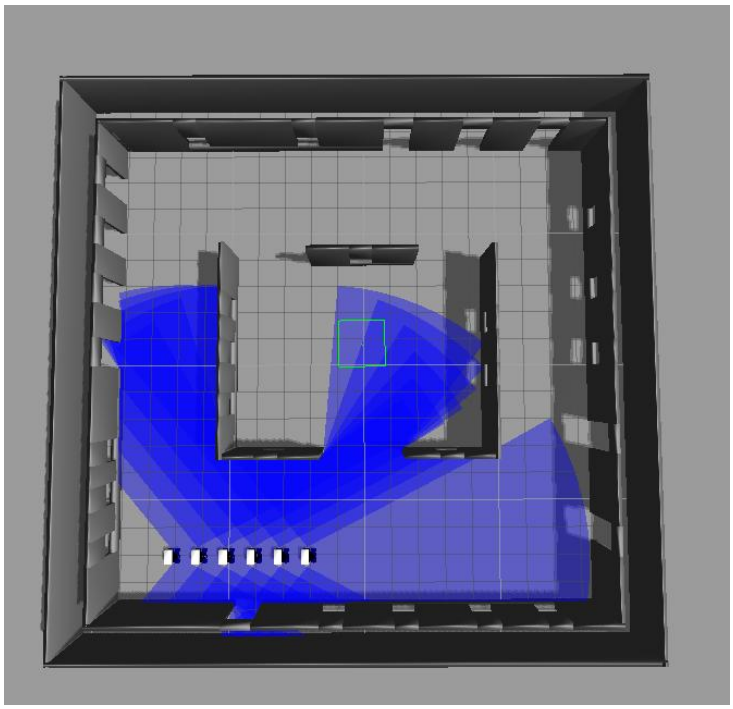

Figura 12 – Cenário de testes com seis robôs.

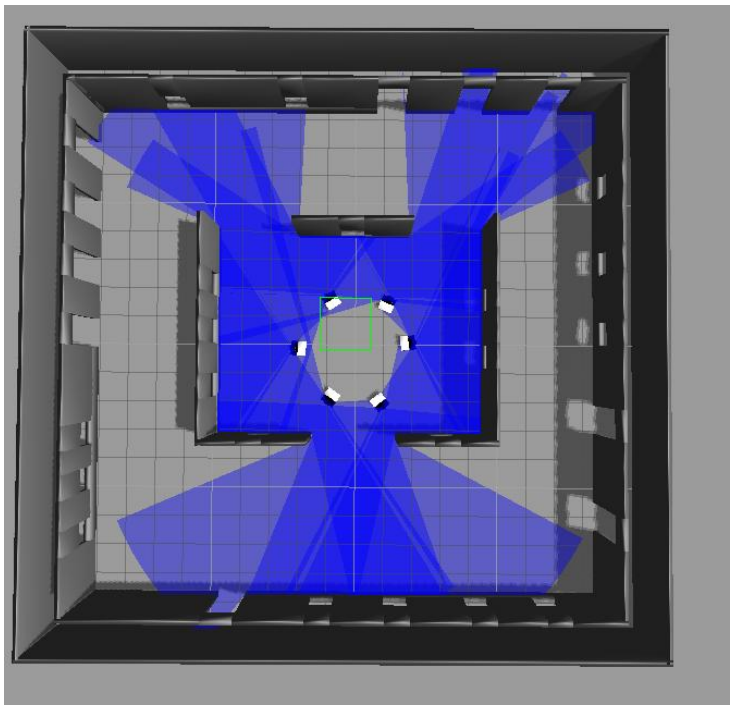

Figura 13 – Estado final do Caso de teste 4 após deslocamento dos robôs.

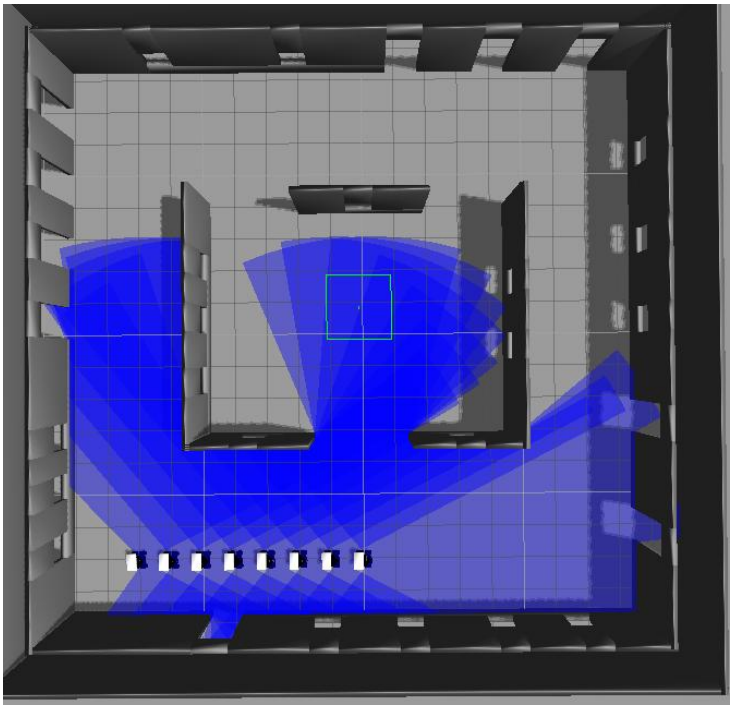

Figura 14 – Cenário de testes com oito robôs.

4.4.1 ANÁLISE DO CASO APRESENTADO

Os robôs efetuam deslocamento e o comportamento é semelhante ao item *iii)* do caso de teste anterior. O tempo de execução total aumenta, porém não de forma significativa.

4.5 CASO DE TESTE 5 - CENÁRIO COM OITO ROBÔS

Para esta última situação, assim como no caso de teste anterior, é esperado que uma formação em círculo seja construída no centro, sem a ocorrência de colisões. Os pontos a serem observados nesta situação são os mesmos observados no caso de teste anterior, além do aumento da complexidade na elaboração da formação.

Resultados obtidos:

- Fator de tempo real: 0.04
- Tempo de simulação: 00:03.38
- Tempo real: 01:04:10

4.5.1 ANÁLISE DO CASO APRESENTADO

Os robôs se deslocam para o ponto designado. Porém, o último robô a entrar na formação possui dificuldades devido ao fato de visualizar obstáculos (referente aos robôs já posicionados na formação desejada) e acaba estacionando, efetuando deslocamento em

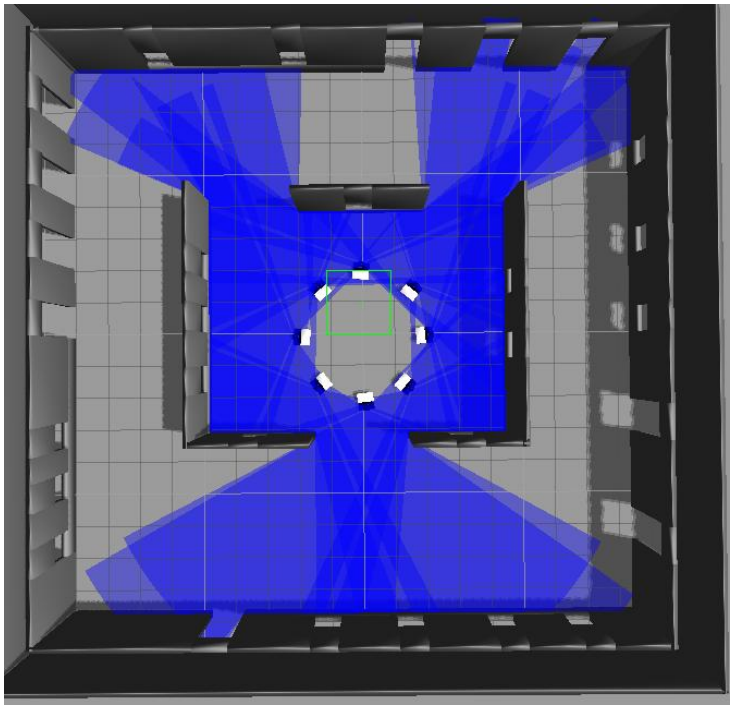

Figura 15 – Estado final do Caso de teste 5 após deslocamento dos robôs.

linha reta: para frente e para trás até corrigir suas leituras. Esta ocorrência aumentou significativamente o tempo total de execução desta simulação. Porém, é um problema exclusivo para ambientes apertados, devido ao fato dos robôs tentarem evitar o máximo possível os obstáculos.

Uma alternativa para solucionar este tipo de situação seria diminuir o tamanho da célula de cada robô, o que consequentemente diminui o tamanho de cada robô considerado como obstáculo. Porém, esta alteração pode ocasionar possíveis colisões durante o deslocamento dos robôs.

4.6 CONSIDERAÇÕES FINAIS

4.6.1 SOBRE OS RESULTADOS

O que se pode observar é que o tempo de execução aumenta a medida em que se acrescenta robôs no cenário. É notavel que com a utilização do *amcl* se obtem maior desempenho, em relação a utilização do *gmapping* ou a não utilização de algoritmos. Isto se explica pelo fato de ser mais vantajoso se deslocar para o ponto designado sem precisar explorar o mapa antes.

A utilização do algoritmo não acarreta no aumento de complexidade na execução das simulações, o que se torna vantajoso quando comparado a executar simulações sem a utilização de algum algoritmo de localização. Para aumentar significativamente a quan-

tidade de robôs disponíveis em um determinado ambiente, seria necessário possuir uma máquina com configuração superior a apresentada neste trabalho. Os resultados obtidos após a execução de cada teste podem variar de acordo com a máquina escolhida para efetuar testes.

4.6.2 SOBRE O PROJETO EM GERAL

Em um primeiro momento, houveram dificuldades na compreensão do ROS como um todo, devido ao fato deste sistema ser composto por inúmeras ferramentas, cada uma com seu propósito para o funcionamento do sistema. Em um segundo momento, houveram dificuldades na instalação do ambiente, o qual exigiu a instalação de todos os componentes descritos ao longo deste Volume. O ideal para a execução dos casos de teste seria realizar estas execuções em um computador com configuração superior, a fim de efetuar os testes com maior rapidez. Afinal, conforme apresentado neste capítulo, para executar um caso de testes composto por oito robôs foi necessário executar o teste durante uma hora e quatro minutos, para concluir uma simulação de três minutos e trinta e oito segundos.

Os problemas relatados nesta seção não impactaram no bom andamento deste projeto, o qual foi implementado dentro do prazo estipulado com sucesso.

5 CONCLUSÃO

Com a grande quantidade de tecnologia existente na área de robótica se torna possível explorar o ramo da comunicação entre robôs, permitindo assim a criação ou aprimoração de frameworks para este determinado fim. Com a implementação deste projeto será possível iniciar projetos a partir do que ja foi elaborado, devido ao fato deste trabalho fornecer uma camada para os solucionar os problemas relacionados a formação de enxame de robôs. Além do fato de contribuir com conhecimentos sobre a ferramenta ROS, incentivando assim a pesquisa e desenvolvimento nesta área.

Um possível trabalho a ser desenvolvido, com base no framework disponibilizado neste projeto, poderia consistir na implementação da movimentação do enxame de robôs.

REFERÊNCIAS

1 IROBOT. Disponível em: <http://www.irobot.com/>. Acesso em: abril de 2014. Citado na página 11.

2 ELETRONICSTEACHER.COM. Disponível em: <http://www.electronicsteacher.com/robotics/what-is-robotics.php>. Acesso em: abril de 2014. Citado na página 11.

3 GAUTAM, A.; MOHAN, S. A review of research in multi-robot systems. *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, Ieee, p. 1–5, ago. 2012. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6304778>. Acesso em: abril de 2014. Citado na página 12.

4 BAYINDIR, L.; ŞAHIN, E. A Review of Studies in Swarm Robotics. *Turkish Journal of Electrical Engineering & Computer Sciences*, v. 15, n. 2, p. 115–147, 2007. Disponível em: <http://journals.tubitak.gov.tr/elektrik/issues/elk-07-15-2/elk-15-2-2-0705-13.pdf>. Acesso em: abril de 2014. Citado na página 12.

5 WISEGEEK. *What is Swarm Robotics?* Disponível em: <http://www.wisegeek.com/what-is-swarm-robotics.htm>. Acesso em: abril de 2014. Citado na página 12.

6 WU, D.; SU, H. An improved probabilistic approach for collaborative multi-robot localization. *Robotics and Biomimetics, 2008. ROBIO 2008. ...*, p. 1868–1875, 2009. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4913286>. Acesso em: abril de 2014. Citado na página 12.

7 ZAMAN, S.; SLANY, W.; STEINBAUER, G. ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues. *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, Ieee, p. 1–5, abr. 2011. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5876943>. Acesso em: abril de 2014. Citado na página 13.

8 DOWLING, K. *What is robotics?* 1996. Disponível em: <http://www.cs.cmu.edu/~chuck/robotpg/robofaq/1.html>. Acesso em: 5 de julho de 2014. Citado na página 15.

9 BELLIS, M. *The Definition of a Robot*. 2006. Disponível em: <http://inventors.about.com/od/roboticsrobots/a/RobotDefinition.htm>. Acesso em: 5 de julho de 2014. Citado na página 15.

10 THRUN, S. et al. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence, Summer 2001*, 2001. Disponível em: <http://robots.stanford.edu/papers/thrun.robust-mcl.pdf>. Acesso em: maio de 2014. Citado na página 16.

11 FOX, D. Adapting the Sample Size in Particle Filters Through KLD-Sampling. 2003. Disponível em: <http://www.robots.ox.ac.uk/~cvrg/hilary2005/adaptive.pdf>. Acesso em: maio de 2014. Citado na página 16.

12 ZHANG, L.; ZAPATA, R.; LéPINAY, P. Self-adaptive monte carlo for single-robot and multi-robot localization. *Automation and Logistics, ...*, n. August, p. 1927–1933, 2009. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5262621>. Acesso em: abril de 2014. Citado na página 16.

13 FOX, D.; BURGARD, W.; THRUN, S. Markov Localization for Mobile Robots in Dynamic Environments. *1999 Journal of Artificial Intelligence Research*, 1999. Disponível em: <http://robotics.caltech.edu/~jerma/research_papers/MarkovLocalization.pdf>. Acesso em: abril de 2014. Citado na página 17.

14 WELCH, G.; BISHOP, G. An Introduction to the Kalman Filter. 2006. Disponível em: <http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf>. Acesso em: abril de 2014. Citado na página 17.

15 MOBILEROBOTS.COM. *Pioneer 3-AT*. Disponível em: <http://www.mobilerobots.com/ResearchRobots/P3AT.aspx>. Acesso em: 6 de julho de 2014. Citado na página 26.

16 XU, Z.; ZHUANG, Y. Simultaneous Localization and Map Building with Modified System State. 2009. Disponível em: <http://cdn.intechopen.com/pdfs-wm/6945.pdf>. Acesso em: maio de 2014. Citado na página 28.