UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

DBML: Uma Biblioteca de Gerenciamento Dinâmico de Banda para Sistemas Multirrobôs Baseado em ROS

Ricardo Emerson Julio

Itajubá, Julho de 2015

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Ricardo Emerson Julio

DBML: Uma Biblioteca de Gerenciamento Dinâmico de Banda para Sistemas Multirrobôs Baseado em ROS

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação

Área de Concentração: Sistemas de Computação

Orientador: Prof. Dr. Guilherme Sousa Bastos

Julho de 2015
Itajubá - MG

Agradecimentos

Agradeço primeiramente a Deus, por ter guiado os meus passos e ter me dado sabedoria para trilhar esse caminho.

Aos meus pais, João e Tina, pelo apoio incondicional.

Ao meu orientador e amigo, Guilherme Sousa Bastos, por toda a ajuda e disponibilidade sempre que precisei.

Aos meus amigos e familiares pelos momentos de alegria. Em especial ao Emerson e ao Lênio, por trilharem grande parte desse caminho comigo.

Aos colegas do ICC-Inatel, em especial ao Ronaldo e à Cristiani, por entenderem as minhas ausências. Vocês foram uma parte importante dessa caminhada.

À Fapemig e à Capes pelo apoio.

E um agradecimento mais que especial à minha eterna namorada, Pamela, por todo o apoio, compreensão e amor. Você foi uma parte essencial e não tenho dúvidas que sem você essa jornada seria muito mais difícil. Esse trabalho também é seu! Amo você!

No fim tudo dá certo, e se não deu certo é porque ainda não chegou ao fim.

Fernando Sabino

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO
EM CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Ricardo Emerson Julio

DBML: Uma Biblioteca de Gerenciamento Dinâmico de Banda para Sistemas Multirrobôs Baseado em ROS

Dissertação aprovada por banca examinadora em 03 de Julho de 2015, conferindo ao autor o título de **Mestre em Ciência e Tecnologia da Computação**.

Banca Examinadora:
Prof. Dr. Guilherme Sousa Bastos - UNIFEI (Orientador)
Prof. Dr. André Luis Marques Marcato - UFJF
Prof. Dr. Carlos Henrique Valério de Moraes - UNIFEI
Prof. Dr. Bruno Tardiole Kuehne - UNIFEI

Itajubá
2015

Resumo

A comunicação é um componente importante em sistemas multirrobôs; a performance do sistema pode ser seriamente afetada quando o número de robôs do sistema ou o número de canais de comunicação aumenta. A Biblioteca de Gerenciamento Dinâmico de Banda (DBML - *Dynamic Bandwidth Management Library*) foi projetada de forma a maximizar a utilização da banda em sistemas multirrobôs. O sistema desenvolvido prioriza os canais de comunicação de acordo com eventos que ocorrem no ambiente oferecendo maior largura de banda para os canais de comunicação mais prioritários. A biblioteca foi desenvolvida utilizando o ROS (*Robot Operating System*) de forma a separar as funcionalidades em módulos independentes que possam ser reutilizados e melhorados em trabalhos futuros. Este trabalho apresenta a biblioteca desenvolvida utilizando um problema de otimização linear e um exemplo do uso da biblioteca em uma aplicação de teleoperação onde diversas simulações foram feitas. Os resultados mostraram que o DBML atribui uma maior frequência de envio de informações para os canais de comunicação mais prioritários, permitindo assim, uma maior eficiência na execução das tarefas.

Palavras-Chave: sistemas multirrobôs; ROS; QoS; gerenciamento dinâmico de banda; pacote ROS.

Abstract

Communication is an important component in multi-robots systems; system performance may get affected when the number of robots or communication channels increases. The Dynamic Bandwidth Management Library (DBML) was designed to provide a way of maximizing bandwidth usage in multi-robots systems. The developed system prioritizes communication channels according to environment events and offers greater bandwidth for the most important channels. The library was developed in ROS (Robot Operating System) in order to separate the functionalities into independent modules to be reused and improved in future works. This paper presents the library design as a linear optimization problem and an example of the library usage in a teleoperation application where many simulations were performed. The results showed that the DBML assigns a higher frequency of sending information to the most priority communication channels, allowing greater efficiency in performing the tasks.

Keywords: multi-robot systems; ROS; QoS; dynamic bandwidth management; ROS package.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos-Fonte

Glossário

DBML	*Dynamic Bandwidth Management Library*
DNS	*Domain Name Service*
EPON	*Ethernet Passive Optical Network*
FPS	*Frames por Segundo*
IA	*Instantaneous Assignment*
IDL	*Interface Definition Language*
IP	*Internet Protocol*
MD5	*Message-Digest algorithm 5*
MRTA	*Multi-Robot Task Allocation*
MR	*Multi-Robot Tasks*
MT	*Multi-Task Robots*
QoS	*Qualidade de Serviço*
RDE	*Robot Development Environment*
FR	*Rádio Frequência*
ROS	*Robot Operating System*
RPC	*Remote Procedure Call*
SAIL	*Stanford Artificial Intelligency Laboratory*
SMR	*Sistemas Multirrobôs*
SR	*Single-Robot Tasks*
ST	*Single-Task Robots*
TA	*Time-extended Assignment*
TCP	*Transmission Control Protocol*
TCPROS	*ROS-based Transmission Control Protocol*
UDP	*User Datagram Protocol*
UDPROS	*ROS-based User Datagram Protocol*
XML	*eXtensible Markup Language*
XMLRPC	*XML-based Remote Procedure Call*

1 Introdução

1.1 Motivação

Sistemas multirrobôs (SMR) é uma área de pesquisa desafiante e motivadora na robótica com muitas áreas de aplicação, como redes de sensores autônomos, transporte de grandes objetos, monitoramento aéreo e aquático de poluição, detecção de focos de incêndio em florestas, sistemas de transporte, busca e resgate em áreas de desastre ou qualquer conjunto de tarefas que podem ser subdivididas entre múltiplos robôs. Problemas que poderiam ser resolvidos por um único robô, projetado para efetuar múltiplas tarefas, podem se beneficiar do uso de um sistema multirrobô. A confiança e robustez da solução podem aumentar com a combinação de vários robôs (LIMA; CUSTODIO, 2005).

A comunicação é um importante componente que deve ser cuidadosamente considerado em sistemas multirrobôs. O número de pacotes transmitidos entre os agentes do sistema pode aumentar com o número de sensores, atuadores e robôs (BALCH; ARKIN, 1994). Uma aplicação de teleoperação é um bom exemplo que ilustra a transmissão de dados entre os agentes do sistema. Nesse tipo de aplicação, um usuário ou um dispositivo mestre (central que controla os robôs automaticamente) controla à distância um conjunto de robôs móveis (FONG; THORPE; BAUR, 2003). De forma a disponibilizar um *feedback* visual do ambiente para que o operador do sistema controle remotamente os robôs, o vídeo coletado das câmeras dos robôs deve ser enviado para a central. Dessa forma, o número de pacotes transmitidos aumenta quando existem mais robôs no sistema ou quando existe uma melhora na qualidade do vídeo sendo enviado. Esse aumento do número de pacotes pode afetar a performance do sistema em ambientes onde existe uma restrição da largura de banda disponível.

Em tais sistemas, não existe a necessidade de transmitir dados dos sensores a todo instante e na mesma frequência. Considerando o exemplo da aplicação de teleoperação, a frequência de envio do vídeo pode ser diminuída caso a velocidade do robô diminua ou não exista nenhum obstáculo próximo ao robô. Isso acontece porque em baixas velocidades

ou quando o robô se encontra a uma distância segura de obstáculos, o tempo de resposta do operador pode ser maior do que em situações onde o robô está sendo operado com uma velocidade mais alta ou se encontra próximo de obstáculos. Ou seja, não é necessário o máximo de qualidade disponível do vídeo e a frequência do sensor pode ser diminuída se não existe nada de relevante para a tarefa ocorrendo no ambiente (MANSOUR et al., 2011).

Outro componente de um sistema multirrobô que precisa ser considerado é a largura de banda. Uma perda de pacotes pode ocorrer quando o número de dados sendo transmitidos é maior que a banda disponível. Nesse caso, a frequência de todos os sensores deve ser ajustada de modo a não exceder o máximo de banda disponível para a aplicação. A tarefa de ajustar as taxas de comunicação pode ser um desafio; em uma solução estática, as frequências não podem ser ajustadas quando existe uma mudança no ambiente ou quando a largura de banda disponível para a aplicação muda.

Alguns trabalhos têm estudado diferentes tipos de algoritmos de gerenciamento de recursos de forma a resolver o problema de alocação de banda em sistemas de controle utilizando redes de computadores. Tais aplicações podem ser categorizadas em dois grupos: soluções estáticas (CHASKAR; MADHOW, 2003) e soluções dinâmicas (TIPSUWAN et al., 2009). Métodos estáticos não possuem a capacidade de adaptarem a solução durante mudanças no ambiente enquanto abordagens dinâmicas adicionam um custo extra de tempo de computação e geralmente possuem soluções mais complexas (MANSOUR et al., 2011).

Tratar o problema de gerenciamento de banda em sistemas multirrobôs não é uma tarefa trivial. A falta desse gerenciamento pode impactar na comunicação entre os elementos do sistema e afetar a eficiência da aplicação em ambientes com restrições de largura de banda. Segundo Gerkey, Vaughan e Howard (2003), o desenvolvimento de software para aplicações para robôs é um tarefa complexa e que consome tempo. Trabalhar com sistemas com múltiplos e distribuídos robôs é uma tarefa ainda mais complexa devido ao aumento do número de robôs e às dificuldades da programação em rede. Dessa forma, a criação de estratégias e ferramentas que padronizem o desenvolvimento e permitam a reutilização de funcionalidades podem facilitar o desenvolvimento de aplicações para sistemas multirrobôs e coloborar com as diversas pesquisas na área.

O ROS (*Robot Operating System*) é um *framework* flexível para o desenvolvimento de software para robôs com o objetivo de encorajar o desenvolvimento colaborativo de software para a área da robótica (QUIGLEY et al., 2009). O *framework* permite que

um sistema construído utilizando a arquitetura do ROS possa ser reutilizado por outros pesquisadores. Assim, o desenvolvimento de uma biblioteca de gerenciamento dinâmico de banda utilizando o ROS pode contribuir com as pesquisas futuras e no desenvolvimento de sistemas na área da robótica.

1.2 Objetivos

O principal objetivo desse trabalho é o desenvolvimento de uma biblioteca para ajudar os desenvolvedores de sistemas multirrobôs a gerenciarem a alocação de banda de forma dinâmica. A biblioteca deve ser desenvolvida como um pacote ROS para ser utilizada em diversos projetos de sistemas envolvendo robôs.

A biblioteca deve ser testada através de uma simulação de uma aplicação real de forma a gerar dados que validem as melhorias propostas no gerenciamento da banda e disponibilizada para a comunidade do ROS com o objetivo de colaborar com as pesquisas na área da robótica e incentivar trabalhos futuros, desenvolvidos em ROS, que tratem do problema de comunicação em sistemas multirrobôs. E finalmente, um estudo sobre o ROS, com o foco na sua camada de comunicação, deve ser feito para servir de referências em trabalhos futuros utilizando o framework.

1.3 Contribuições

A principal contribuição desse trabalho é o desenvolvimento da Biblioteca de Gerenciamento Dinâmico de Banda (DBML - *Dynamic Bandwidth Management Library*) que realiza o gerenciamento dos canais de comunicação através do ajuste dinâmico das frequências dos canais de comunicação. Isto é conseguido através do monitoramento dos eventos que ocorrem no ambiente, atribuindo uma prioridade a cada canal de comunicação. A estratégia adotada considera que, quando uma mudança ocorre no ambiente do robô, como a aproximação de um obstáculo por exemplo, a comunicação entre os elementos do sistema deve obter todos os recursos de banda necessários, de forma a manter o mesmo nível de performance (MANSOUR et al., 2011).

Um problema de otimização linear é dinamicamente resolvido utilizando o tamanho da mensagem, a prioridade baseada nos eventos do ambiente e a largura de banda disponível para a aplicação. A biblioteca foi projetada de forma a ser utilizada em um sistema com um variável número de robôs heterogêneos. No sistema proposto, os robôs podem efetuar

qualquer tipo de tarefa colaborativa que necessita do envio de dados sensoriais para outros agentes do sistema, tais como outros robôs ou um dispositivo mestre (central de controle).

1.4 Estrutura do Trabalho

No capítulo 2 é apresentado uma revisão sobre o gerenciamento de banda em sistemas multirrobôs, onde são definidos os principais conceitos, as aplicações e os trabalhos relacionados. Nesse capítulo é feito uma análise sobre o gerenciamento de banda em redes de computadores e uma revisão focada em aplicações utilizando sistemas multirrobôs. O capítulo 3 descreve o ROS, a plataforma sobre o qual a biblioteca foi construída, onde são apresentadas as definições e uma análise da sua camada de comunicação. O capítulo 4 trata do desenvolvimento da biblioteca (DBML) e das suas principais características e configurações. As simulações utilizando uma aplicação de teleoperação e os resultados obtidos são mostrados no capítulo 5, enquanto o capítulo 6 contém as conclusões e sugestões de trabalhos futuros.

2 Gerenciamento de Banda em Sistemas Multirrobôs

2.1 Gerenciamento de Banda em Redes de Computadores

O gerenciamento de tráfego em redes de computadores consiste de um conjunto de mecanismos que determinam como caracterizar o tráfego, alocar a banda e gerenciar os recursos da rede baseado nas características e nos requisitos da conexão. Esse gerenciamento é essencial para uma eficiente utilização dos recursos da rede e para prover a qualidade do serviço (QoS - *Quality of Service*), garantindo aos usuários da rede uma transmissão de pacotes de alta velocidade (PEYRAVIAN; GÜN, 1996).

De acordo com Ganz, Ganz e Wongthavarawat (2003), o termo QoS é utilizado com uma variedade de significados e perspectivas. Diferentes grupos interpretam QoS de modos diferentes. Entre os pesquisadores que trabalham com aplicações, QoS geralmente se refere à qualidade percebida pelo usuário ou aplicação, enquanto que para os pesquisadores de redes de computadores, QoS é definido como uma medida da qualidade do serviço que a rede oferece. Crawley e Nair (1998) caracteriza QoS como um conjunto de requisitos de serviços que devem ser atingidos ao transportar um fluxo de pacotes de uma origem para o seu destino. Proporcionar QoS, principalmente tratando das restrições da taxa de dados e do atraso nos pacotes, é um dos requisitos para a transmissão de dados em alta velocidade (ANDREWS et al., 2001).

As conexões em uma rede podem ter larguras de banda diferentes. Se a banda alocada for igual à taxa máxima da conexão, a QoS tem o seu valor mais alto, mas vai existir um desperdício de rede quando a conexão não estiver operando no seu nível máximo. Um esquema de alocação de banda ideal deve adaptar às caracteríscas do ambiente de forma a disponibilizar a banda não utilizada para outras conexões.

Uma aplicação que se comunica através de uma rede de computadores possui, geralmente, muitos elementos conectados competindo pelo uso da banda disponível. Yaïche,

Mazumdar e Rosenberg (2000) apontam algumas questões que necessitam ser consideradas nesse tipo de aplicação:

1. Uma alocação de banda eficiente deve levar em consideração que diferentes elementos do sistema possuem necessidades e requisitos de performance diferentes;
2. A noção de *equidade* deve ser levada em consideração;
3. A habilidade de implementar a alocação de uma forma distribuída com um *overhead* mínimo de comunicação;
4. A questão do aproveitamento da largura de banda, de tal maneira que a utilização da rede será maximizada se a alocação da banda for feita de acordo com os itens 1 e 2 acima.

Braun e Petr (1994) dividem o gerenciamento de banda em dois grupos: *estático* e *dinâmico*. O gerenciamento estático não altera o seu comportamento através do tempo. Como não existe nenhum monitoramento do tráfego, o único controle feito nesse tipo de gerenciamento é não permitir a utilização da banda acima do valor previamente definido. O gerenciamento dinâmico altera a alocação da banda de acordo com a utilização da rede através do tempo. Este processo é feito monitorando o tráfego para determinar o seu comportamento ou recebendo requisições dos clientes da rede. De posse dessas informações, é possível alocar os recursos disponíveis da rede entre os elementos do sistema para acomodar todo o tráfego demandado. Em um gerenciamento dinâmico, se um usuário da rede aumenta a quantidade de tráfego sendo enviado ou requisita mais banda, o esquema de gerenciamento disponibiliza mais recursos de rede (se disponível) para o usuário. Por outro lado, se o usuário diminui a quantidade de tráfego sendo enviado ou requisita uma diminuição de banda, o algoritmo reduz a quantidade de recursos da rede associados ao usuário.

Nahrstedt, Shah e Chen (2005) listam as tarefas que são essenciais em um esquema de gerenciamento de banda:

- Monitoramento e estimativa da banda disponível: antes de distribuir a capacidade da rede entre os fluxos de comunicação, um esquema de gerenciamento de banda deve fazer uma estimativa da banda disponível;
- Sinalização: um esquema de gerenciamento de banda deve implementar um protocolo de sinalização para transportar o estado da alocação (estado da reserva de

banda) através da rede. A necessidade de um protocolo de sinalização em uma rede *ad hoc* (redes onde não existe um único ponto de acesso) é evidente, uma vez que os recursos devem ser reservados em cada nó do fluxo;

- Política de alocação de banda: uma vez que a banda e os requisitos dos fluxos são conhecidos, o próximo passo é alocar a banda para os fluxos. Algumas estratégias simplesmente adotam um política de fila enquanto outras estratégias adotam soluções mais complexas;

- Estado da alocação: baseado na política de alocação de banda, os fluxos são admitidos e uma parte da banda disponível é alocada;

- Controle da taxa: uma vez que a banda tenha sido alocada, os fluxos precisam ter suas taxas de comunicação restringidas conforme a quantidade de banda reservada para o fluxo;

- Adaptação: uma tarefa comum entre os esquemas de gerenciamento de banda é a adaptação. Essa funcionalidade permite que o gerenciamento da banda continue funcionando mesmo em ambientes dinâmicos.

Diversos pesquisadores têm trabalhado com o problema de gerenciamento de banda. Como exemplo de trabalhos encontrados nessa área, pode-se citar Choi e Huh (2002), que desenvolveram um algoritmo dinâmico de alocação de banda para serviços multimídia através de EPONs (*Ethernet Passive Optical Network*). Esse algoritmo gerencia vários tipos de tráfegos categorizando-os em três filas de prioridades: baixa, média e alta. Essas prioridades são então utilizadas pelo algoritmo para alocar a banda entre os fluxos de dados.

Hull, Jamieson e Balakrishnan (2003) estudaram o gerenciamento de banda em redes de sensores *wireless*. A arquitetura proposta no trabalho inclui um sistema de regras que permite que as aplicações e o administrador da rede especifiquem o quanto do tráfego gerado pelos sensores devem ser tratados pela rede.

Wang et al. (2004) propuseram um esquema de gerenciameno de banda baseado em QoS para redes wireless hererogêneas. O algoritmo desenvolvido decide como ajustar a QoS de forma a manter as conexões futuras em um nível satisfatório e com um uso eficiente dos recursos da rede.

2.2 Sistemas Multirrobôs

2.2.1 Conceitos e Aplicações

Os estudos em sistemas multirrobôs iniciaram por volta da década de 80 quando os pesquisadores começaram a investigar as questões relacionadas a sistemas com múltiplos robôs móveis. Antes disso, os pesquisadores se concentravam em sistemas envolvendo um único robô ou na solução de problemas distribuídos que não envolviam componentes de robótica (PARKER, 2000). De forma geral, um sistema multirrobô (SMR) pode ser caracterizado como um conjunto de robôs operando no mesmo ambiente (FARINELLI; IOCCHI; NARDI, 2004). Esses sistemas têm sido utilizados com sucesso em uma variedade de aplicações com diferentes objetivos de pesquisa. Atualmente, tem-se dado uma atenção especial para SMR desenvolvidos para operar em ambientes dinâmicos, onde incertezas e mudanças imprevistas podem ocorrer devido à presença dos robôs ou outros agentes externos ao sistema.

Podem ser encontrados na literatura diversos domínios de aplicações que utilizam SMR. Geralmente são utilizados em problemas que possuem tarefas de alta complexidade que devem ser coordenadas de forma a atingir o objetivo final da aplicação. Alguns exemplos são redes de sensores autônomos, transporte de grandes objetos, monitoramento de poluição aéreo e aquático, detecção de focos de incêndio em florestas, sistemas de transporte, busca e resgate em áreas de desastre ou qualquer conjunto de tarefas que podem ser subdivididas entre múltiplos robôs (FARINELLI; IOCCHI; NARDI, 2004; LIMA; CUSTODIO, 2005).

Nessas áreas de aplicação, SMRs podem frequentemente tratar tarefas que são difíceis, senão impossíveis, de serem tratadas por um único robô. Mesmos problemas que poderiam ser resolvidos por um único robô projetado para efetuar múltiplas tarefas podem se beneficiar do uso de um sistema multirrobô uma vez que a confiança e robustez da solução podem aumentar com a combinação de vários robôs (LIMA; CUSTODIO, 2005).

Comparado com um sistema onde exista um único robô atuando, existem diversas razões para a utilização de um SMR (ARAI; PAGELLO; PARKER, 2002):

- Uma tarefa pode ser inerentemente complexa demais para um único robô executar devido ao fato de um único robô estar limitado espacialmente;
- Um SMR pode prover redundância e contribuir cooperativamente para resolver as tarefas alocadas;

- Um SMR pode aumentar a eficiência tanto do ponto de vista da performance em completar certas tarefas quanto na robustez e na confiabilidade do sistema;
- Um SMR pode completar as tarefas atribuídas de forma mais confiável, rápida ou barata do que com a utilização de um único robô;
- Aumentar a performance do sistema como um todo.

Gerkey e Matarić (2004) propõe uma análise formal e uma taxonomia para sistemas multirrobôs focado no problema de alocação de tarefas (MRTA - Multi-Robot Task Allocation). De acordo com o trabalho, os problemas de MRTA podem ser classificados através de uma combinação de três agrupamentos:

- **Robôs de uma única tarefa - ST (*Single-Task robots*)** versus **Robôs multi-tarefas - MT (*Multi-Task robots*):** ST significa que cada robô é capaz de executar no máximo uma tarefa por vez, enquanto MT significa que alguns robôs conseguem executar múltiplas tarefas simultâneamente;
- **Tarefas de um único robô - SR (*Single-Robot tasks*)** versus **Tarefas multirrobôs - MR (*Multi-Robot tasks*):** SR significa que cada tarefa necessita de exatamente um robô para executá-la, enquanto MR significa que algumas tarefas podem necessitar de múltiplos robôs;
- **Atribuição instantânea - IA (*Instantaneous Assignment*)** versus **Atribuição de tempo estendido - TA (*Time-extended Assignment*):** IA significa que as informações disponíveis sobre os robôs, as tarefas e o ambiente permitem somente uma alocação instantânea das tarefas para os robôs, sem planejar alocações futuras. TA significa que existem mais informações disponíveis, tais como o conjunto de todas as tarefas que devem ser alocadas ou um modelo de como as tarefas são esperadas através do tempo.

Esses agrupamentos são utilizados para classificar os problemas de MRTA utilizando as abreviações acima. Por exemplo, um problema onde um grupo de robôs, que podem executar uma única tarefa, devem completar um conjunto de tarefas que necessitam de multirrobôs é classificado como um problema ST-MR-IA (*Single-Task robots/Multi-Robot tasks/Instantaneous Assignment*).

Arai, Pagello e Parker (2002) divide as pesquisas na área de SMR em sete principais áreas que geram significantes níveis de estudo:

- Inspiração biológica;
- Arquiteturas, alocação de tarefas e controle;
- Localização, mapeamento e exploração;
- Manipulação e transporte de objetos;
- Coordenação de movimento;
- Robôs reconfiguráveis; e
- Comunicação.

Nas pesquisas na área de inspiração biológica, as características sociais de insetos e animais são estudadas de forma a aplicar esses conhecimentos no projeto de sistemas multirrobôs. A aplicação mais comum desse tipo de conhecimento está no uso de regras de controle simples de várias sociedades biológicas - particularmente formigas, abelhas e pássaros - para o desenvolvimento de comportamentos similares em sistemas multirrobôs (BALCH; HYBINETTE, 2000; DORIGO et al., 2008; YINGYING; YAN; JINGPING, 2003).

Diversos pesquisadores de robótica distribuída tem focado no desenvolvimento de arquiteturas, planejamento de tarefas e controle. Essa área de pesquisa trata dos problemas relativos à seleção da ação; delegação da autoridade e controle; a estrutura de comunicação; sistemas heterogêneos e homogêneos de robôs; resolução de conflitos e outras questões relacionadas (GERKEY; MATARIĆ, 2004; BOTELHO; ALAMI, 1999; PARKER, 1996; GERKEY; MATARIC, 2003).

A área de localização, mapeamento e exploração em SMR trata esses conceitos de forma distribuída utilizando um time de robôs enquanto a manipulação e transporte de objetos permite que diversos robôs carreguem, empurrem ou manipulem objetos de forma cooperativa (THRUN; BURGARD; FOX, 2000; BASTOS; RIBEIRO; SOUZA, 2008; FOX et al., 2000; HOWARD, 2006). Estas áreas de pesquisa têm um grande número de aplicações práticas que fazem delas interessantes áreas de estudo.

Outro tópico de estudo popular em SMR é a coordenação de movimento. Temas de pesquisa nesse domínio têm sido estudados incluindo planejamento de caminho, controle de tráfego e geração e manutenção de uma formação do time de robôs (ŠVESTKA; OVERMARS, 1998; LUMELSKY; HARINARAYAN, 1997; FERRARI et al., 1998; YAMASHITA et al., 2000).

Estudos na área de robôs reconfiguráveis têm resultado em sistemas de robôs que possuem a capacidade física de serem reconfigurados, permitindo que módulos individuais ou robôs se conectem entre si de diversos modos para gerar formas que atendam a uma necessidade específica. Esses sistemas possuem a capacidade teórica de atingir grande robustez, versatilidade e até auto-reparo (ARAI; PAGELLO; PARKER, 2002; FUKUDA; NAKAGAWA, 1988; BENI, 1988).

2.2.2 Comunicação

A comunicação entre times de robôs tem sido estudada desde a concepção das pesquisas em robótica distribuída. Muitos pesquisadores têm estudado os efeitos da comunicação na performance de multirrobôs em uma variedade de tarefas e têm concluído que a comunicação pode prover certos benefícios em alguns tipos de tarefas como coleta de lixo, coleta de amostras em ambientes de risco, montagem, tarefas de limpeza, etc (MACLENNAN, 1990; BALCH; ARKIN, 1994). Adicionalmente, Balch e Arkin (1994) mostram que, em muitos casos, a comunicação de pequenas quantidades de informações podem trazer grandes benefícios tornando a execução das tarefas mais eficientes.

Projetistas de sistemas multirrobôs precisam considerar cada componente do projeto. A inclusão de sensores, atuadores ou robôs adicionais precisam contribuir com eficiência na execução das tarefas. Os componentes que não contribuem diretamente podem gerar um *overhead* na comunicação e adicionar um custo sem gerar benefícios para a aplicação. A questão não é apenas se a comunicação entre robôs deve ser incluída, mas que tipo de comunicação, velocidade, complexidade e estrutura deve ser considerada (BALCH; ARKIN, 1994).

(ARKIN, 1989) e (ARKIN et al., 1993) propuseram uma metodologia para considerar a inserção de novos componentes em sistemas com um único robô e em SMR com dois pontos principais que devem ser considerados:

1. A criação de uma métrica objetiva para a performance do sistema;
2. Um ciclo iterativo de simulações e a utilização da solução em sistemas reais.

Através da simulação, o projetista pode descobrir rapidamente quais sensores, atuadores e parâmetros de controle são mais críticos. A meta é encontrar um sistema que maximiza (ou minimiza) a métrica da performance definida no item 1. Finalmente, a configuração é utilizada em um sistema real onde as conclusões são validadas.

Em um SMR, a comunicação pode aumentar exponencialmente com aumento do número de robôs. Klavins (2004) estudou a complexidade da comunicação em SMR e, em seu modelo, cada robô conhece seu próprio estado e pode se comunicar diretamente com os outros elementos do sistema através de algum mecanismo de comunicação. O custo da comunicação, em termos de banda utilizada, latência ou tempo de espera, é sumarizado como um único custo. Em cada instante, os custos são somados de forma a gerar uma métrica do custo da comunicação naquele instante. A complexidade de qualquer algoritmo de comunicação, no seu pior caso, é $O(n^2)$: todos os robôs precisam se comunicar constantemente com todos os outros robôs. Com uma complexidade de $O(n^2)$, a banda disponível para a comunicação precisa aumentar em proporção quadrática com o número de robôs do sistema. Por esse motivo, soluções com complexidade de $O(n^2)$ não são considerados escaláveis.

Arai, Pagello e Parker (2002) apresentam diversos trabalhos na área de comunicação em SMR. Alguns trabalhos têm focado na representação de linguagens e na fundamentação dessas linguagens no mundo real. A tolerância a faltas na comunicação entre multirrobôs tem sido estudada em algumas pesquisas através da configuração e manutenção de redes de comunicação distribuídas e da garantia da confiabilidade da comunicação em SMR.

Shen, Salemi e Will (2002) examinam o uso de comunicação adaptativa em robôs modulares e reconfiguráveis. O principal desafio desses sistemas é manter a comunicação mesmo quando as conexões entre os robôs podem mudar dinamicamente e de forma inesperada. Rybski et al. (2002) exploram em seu trabalho a comunicação em um time de robôs em miniatura que precisam utilizar comunicação via rádio frequência (RF) de baixa capacidade devido ao tamanho reduzido dos robôs.

Como pode ser visto, a comunicação está presente na maioria das áreas de estudo em SMR e pode influenciar diretamente na qualidade da solução. A largura de banda é um recurso importante de alguns sistemas de comunicação em SMR que deve ser levado em consideração. A má utilização desse recurso pode acarretar em perda de pacotes ou atraso na entrega prejudicando a comunicação.

2.2.3 Gerenciamento de Banda

Diferentes tipos de algoritmos de gerenciamento de recusos têm sido utilizados de forma a resolver o problema de alocação de banda em SMR. Tais aplicações podem ser categorizadas em dois grupos principais: estático (CHASKAR; MADHOW, 2003) e dinâmico (TIPSUWAN et al., 2009). Métodos estáticos não podem se adaptar às mudanças

no ambiente enquanto os métodos dinâmicos melhoram a performance mas com um custo adicional de tempo de computação ou da complexidade da solução (MANSOUR et al., 2011).

Mourikis e Roumeliotis (2006) tratam do problema de alocação de recursos em formações de robôs móveis. O principal objetivo é determinar a frequência em que cada sensor deve enviar os dados de forma a obter a melhor precisão possível na localização dos robôs. As frequências são obtidas através da solução de um problema de otimização que maximiza a matriz de precisão expressada em termos das frequências dos sensores. Entretanto, o problema é resolvido de forma *offline* e o algoritmo não leva em consideração nenhum evento dinâmico que possa ocorrer no ambiente.

Sugiyama, Tsujioka e Murata (2009) propõe em seu trabalho um algoritmo de gerenciamento de banda para SMR em uma missão de monitoramento de alvos. A informação importante para a aplicação, que é a detecção de sobreviventes, precisa ser enviada para uma estação base através de imagens, por exemplo, onde um operador humano decide se a imagem indica ou não uma vítima real. Nessa abordagem, a decisão do operador é crucial na alocação da banda e, desse modo, a decisão não é totalmente automatizada.

Alguns trabalhos em QoS abordam o problema de gerenciamento de banda para a transmissão de dados entre elementos de um SMR. O gerenciamento de banda pode ser feito, de forma menos específica, utilizando a tecnologia disponível para gerenciamento de fluxos de pacotes TCP/IP. Em Sugiyama, Tsujioka e Murata (2006), uma estrutura modificada do *frame* em um roteamento sincronizado com QoS (como estudado em Perkins e Bhagwat (1994)) é proposto em uma rede *ad hoc* aplicada em um SMR para detecção de vítimas em áreas urbanas de desastres.

Mansour et al. (2011) propuseram um algoritmo de gerenciamento de banda baseado nas condições das tarefas. Em outras palavras, se nada de relevante para a tarefa está ocorrendo no ambiente, a taxa de comunicação é diminuída. Entretanto, o algoritmo implementado não impõe nenhuma restrição no total de banda do sistema, apenas demostrando que uma maior taxa de comunicação é necessária para manter o mesmo nível de performance obtido quando o ambiente é menos dinâmico.

Finalmente, Mansour et al. (2012) estudaram o gerenciamento de banda através de diferentes canais de comunicação em um sistema de teleoperação utilizando multirrobôs. O trabalho foca em manter o mesmo nível de performance na execução da tarefa de teleoperação e na colaboração dos robôs mesmo quando existem restrições na rede. O gerenciamento de banda é feito levando em consideração os eventos no ambiente e a

qualidade da colaboração entre os robôs.

3 ROS - Robot Operating System

O desenvolvimento de software para a área da robótica possui diversos desafios. A reutilização de códigos ao escrever softwares para robôs é uma tarefa difícil devido à grande variedade de hardware. Essa complexidade cresce a medida em que a diversidade de robôs utilizados no sistema também cresce. Um sistema para robótica inclui diversas áreas do desenvolvimento de software, indo do nível do desenvolvimento de drivers para a comunicação com o hardware até o desenvolvimento de aplicações de alto nível de abstração. Uma vez que o conhecimento dessas diversas áreas extrapola as capacidades de um único pesquisador, é necessário um esforço de integração com diversos outros softwares.

Com o intuito de facilitar as pesquisas na área, diversos pesquisadores têm construído ambientes para o desenvolvimento de software para robôs (RDE - *Robot Development Environment*) (KRAMER; SCHEUTZ, 2007). Esses RDEs auxiliam desde a arquitetura do sistema até a integração com o hardware dos robôs. Cada um desses *frameworks* foi escrito com um propósito em particular, talvez como resposta a um ponto fraco de um outro *framework* ou para dar ênfase a um ponto específico do sistema em desenvolvimento.

O ROS (*Robot Operating System*) é um *framework* flexível para o desenvolvimento de software para robôs com o objetivo de promover o desenvolvimento colaborativo de software para a área da robótica. Como exemplo, um laboratório de pesquisa com habilidades em desenvolvimento de software para o mapeamento de ambientes internos pode criar um sistema utilizando a arquitetura do ROS e disponibilizar para que outros grupos de pesquisadores o utilizem em suas pesquisas (QUIGLEY et al., 2009).

O *framework* foi originalmente desenvolvido em 2007 pelo *Stanford Artificial Intelligency Laboratory* (SAIL) com o suporte do *Stanford AI Robot Project* e provê as facilidades de um sistema operacional padrão como a abstração de hardware, controle de dispositivos de baixo nível, implementação de funcionalidades largamente utilizadas, trocas de mensagens entre processos e gerenciamento de pacotes (MARTINEZ; FERNÁNDEZ, 2013).

De acordo com Quigley et al. (2009), o ROS foi desenvolvido seguindo algumas metas específicas como arquitetura *peer-to-peer*, baseado em ferramentas, multilingual, pequeno, gratuito e *open-source*.

Um sistema construído em ROS consiste de um conjunto de processos, que podem ser executados em diferentes *hosts*, conectados em uma topologia *peer-to-peer*. Essa topologia permite resolver diversos problemas encontrados em *frameworks* baseados em um servidor central (QUIGLEY et al., 2009). Devido aos programadores terem preferências distintas de linguagens de programação, o ROS foi projetado para suportar diversas linguagens. Atualmente o *framework* suporta quatro liguagens distintas: C++, Python, Octave e LISP. Utilizando uma linguagem de definição de interfaces (IDL) para descrever as mensagens enviadas entre os módulos, é simples escrever um gerador de código para uma linguagem de programação que gere objetos específicos dessa linguagem. Esses objetos são automaticamente serializados e desserializados pelo ROS ao enviar e receber mensagens entre os módulos. Como resultado, tem-se um sistema que permite que diferentes linguagens sejam utilizadas em um mesmo software.

O *framework* consiste em um *microkernel* e um grande conjunto de ferramentas que são utilizadas para construir e executar os diversos componentes do ROS. Essa separação faz com que o ROS seja baseado em ferramentas e facilite o gerenciamento da complexidade do sistema. Essas ferramentas implementam diversas funcionalidades como navegação pela estrutura do código fonte, ajuste de parâmetros de configuração, visualização da topologia *peer-to-peer*, medição da utilização de banda, entre outros.

O ROS foi desenvolvido utilizando uma arquitetura que permite que partes complexas como os *drivers* possam ser encapsuladas em bibliotecas que não possuam dependência com o *framework*. Essa separação da complexidade e a utilização de bibliotecas de código aberto fazem com que o ROS seja pequeno e simples de manter.

Segundo Martinez e Fernández (2013), a arquitetura do ROS pode ser dividida em três seções ou níveis de conceitos: o sistema de arquivos, o grafo de computação e a comunidade. O sistema de arquivos é responsável por descrever como o ROS é internamente formado, a estrutura de pastas e os arquivos mínimos para o funcionamento do *framework*. O grafo de computação é onde a comunicação entre processos e sistemas ocorre. O terceiro nível de conceito é onde se explica as ferramentas e conceitos para o compartilhamento de conhecimento, algoritmos e código de qualquer desenvolvedor. Esse nível é importante para que o ROS possa crescer com a ajuda da comunidade.

3.1 Sistema de Arquivos

O sistema de arquivos cobre, principalmente, os recursos do ROS que são encontrados no disco, como apresentados na figura 3.1:

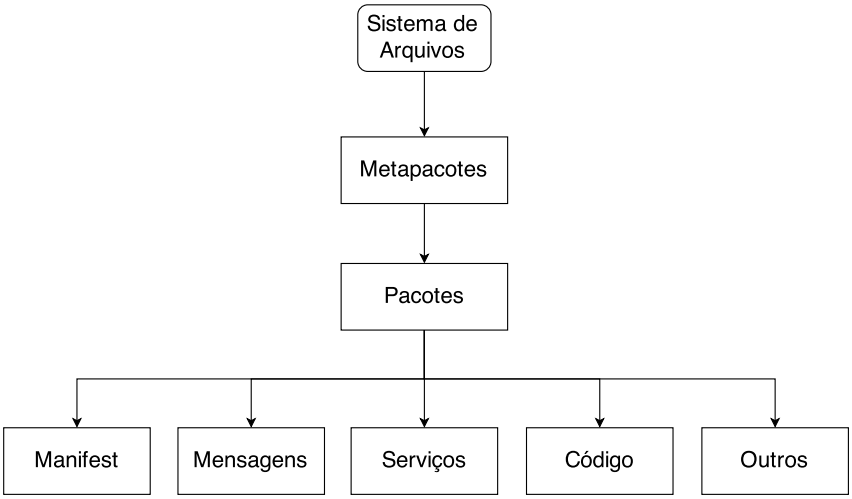

Figura 3.1: Sistema de arquivos do ROS.

- Pacotes: Pacotes são as principais unidades de organização de software em ROS. Um pacote pode conter executáveis ROS (nós), bibliotecas ROS, conjuntos de dados, arquivos de configuração ou qualquer outro recurso necessário.
- Metapacotes: São pacotes especializados que servem para agrupar um conjunto de pacotes relacionados entre si.
- *Package Manifests*: *Manifests* (package.xml) contém meta-dados sobre os pacotes como nome, versão, descrição, informação sobre licença, e dependências.
- Tipos de mensagens (msg): Descrição das mensagens. Definem a estrutura de dados das mensagens enviadas pelo ROS.
- Tipos de serviços (srv): Descrição dos serviços. Definem a estrutura de dados das requisições e respostas dos serviços disponibilizados pelo ROS.

3.2 Grafo de Computação

O grafo de computação é uma rede ponto-a-ponto de processos ROS que estão processando dados juntos. Os conceitos básicos do grafo de computação são os nós, *Master*,

Parameter Server, mensagens, serviços, tópicos e *bags*. Todos esses conceitos proveem dados para o grafo em diferentes modos. A figura 3.2 mostra os elementos que fazem parte do grafo de computação do ROS.

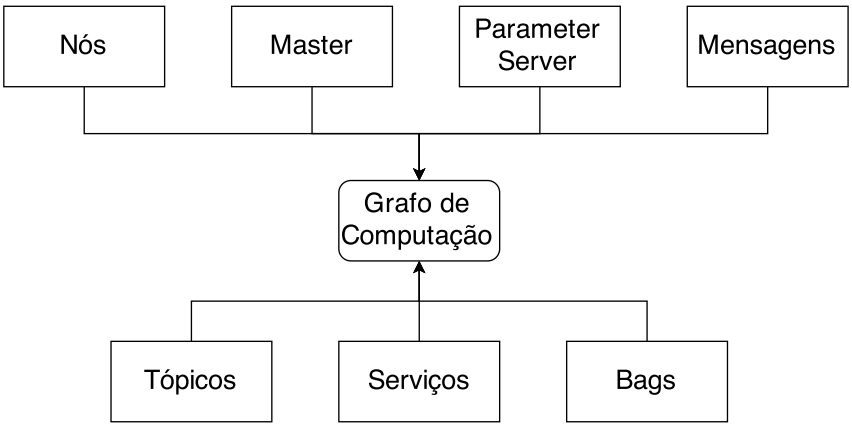

Figura 3.2: Grafo de computação do ROS.

- Nós: São processos que executam a computação. O ROS é projetado para ser modular e possuir diversos módulos pequenos, ou seja, um sistema desenvolvido em ROS geralmente possui diversos nós. Por exemplo, um nó faz medições do sensor laser, outro controla a movimentação, outro faz o planejamento de caminho, etc. Um nó é escrito utilizando uma biblioteca cliente do ROS como roscpp (C++) ou o rospy (Python).

- *Master*: O ROS *Master* é responsável pelo registro e pesquisa de nomes para o resto do grafo de computação. Sem o *Master*, os nós não conseguiriam encontrar os outros nós do grafo, trocar mensagens ou invocar serviços.

- *Parameter Server*: O *Parameter Server* é um sub-módulo do *Master* que permite que dados sejam armazenados na forma de chave-valor em um local central disponível para todos os outros nós da rede.

- Mensagens: Os nós comunicam entre si através de mensagens. Uma mensagem é uma simples estrutura de dados contendo campos tipados. Tipos primitivos de dados como *integer*, *float*, *boolean*, etc e *arrays* desses tipos primitivos são suportados como tipos dos campos das mensagens.

- Tópicos: As mensagens são enviadas através de um sistema baseado no padrão *publish - subscribe*. Um nó envia uma mensagem para outro nó publicando em um dado tópico. Um tópico é um nome que é usado para identificar o conteúdo da

mensagem. Um nó que deseja receber um tipo de dado irá se inscrever em um tópico apropriado e pode se inscrever e publicar diversos tópicos. Os tópicos tem como objetivo desacoplar a produção de informação de seu consumo. Um nó que publica uma dada informação não precisa conhecer os nós que estão inscritos para receber esse dado.

- Serviços: Os serviços são utilizados para comunicações do tipo requisição-resposta e são definidos através de um par de mensagens (uma para a requisição e outra para a resposta). Um nó que esteja oferecendo um serviço, disponibiliza-o através de um nome e o cliente utiliza o serviço enviando uma mensagem de requisição e aguardando uma resposta.

- *Bags*: *Bags* são um formato para salvar e reproduzir dados de mensagens enviadas através do ROS. *Bags* são um importante mecanismo para armazenar dados, tais como dados de sensores, que são difíceis de coletar em ambientes simulados, mas necessários para o desenvolvimento e teste de algoritmos.

Segundo a ROS-Wiki (2015), o *Master* trabalha como um servidor de nomes no grafo de computação do ROS armazenando as informações necessárias sempre que um nó registra em algum tópico ou serviço. Os nós comunicam com o *Master* de forma a reportar qualquer informação de registro. Devido ao fato de se comunicarem com o *Master*, os nós podem receber informações sobre outros nós e criar conexões sempre que for apropriado. O *Master* também é responsável por efetuar chamadas de *callbacks* para os outros nós da rede toda vez que as informações de registro são alteradas, permitindo que os nós criem conexões dinâmicamente com novos nós.

Os nós se comunicam com os outros nós diretamente; o *Master* apenas provê as informações de conexão como um servidor de DNS. Sempre que um nó se inscreve em um tópico, a sua conexão com o nó que publica o tópico é estabelecida através de um protocolo de comunicação. O protocolo mais utilizado no ROS é o TCPROS, que usa sockets padrão TCP/IP.

Essa arquitetura permite o desacoplamento da comunicação. Como exemplo desse desacoplamento, pode-se considerar um nó *camera_publisher_node* que lê as informações de uma câmera de vídeo instalada no robô e publica essas informações em um tópico *camera*. Durante a criação do tópico, o *camera_publisher_node* informa os dados do tópico para o *Master*. Como ainda não existe nenhum cliente inscrito no tópico nenhuma conexão é feita. Se um outro nó executando em uma central (*camera_reader_node*) se

inscrever nesse tópico, as informações são solicitadas para o *Master* e uma conexão entre os dois nós é estabelecida. O *camera_reader_node* passa então a receber os dados da câmera.

3.2.1 Convenção de Nomes

O ROS possui uma estrutura de nomenclatura hierarquizada que é utilizada em todos os recursos do grafo de computação tais como nós, parâmetros, tópicos e serviços. Essa nomenclatura é a base para sistemas maiores e mais complexos evitando conflito de recursos e provendo encapsulamento. Alguns exemplos de nomes utilizados no ROS são:

- / (namespace global)
- /camera
- /unifei/robot/name
- /wg/node1

De forma a conseguir um melhor encapsulamento, cada recurso é definido dentro de um *namespace* que pode ser compartilhado com outros recursos. Em geral, os recursos podem criar outros recursos dentro do seu *namespace* e podem acessar outros recursos que estão acima do seu *namespace*.

Os nomes são resolvidos de forma relativa para que os recursos não precisem identificar em qual *namespace* se encontram. Isso simplifica a programação e evita conflito de nomes quando sistemas são integrados. Como exemplo, pode-se considerar que dois sistemas com *namespace /unifei* e */stanford* possuem um nó com nome *camera*. Se esses dois sistemas forem integrados, o conflito de nomes dos nós é evitado pelo *namespace*. Os nós terão como nomes */unifei/camera* e */stanford/camera*, respectivamente.

Um nome válido deve começar com um caracter alfa ([a-z|A-Z]), til (~) ou barra (/). Os caracteres subsequentes devem ser alfanuméricos ([0-9|a-z|A-Z]), sublinhados (_) ou barras (/).

3.2.2 Nó

Segundo Martinez e Fernández (2013), nós são programas executáveis que podem se comunicar com outros processos utilizando tópicos, serviços ou o *Parameter Service*. A

utilização de nós no ROS provê tolerância a falhas e uma melhor separação do código e das funcionalidades, simplificando o sistema. Um nó precisa ter um nome único no sistema, o qual é utilizado para permitir a comunicação do nó com qualquer outro nó do sistema utilizando seu nome sem nenhum conflito de nomes e ambiguidade. Um nó pode ser escrito utilizando diversas bibliotecas como *roscpp* (C++) e *ropsy* (Python).

Os nós foram projetados para operar de forma que cada parte do sistema seja implementado utilizando um nó diferente. Cada pequena parte do sistema deve ser desenvolvida utilizando um nó. Esse comportamento permite que erros sejam isolados no nó e não afetem todo o sistema. Como exemplo, um nó lê os dados de um sensor laser *rangefinder*, enquanto outro nó pode controlar as rodas do robô, tratar da localização, ou por exemplo, executar o planejamento do caminho a ser seguido (ROS-WIKI, 2015).

3.2.3 Tópico

Definido inicialmente por Gamma et al. (1994), o padrão *publisher-subscriber* define um padrão de envio de mensagens onde os elementos que publicam as mensagens, chamados de *publishers* (publicadores), não programam o envio das mensagens diretamente para elementos específicos do sistema, chamados *subscribers* (assinantes). Ao invés disso, as mensagens são enviadas para *containers* que tratam de entregar essas mensagens para qualquer *suscriber* interessado. A principal vantagem desse tipo de implementação é o desacoplamento entre *publisher* e *susbcribers*: um *publisher* pode publicar mensagens sem que exista *subscribers* e vice-versa.

Segundo a ROS-Wiki (2015), tópicos são *containers* nomeados através dos quais os nós trocam mensagens. Os tópicos possuem uma arquitetura baseada no padrão *publisher-subscriber*, o que desacopla a produção da informação de seu consumo. Em geral, os nós são se preocupam com quem eles estão se comunicando. Ao invés disso, os nós se inscrevem em um determinado tópico quando estão interessados em um dado ou publicam em um tópico relevante os dados que deve ser externados. Podem existir múltiplos *publishers* e *subscribers* em um mesmo topic.

Cada tópico é fortemente tipado pelo tipo de mensagem ROS publicado e os nós só podem receber mensagens se o tipo for condizente com o tipo de mensagem do tópico. O *Master* não força qualquer tipo de consistência entre *publishers*, mas os *subscribers* somente irão estabeler o transporte das mensagens se os tipos forem iguais.

3.2.4 Serviço

O modelo *publisher-subscriber* é um paradigma de comunicação extremamente flexível mas seu esquema de transporte não é apropriado para chamadas do tipo RPC (*Remote Procedure Call*) ou requisição/resposta, os quais são frequentemente utilizados em sistemas distribuídos. Esse tipo de interação é implementado através de serviços no ROS. Um serviço é definido por um nome e um par de mensagens: uma para a requisição e outra para a resposta. Um cliente chama um serviço enviando uma mensagem de requisição e esperando uma mensagem de resposta. Essa comunicação é implementada para o programador como se fosse uma RPC.

3.2.5 Master

O ROS *Master* é responsável por fornecer o serviço de nome e registro para o resto dos nós do grafo de computação do ROS. É através do *Master* que os *publishers* e *subscribers* adiquirem informações sobre os tópicos e os serviços disponíveis na aplicação. A principal funcionalidade do *Master* é permitir que os nós localizem outros nós presentes no sistema. Uma vez que um nó localize outro nó na rede, a comunicação é feita de forma *peer-to-peer*.

O *Master* é implementado utilizando XMLRPC, que é um protocolo para a chamada de procedimentos remotos (RPC) baseado em HTTP que utiliza XML para descrever os procedimentos (LAURENT et al., 2001).

De forma a ilustrar o serviço de gerenciamento de *hosts* e nomes disponibilizado pelo *Master*, pode-se considerar, como exemplificado por ROS-Wiki (2015), um sistema com dois nós: um *camera_node* e um *image_viewer_node*. Uma sequência típica de eventos irá iniciar com *camera_node* notificando o *Master* a intenção de publicar imagens no tópico *images* (Figura 3.3).

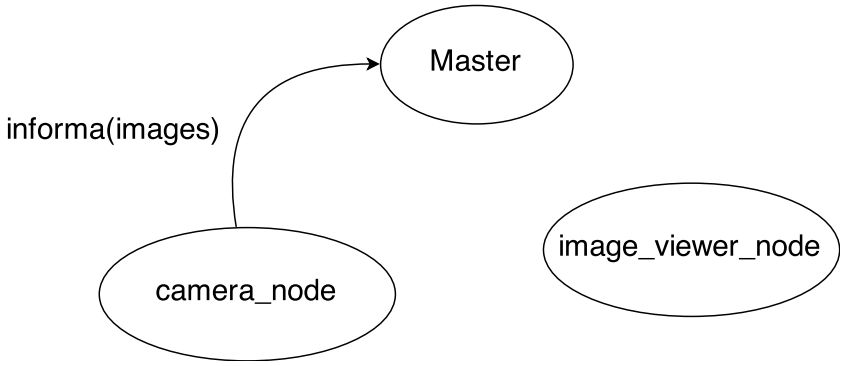

Figura 3.3: Criação do tópico *images* pelo *camera_node*.

Após a criação do tópico, o *camera_node* pode publicar imagens no tópico *images*. Mas como não existem nós inscritos no tópico, nenhum dado é enviado na rede. Como próximo passo, o *image_viewer_node* se inscreve no tópico *images* para receber as informações da camera, como pode ser visto na Figura 3.4.

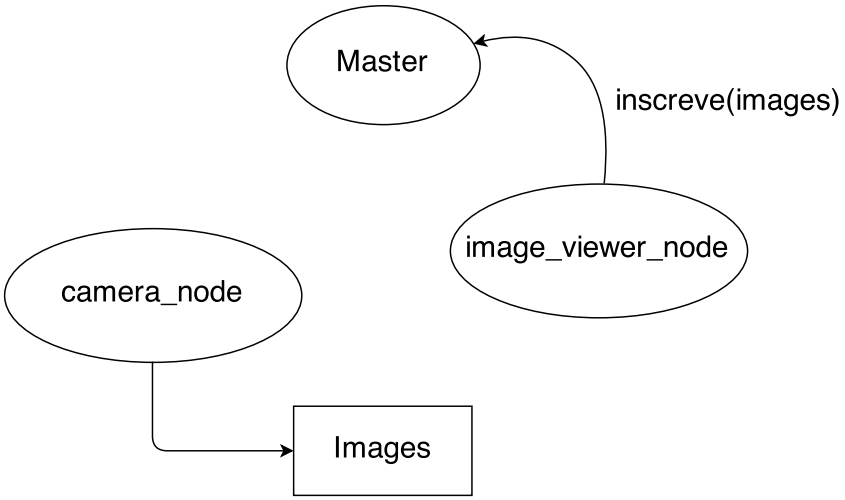

Figura 3.4: Inscrição do *image_viewer_node* no tópico *images*.

Considerando que o tópico *images* tem um *publisher* e um *subscriber*, o *Master* notifica os nós *camera_node* e *image_viewer_node* sobre a existência de ambos para que o dado da câmera possa ser transferido. A Figura 3.5 mostra essa comunicação entre os nodes.

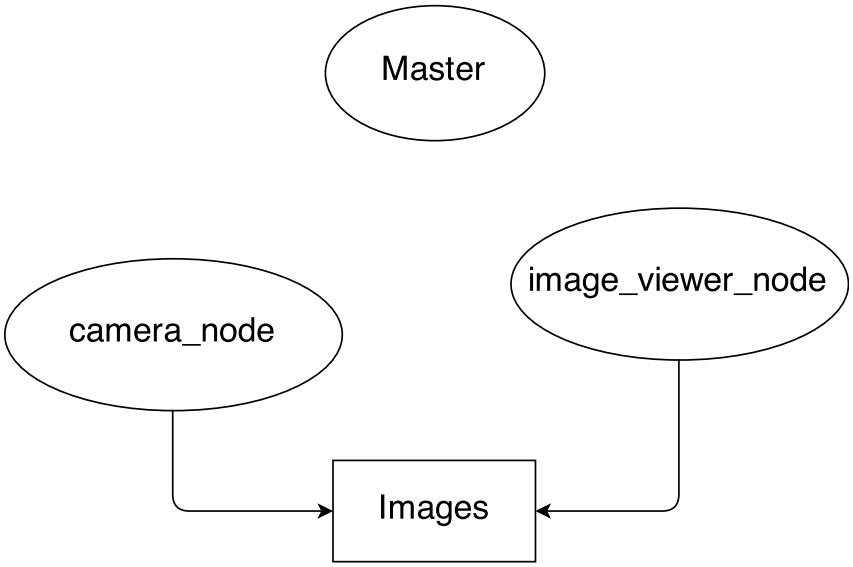

Figura 3.5: Comunicação entre nós via topic.

3.2.6 Parameter Server

De acordo com Martinez e Fernández (2013), o *Parameter Server* é um dicionário multivariável e compartilhado que é acessível pela rede. Os nós utilizam esse servidor para armazenar e recuperar os parâmetros em tempo de execução. O *Parameter Server* é implementado dentro do ROS *Master* e não foi projetado para alta performance, sendo melhor utilizado para dados estáticos e não binários como parâmetros de configuração (ROS-WIKI, 2015).

Segundo a ROS-Wiki (2015), os parâmetros são nomeados utilizando a convenção de nomes do ROS. Isto significa que os parâmetros possuem uma hierarquia. Esta hierarquia é importante para evitar uma colisão de nomes e permitir que os parâmetros sejam acessados individualmente ou na forma de uma árvore. Por exemplo, para os parâmetros seguintes:

/camera/left/name: leftcamera
/camera/left/exposure: 1
/camera/right/name: rightcamera
/camera/right/exposure: 1.1

O parâmetros */camera/left/name* possui o valor *leftcamera*. É possível recuperar o valor do parâmetro */camera/left* que possui como valor o dicionário:

name: leftcamera

exposure: 1

Também é possível recuperar o valor para */camera* que possui um dicionário de dicionários representando a árvore do parâmetro:

left: name: leftcamera, exposure: 1
right: name: rightcamera, exposure: 1.1

O *Parameter Server* utiliza tipos de dados XMLRPC para os tipos dos parâmetros:

- 32-bit intergers;
- Booleans;
- Strings;
- Doubles;
- ISO 8601 dates;
- Base 64-encoded binary data.

3.3 Comunicação

Em um sistema desenvolvido em ROS, a comunicação entre os diversos agentes pode ser feita através de tópicos e serviços. Essas duas formas de comunicação possuem objetivos distintos e são utilizadas em contextos diferentes.

Uma comunicação baseada em tópicos utiliza o esquema de comunicação chamado *publisher-subscriber*, oferecendo um baixo acoplamento entre o agente que envia a mensagem e o que recebe. Dessa forma, a produção de informação não depende diretamente do seu consumo.

O tópico é apenas um nome que é utilizado para identificar o conteúdo da mensagem. Em uma comunicação baseada no padrão *publisher-subscriber*, um nó envia uma mensagem publicando em um determinado tópico e somente quem está interessado naquele tipo de dado se inscreve, passando assim a receber a informação publicada (figura 3.6).

A comunicação por tópicos é utilizada em fluxos de dados contínuos (dados sensoriais, estado do robô, etc). Os dados podem ser publicados e observados a qualquer momento independente de quem envia e recebe esses dados. Pode haver múltiplos publicadores e

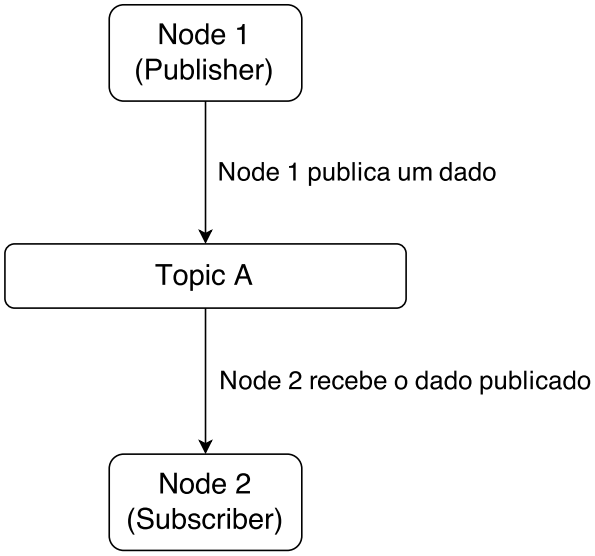

Figura 3.6: Exemplo de um tópico.

assinantes em um mesmo tópico e um único nó pode publicar e se inscrever em quantos tópicos seja necessário.

Os tópicos apresentam um modelo de comunicação de uma via, onde um nó publica a informação e o outro recebe. Dessa forma, esse tipo de paradigma baseado no *publisher-subscriber* não é apropriado para uma interação do tipo requisição-resposta, onde um nó requisita uma ação e aguarda uma resposta. Esse tipo de interação pode ser feito utilizando os serviços em ROS.

Os serviços são definidos por uma estrutura contendo um par de mensagens: uma para a requisição e outra para a resposta. Um nó oferece um serviço através de um nome e um cliente utiliza esse serviço enviando uma mensagem de requisição e aguardando uma resposta. As bibliotecas do ROS geralmente apresentam essa interação para o programador como se fosse uma chamada de procedimento remota (RPC).

A comunicação utilizando serviços deve ser utilizada em chamadas de procedimento remoto que terminam rapidamente, como por exemplo, para consultar o estado de um nó ou computar cálculos rápidos. Os serviços nunca devem ser utilizados em processos de execução mais longos, principalmente quando existe a necessidade de interrupção da tarefa (preempção) em situações excepcionais.

Em casos que o serviço leva um tempo considerável para ser executado, pode ser desejável que se possa cancelar a execução durante a requisição ou receber *feedbacks* periódicos sobre o seu andamento. Para esse caso especial de comunicação do tipo requisição-

resposta, o ROS oferece a *ActionLib*. A *ActionLib* é um pacote que oferece uma interface padronizada de comunicação do tipo requisição-resposta para a utilização em tarefas preemptivas. Exemplos desse tipo de tarefa inclui mover o robô de uma origem para um destino, executando um escaneamento por laser e retornando um *feedback* de posição e os obstáculos encontrados durante a trajetória.

De forma resumida, as ferramentas para a troca de mensagens em ROS pode ser definida por:

- Tópicos
  - Baseado no padrão *publisher-subscriber*;
  - Comunicação de muitos para muitos;
  - Utilizado em fluxos de dados contínuos como dados sensoriais, estado do robô, etc.
- Serviços
  - Baseado no padrão requisição-resposta;
  - Comunicação de um para um;
  - Utilizado em chamadas de procedimento remoto com execução de curta duração como consultar o estado de um nó ou computar cálculos rápidos.
- *ActionLib*
  - Baseado no padrão requisição-resposta;
  - Comunicação de um para um;
  - Utilizado na execução de tarefas que necessitem de preempção como cancelar a execução durante uma requisição ou receber *feedbacks* periódicos.

Existem diversas formas de transmitir dados em uma rede, sendo que cada uma possui vantagens e desvantagens dependendo da aplicação. O TCP é um protocolo bastante utilizado por ser simples e confiável. Os pacotes TCP sempre chegam em ordem e caso haja perda de pacotes, eles são reenviados até que os pacotes cheguem ao seu destino. Por outro lado, a utilização de TCP em redes *WiFi* com altas taxas de perda e em redes mobile pode causar problemas. Nesse tipo de rede, o UDP é mais apropriado. Se existirem muitos *subscribers* agrupados em uma única subrede, pode ser mais eficiente

para o *publisher* comunicar com todos eles simultaneamente através de um *broadcast* UDP (ROS-WIKI, 2015).

A utilização de TCP ou UDP depende do tipo de aplicação. O TCP é um protocolo de comunicação confiável onde existe garantia na entrega da mensagem. Quando uma mensagem (ou parte dela) é perdida e não chega no seu destino, essa mensagem é reenviada até que a troca de informações seja completada com sucesso. Por outro lado, quando o UDP detecta que a mensagem contém um valor não válido, o mesmo é descartado, e não reenviado como no caso do TCP. Essa é uma questão importante em diversos tipos de aplicações onde a garantia de entrega não é importante. No caso da transmissão de dados de sensores em robótica, por exemplo, a perda de alguns pacotes não causa grandes mudanças no valor médio da leitura.

Por essas razões, o ROS não utiliza apenas um protocolo de transporte. O nó que está se inscrevendo negocia a conexão utilizando o protocolo de transporte apropriado com o nó que está publicando os dados. O resultado da negociação é que os dois nós são conectados e prontos para a transferência dos dados como *publisher* e *subscriber*.

O ROS possui duas camadas de transporte básicas: TCPROS e UDPROS. O TCPROS é uma camada de transporte padrão para as mensagens e serviços do ROS. Ele utiliza sockets TCP/IP para transportar os dados das mensagens. O UDPROS utiliza datagramas UDP para transportar a mensagem serializada e é extremamente útil quando a latência é mais importante que a confiança dos dados. Exemplos de utilização na robótica incluem aplicações de teleoperação e envio de streaming de audio (ROS-WIKI, 2015).

Cada protocolo de transporte possui sua própria forma de como os dados são trocados. Por exemplo, utilizando TCP, a negociação envolve o *publisher* fornecenedo para o *subscriber* o endereço de IP e a porta de conexão. O *subscriber* cria então um *socket* TCP/IP. Os nós trocam um cabeçalho de conexão que inclui informações como a soma MD5 do tipo de mensagem e o nome do tópico. Depois disso, o *publisher* está pronto para enviar a mensagem serializada diretamente através do *socket*.

É importante enfatizar que os nós comunicam entre si diretamente através do mecanismo de transporte apropriado. Os dados não são roteados através do *Master* e nem enviados via XMLRPC, que é utilizado apenas para negociar a conexão entre os nodes.

De forma a ilustrar como é feita a conexão entre dois nós antes da troca de informações entre eles, considere o exemplo mostrado anteriormente onde um nó *camera_node* (*publisher*) publica as imagens da câmera de um robô através do tópico *images*. Um

outro nó *image_viewer_node* (*subscriber*) se inscreve nesse tópico e recebe as imagens da câmera. A figura 3.7 mostra essa conexão entre os nós.

1. O nó *camera_node* inicia e registra no *Master* utilizando XMLRPC informando ao *Master* que deseja publicar o tópico *images*

2. O nó *image_viewer_node* inicia e registra no *Master* utilizando XMLRPC informando ao *Master* que deseja receber informações do tópico *images*

3. O *Master* informa o nó *image_viewer_node* (*subscriber*) do novo publicador do tópico *images* (node *camera_node*)

4. Utilizando XMLRPC, o nó *image_viewer_node* envia uma mensagem para o nó *camera_node* requisitando uma conexão e negocia o protocolo de transporte (TCPROS ou UDPROS)

5. O nó *camera_node*) envia para o nó *image_viewer_node* as configurações necessárias para a conexão utilizando o protocolo selecionado. Essa comunicação é feita utilizando XMLRPC.

6. O nó *image_viewer_node* conecta com o nó *camera_node* e efetua a troca de mensagens do tópico *images* utilizando o protocolo selecionado (TCPROS ou UDPROS)

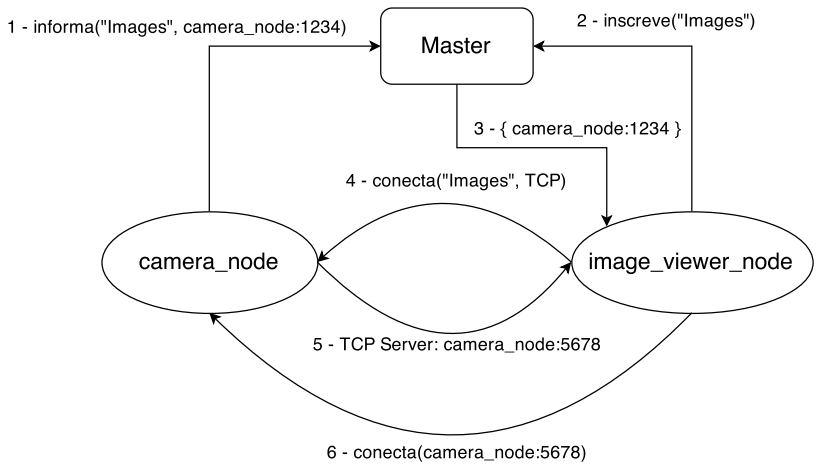

Figura 3.7: Conexão entre nós no ROS.

4 Desenvolvimento da Arquitetura

Como exposto no capítulo 3, os softwares desenvolvidos utilizando o ROS são organizados na forma de pacotes (*packages*). Um pacote pode conter arquivos executáveis (nós), bibliotecas, conjuntos de dados ou quaisquer outros recursos necessários para a sua execução.

Para resolver o problema proposto e fazer um gerenciamento dinâmico dos canais de comunicação, foi desenvolvido um pacote ROS denominado *dynamic_bandwidth_manager* responsável por controlar a taxa em que um nó publica um tópico, gerenciando os diferentes canais de comunicação onde comandos, dados sensoriais e *frames* de vídeo são enviados para outros agentes do sistema multirrobô.

O pacote foi desenvolvido utilizando o ROS para que possa ser testado e aplicado em projetos reais de aplicações multirrobôs. Uma das principais características do ROS é permitir o desenvolvimento colaborativo de software para a área da robótica através de uma plataforma comum e através da possibilidade de juntar diversos pacotes em um único sistema. Dessa forma, o pacote desenvolvido nesse trabalho pode ser facilmente utilizado em projetos de pesquisa futuros.

Nesse capítulo é apresentado uma formulação do problema; a modelagem do pacote com o conjunto de classes, a interação entre elas e os principais algoritmos; e a forma de utilização do pacote.

4.1 Formulação do Problema

Este trabalho propõe o desenvolvimento de uma biblioteca para o gerenciamento de alocação de banda em sistemas multirrobôs. O principal foco está no gerenciamento dos canais de comunicação do sistema ajustando dinamicamente as frequências nos quais os elementos sensores enviam dados para os outros elementos do sistema.

Um canal de comunicação pode ser um sensor, como uma câmera de vídeo ou um

laser scanner, onde a taxa de comunicação é controlada por uma frequência. Em um sistema estático, essa frequência é calculada utilizando-se de algum parâmetro de projeto e não é alterada caso alguma mudança no ambiente ocorra. Essa frequência de comunicação pode não ser a melhor solução para o sistema nesse momento, principalmente em ambientes com restrições de banda. Caso o robô esteja em uma situação onde uma alta taxa de comunicação de um determinado sensor não seja importante para a performance do sistema como um todo, uma alta frequência desse canal de comunicação pode criar restrições na rede que atrapalhem a comunicação de outros robôs que se encontrem em situações mais prioritárias.

Como exemplo, pode-se considerar um cenário com três robôs em uma aplicação de identificação de vítimas em áreas de desastre. Cada robô desloca-se sobre a área lendo informações do ambiente através de sensores e enviando para uma central de monitoramento remota onde operadores humanos auxiliam na tarefa de identificação de vítimas utilizando como base as informações enviadas pelos robôs. A Figura 4.1 apresenta o cenário de exemplo apresentado.

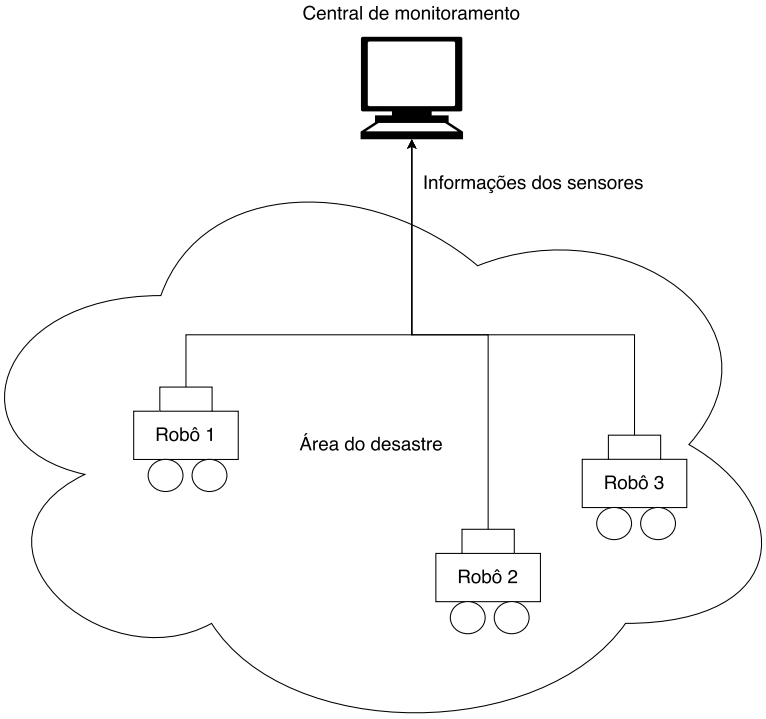

Figura 4.1: Cenário de uma aplicação de identificação de vítimas em áreas de desastre.

Nesse exemplo, com base em um estudo feito pelos projetistas do sistema, a taxa de comunicação ideal para a máxima eficiência da aplicação é de 24 Hz, ou seja, cada robô deve enviar os dados lidos pelos sensores 24 vezes a cada segundo. Essa taxa de

comunicação garante que o sistema de monitoramento possa prever, com a maior certeza possível, a presença e a localização de vítimas na área monitorada pelo robô.

Como o sistema apresentado é executado em uma área com condições limitadas, a rede de comunicação disponível não permite a taxa de comunicação máxima (24 Hz para cada robô). Ao invés disso, a frequência disponível, dividindo igualmente a largura de banda, é de 15 Hz totalizando 45 mensagens sendo transmitidas por segundo. Essa taxa de comunicação de 15 Hz permite que um robô identifique um vítima, mas com um nível menor de precisão da localização (quanto maior a frequência, maior é a precisão e menor é o erro, em metros, da localização da vítima). Ou seja, o robô consegue identificar que existe uma vítima na área, mas não consegue definir a área em que ela se encontra (figura 4.2).

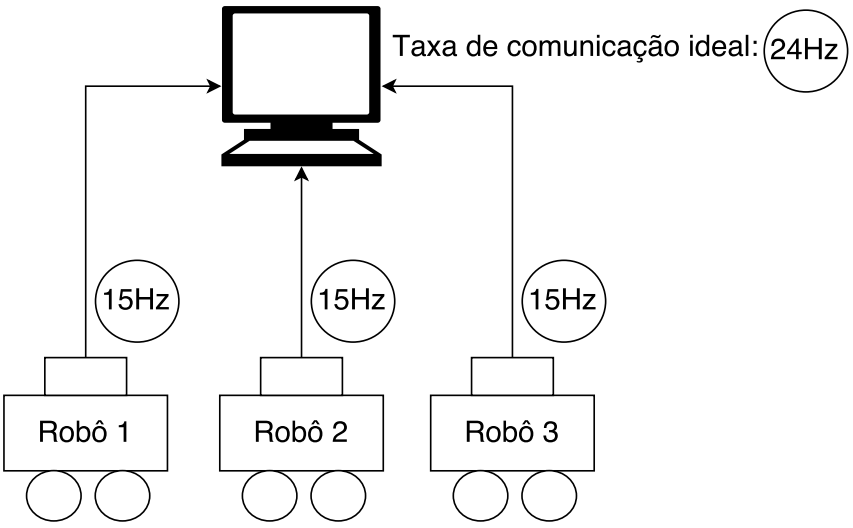

Figura 4.2: Taxas de comunicação estáticas na aplicação de identificação de vítimas em áreas de desastre.

Como pode ser visto no exemplo acima, restrições de banda podem influenciar na eficiência da solução. Atribuir uma frequência fixa de 15 Hz para todos os robôs permite uma distribuição da banda e evita que a comunicação ultrapasse o máximo de banda disponível para a aplicação. Mas essa solução não faz com que um robô encontre uma vítima com o máximo de precisão possível mesmo que os outros robôs estejam longe desse objetivo. Nesse caso, o sistema poderia reduzir a frequência dos robôs que ainda não detectaram nenhuma vítima para o mínimo de frequência aceitável, aumentando o erro da localização da vítima, e disponibilizando uma maior frequência para o robô que encontrou uma vítima nas proximidades e necessita agora procurar a sua localização exata.

4.2 Gerenciamento de Banda Baseado em Eventos do Ambiente

O controle das frequências de envio dos canais de comunicação pode ser feito de forma dinâmica utilizando o estado do ambiente e a banda disponível no momento. Essa abordagem é criada a partir da pressuposição de que a taxa de comunicação de um canal depende das mudanças que ocorrem no ambiente do robô.

Na aplicação descrita acima, o sistema pode disponibilizar uma frequência variável para cada robô. Um robô pode utilizar uma taxa de comunicação mais baixa até encontrar indícios de que existe uma vítima nas proximidades. O sistema diminui a frequência de envio de algum dos outros robôs e aumenta a frequência do robô que mais necessita no momento. Dessa forma será possível ter a posição da vítima de forma mais exata para que a equipe de resgate perca o mínimo de tempo possível.

Nesse caso, a otimização da banda é feita de acordo com a necessidade do momento, levando em consideração os eventos do ambiente importantes para a execução da tarefa. A Figura 4.3 mostra as frequências de cada robô no momento em que o Robô 2 encontra uma vítima. Nesse momento, o Robô 1 e o Robô 3 não possuem nenhuma evidência de ter encontrado vítimas na área que eles estão monitorando e por isso podem ter suas frequências ajustadas para o mínimo desejável.

Na biblioteca proposta nesse trabalho, esse comportamento foi implementado atribuindo uma prioridade baseada nos eventos do ambiente para cada canal de comunicação gerenciado. Ou seja, a prioridade pode ser modelada como uma função dos eventos do ambiente e representa quão importante um canal é para a aplicação. Estes eventos são modelados dependendo da aplicação onde a biblioteca está sendo utilizada.

Utilizando uma aplicação de teleoperação como exemplo, onde um operador controla um conjunto de robôs remotamente baseado nas imagens da câmera do robô, pode-se definir a velocidade e a distância para os obstáculos como os eventos do ambiente que afetam o canal de comunicação. Assim, se a velocidade do robô aumenta e a distância de obstáculos diminui, a prioridade do canal de comunicação que representa a câmera aumenta para que o operador tenha uma melhor qualidade de imagem e possa controlar melhor o robô (se a velocidade do robô aumenta ou se ele se encontra mais próximo de obstáculos a possibilidade de colisão é maior).

Cada canal de comunicação que possui sua frequência gerenciada pela biblioteca implementada é considerado um canal de comunicação gerenciado. Considerando i (onde $i \in \{1, 2, 3, ..., n\}$) como sendo os canais de comunicação gerenciados, cada canal está asso-

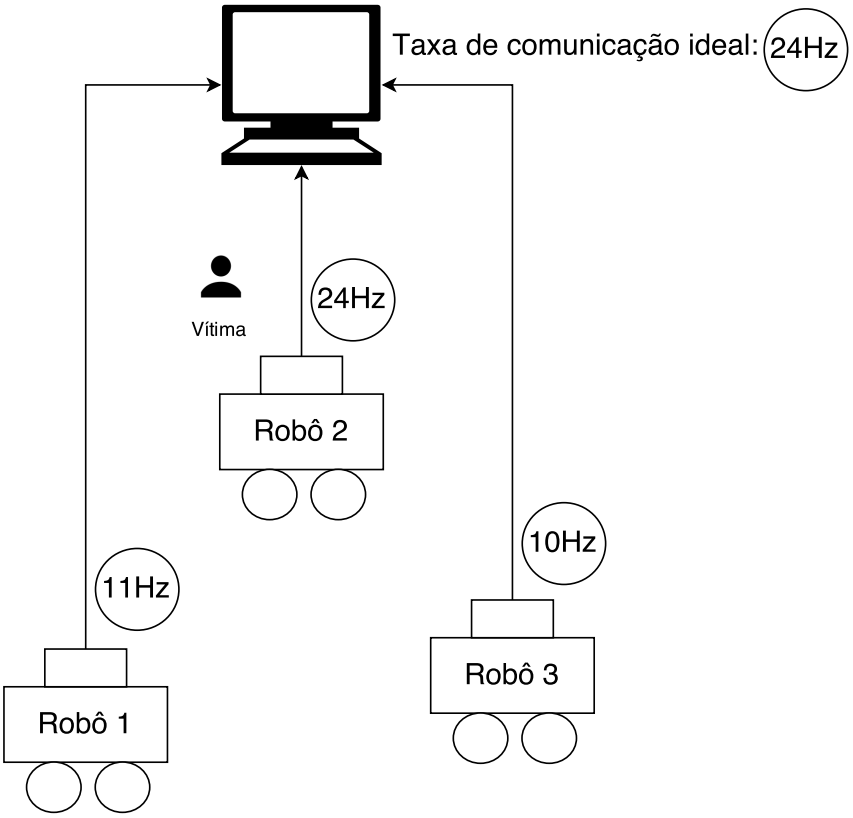

Figura 4.3: Taxas de comunicação dinâmicas na aplicação de identificação de vítimas em áreas de desastre.

ciado à uma frequência f_i representando a taxa de comunicação em Hz (ou seja, quantas mensagens são enviadas por segundo pelo canal i). Assim, pode-se definir um vetor F representando as frequências de cada canal gerenciado pelo sistema:

$$F = \{f_1, f_2, f_3, ..., f_n\} \tag{4.1}$$

A prioridade p_i do canal de comunicação c_i é calculada em função dos eventos do ambiente e_i que afetam esse canal de comunicação. Assim, p_i em um dado tempo pode ser definido por:

$$p_i = f(e_1, e_2, ..., e_n) \tag{4.2}$$

O resultado da função é normalizado para valores entre $[0 : 1]$ levando em consideração o tamanho da mensagem e as prioridades dos outros canais de comunicação de acordo com a equação 4.3. O valor 1 é a maior prioridade e representa os casos onde o canal de comunicação precisa utilizar a frequência mais alta dentro dos limites estabelecidos pela

aplicação e pela largura de banda disponível.

$$p_i' = \frac{p_i.w_i}{\sum_{k=1}^{c} p_k.w_k}, \tag{4.3}$$

onde c representa os canais de comunicação e w_i é o tamanho da mensagem do canal de comunicação c_i.

Essa normalização garante que o tamanho da mensagem seja levado em consideração no cálculo das frequências. Sem esse ajuste, o otimizador não atribui as frequências de forma proporcional à prioridade, gerando resultados incoerentes com o objetivo da biblioteca.

4.3 Frequência Dinâmica em um Tópico ROS

Em ROS, os canais de comunicação são representados pelos tópicos. São através deles que os dados dos sensores são enviados para outros elementos do sistema. O código abaixo mostra a criação de um nó *rescue_node* que publica um tópico chamado *rescue_info*:

Código-Fonte 4.1: Exemplo de uso da classe ros::Rate

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from rescue_app.msg import RescueInfo

def getRescueInfo():
    # Retorna a mensagem a ser publicada pelo topic

def run():
    pub = rospy.Publisher('rescue_info', RescueInfo, queue_size=10)
    rospy.init_node('rescue_node', anonymous=True)
    rate = rospy.Rate(15) # Frequencia de 15hz
    while not rospy.is_shutdown():
        message = getRescueInfo()
        pub.publish(message)
        rate.sleep()

if __name__ == '__main__':
    try:
        run()
    except rospy.ROSInterruptException:
```

pass

Como pode ser visto, no ROS a frequência de envio das informações no tópico é controlada por uma classe utilitária *ros::Rate* que mantém uma taxa particular para um *loop*. Essa classe pode ser utilizada para controlar a frequência de transmissão do tópico. Entretanto, a classe *ros::Rate* faz um controle estático da frequência que deve ser escolhida em tempo de desenvolvimento. Isso faz com que a arquitetura da aplicação não permita um ajuste das frequências de envio dos tópicos dependendo da situação.

Para a criação de um sistema dinâmico de gerenciamento das frequências de comunicação em ROS, é necessário a implementação de outras formas de controlar as frequências de transmissão dos tópicos. O *dynamic_bandwidth_manager* disponibiliza uma classe *DBMRate* para esse controle dinâmico ajustando as frequências de acordo com um parâmetro armazenado no *Parameter Server*. A classe *DBMRate* foi construída herdando todas as características da classe padrão disponibilizada pelo ROS (*ros::Rate*). Dessa forma, qualquer correção ou melhoria implementada na classe padrão é automaticamente incorporada.

O nome do parâmetro que contém o valor da frequência é informado durante a construção do objeto e o parâmetro é criado no *Parameter Server*. A cada chamada do método *sleep()*, o valor da frequência é atualizado de acordo com o parâmetro e o tempo de espera do *loop* é ajustado. A figura 4.4 mostra um esquema do funcionamento da classe *DBMRate*.

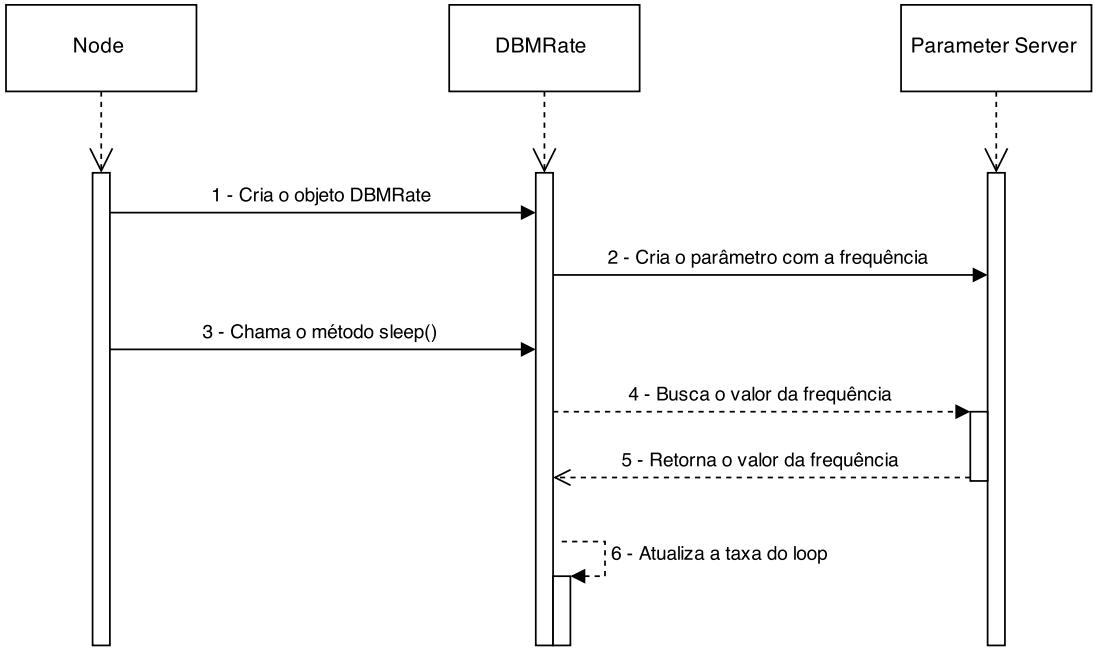

Figura 4.4: Funcionamento da classe *DBMRate*.

Como pode ser visto, um dos problemas dessa abordagem é a quantidade de chamadas ao *Parameter Server*. Para resolver esse problema, os valores das frequências são buscados através de *Cached Parameters*, provendo um cache local dos dados do parâmetro. Essa forma de busca diz ao *Parameter Server* que o nó deve ser informado toda vez que o valor do parâmetro é alterado. Ou seja, o valor do parâmetro é armazenado localmente pelo nó e, toda vez que o parâmetro é alterado, o nó é informado dessa alteração. Esse método de consulta dos parâmetros faz com que a busca seja mais rápida, mas pode sobrecarregar o *Master*, uma vez que todos o nós que utilizam esse parâmetro devem ser notificados sempre que uma alteração no parâmetro é feita.

Para a criação de tópicos em ROS é utilizado a classe *ros::Publisher*. Essa classe registra o tópico no *Master* e fornece o método *publish*, responsável por publicar a mensagem neste tópico. Mas como pode ser visto no código mostrado anteriormente, o controle da taxa de comunicação do tópico é feito manualmente dentro de um *loop*.

Para facilitar o processo de criar tópicos que publicam mensagens em uma taxa dinâmica de comunicação, o *dynamic_bandwidth_manager* disponibiliza uma classe chamada *DBMPublisher*. Essa classe utiliza o *DBMRate* internamente e recebe as frequências mínima e máxima, e um método que retorna a mensagem a ser enviada no seu construtor e no seu método *start*. Dessa forma, a publicação das mensagens é feita automaticamente de acordo com a frequência armazenada no *Parameter Server* sem que haja a necessidade

de um controle manual da publicação através de *loops*.

O código abaixo apresenta a criação do mesmo tópico mostrado anteriormente, mas utilizando o *DBMPublisher*. A principal diferença é que não existe mais a necessidade de um *loop* para enviar as mensagens do tópico. Durante a chamada do método *start*, é passado como parâmetro um método que retorna a mensagem a ser publicada pelo tópico (método *getRescueInfo*, no exemplo) e, internamente, a classe *DBMPublisher* trata de publicar a mensagem na frequência configurada.

Código-Fonte 4.2: Exemplo de uso da classe DBMRate

```
#!/usr/bin/env python
# license removed for brevity
import rospy
import dynamic_bandwidth_manager
from rescue_app.msg import RescueInfo

def getRescueInfo():
    # Retorna a mensagem a ser publicada pelo topic

def run():
    # Frequencia minima de 10hz e maxima de 24hz
    pub = dynamic_bandwidth_manager.DBMPublisher(
        'rescue_info', RescueInfo, 10, 24)
    rospy.init_node('rescue_node', anonymous=True)

    # Inicia a publicacao das mensagens com a
    # frequencia armazenada no Parameter Server
    pub.start(getRescueInfo)

if __name__ == '__main__':
    try:
        run()
    except rospy.ROSInterruptException:
        pass
```

4.4 Otimização das Frequências dos Canais de Comunicação

A classe *DBMPublisher* possibilita que um tópico publique mensagens em uma frequência dinâmica mas não resolve o problema de calcular o melhor valor para as frequências levando em consideração os eventos do ambiente. De forma a calcular as frequências

de envio dos canais de comunicação gerenciados pelo *dynamic_bandwidth_manager* foi criado uma modelagem de um problema de programação linear.

A programação linear é o mecanismo mais natural para a formulação de uma grande variedade de problemas. Um problema de programação linear é um problema de otimização e é caracterizado, como o nome implica, por funções lineares das variáveis. A função objetivo é linear e as restrições são equações ou inequações lineares. A finalidade da programação linear é maximizar ou minimizar a função objetivo dado um conjunto de restrições (LUENBERGER, 1973).

O tamanho da mensagem w_i do canal i representa o tamanho das mensagens, em bytes por exemplo, que um canal i envia para os outros elementos do sistema. Este parâmetro é importante para calcular o total de banda consumida por todos os canais de comunicação que é limitado pela banda total disponível para o sistema (B_{max}):

$$\sum_{i=1}^{n} w_i.f_i \leq B_{max}, \tag{4.4}$$

onde n é o número de canais de comunicação.

Todas as frequências f_i são limitadas com um valor mínimo e máximo: $f_{i_{min}}$ e $f_{i_{max}}$, respectivamente. Os canais de comunicação devem ser otimizados de forma a maximizar a utilização da banda disponível para a aplicação.

A prioridade p_i define quais canais são mais importantes para a aplicação em um dado instante. Os canais mais prioritários devem ter mais recursos de banda reservados. Esse comportamento é atingido ajustando os limites $f_{i_{min}}$ e $f_{i_{max}}$ de acordo com o valor da prioridade p_i. Se um canal tem uma prioridade $p_i = 1$, os limites da frequência f_i, em um dado tempo, devem ser calculados próximo do máximo ($f_{i_{max}}$). Em outras palavras, o novo valor da frequência mínima $f'_{i_{min}}$ é uma função de p_i. Assim, $f'_{i_{min}}$ pode ser definida por:

$$f'_{i_{min}} = (f_{i_{max}} - f_{i_{min}})p_i + f_{i_{min}} \tag{4.5}$$

A equação (4.5) define um valor mínimo para a frequência f_i em um dado instante baseado na prioridade p_i. Se a prioridade $p_i = 0$, a frequência do canal de comunicação é limitada com os valores mínimos e máximos definidos para o canal ($f_{i_{min}}$ e $f_{i_{max}}$). Enquanto a prioridade aumenta, o valor mínimo para a frequência se aproxima do valor máximo fazendo com que o sistema disponibilize uma maior banda para o canal.

As frequências de cada canal de comunicação são descritas como um problema de programação linear. Dessa forma, a formulação do problema é dada por:

$$\begin{aligned} \text{maximizar} \quad & \sum_{i=1}^{n} w_i.f_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n} w_i.f_i \leq B_{max} \\ & f_i \geq f'_{i_{min}} \\ & f_i \leq f_{i_{max}}, \end{aligned} \tag{4.6}$$

onde n é o número de canais de comunicação.

Entretanto, em alguns casos, não existe nenhuma solução para o problema e o sistema informa essa situação ao projetista. Nesses casos, os limites das restrições devem ser ajustados aumentando a banda máxima disponível ou diminuindo as frequências mínima e máxima. Como as frequências mínima e máxima impactam diretamente na qualidade das tarefas que estão sendo executadas, o mais recomendado é aumentar a banda máxima disponível para a aplicação.

De forma a fazer um gerenciamento dinâmico da banda, o sistema deve levar em consideração que o tamanho da mensagem pode mudar e não tratá-lo como um valor estático. Toda vez que uma mensagem é enviada por um canal gerenciado, o *DBMPublisher* verifica se o tamanho da mensagem foi alterado e modifica o seu valor em um parâmetro armazenado no *Parameter Server*. Assim, o tamanho das mensagens enviadas através dos canais de comunicação podem ser dinamicamente alterados.

4.5 Implementação de Outras Estratégias de Otimização

Para uma independência do algoritmo de otimização utilizado na biblioteca, um módulo que trata da otimização da banda foi criado. O *DBMOptimizer* é uma biblioteca ROS que ajuda na criação de estratégias mais complexas para o problema de otimização das frequências. Esse módulo executa o algoritmo de otimização a cada instante como definido pelo parâmetro */dbm/optimization_rate_in_seconds* e armazena o resultado das frequências calculadas no parâmetro $[topic\_name]/dbm/frequency/current\_value$. Esse último parâmetro é utilizado pelo *DBMPublisher* para recuperar o valor da frequência de transmissão do tópico. Dessa forma, os algoritmos de otimização utilizados pelo DBML podem ser substituídos sem que a biblioteca seja alterada. Um pesquisador pode

implementar novas estratégias de otimização de maneira independente e utilizá-las para calcular as frequências de envio dos canais de comunicação gerenciados pelo DBML.

Neste trabalho foi criado o nó *default_optimizer_node*, utilizando o *DBMOptimizer*, que faz a otimização das frequências de acordo com a modelagem do problema de programação linear. O código abaixo mostra a criação de uma estratégia de otimização utilizando o *DBMOptimizer*:

Código-Fonte 4.3: Exemplo de criação de um nó utilizando o DBMOptimizer

```
#!/usr/bin/env python

import rospy
import dynamic_bandwidth_manager
import pulp
import numpy as np

def optimize(managed_topics):
    # Faz a otimizacao e retorna um dicionario do tipo
    # [nome_do_topic: valor_da_frequencia] (o parametro
    # managed_topics possui uma lista com os topics que
    # devem ser levados em consideracao na otimizacao)

if __name__ == '__main__':
    try:
        rospy.init_node('default_optimizer', anonymous=True)
        optimizer = dynamic_bandwidth_manager.DBMOptimizer(optimize)
        optimizer.start()

    except rospy.ROSInterruptException: pass
```

No exemplo acima, um novo algoritmo de otimização é criado utilizando o *DBMOptimizer*. O método *optimize(managed_topics)* implementa a estratégia de otimização das frequências dos canais de comunicação. Esse método recebe como parâmetro uma lista com os nomes dos tópicos gerenciados pelo DBML e retorna um dicionário do tipo [nome_do_tópico: valor_da_frequência] com os valores das frequências calculadas. Esse método é executado automaticamente pelo DBML e as frequências são atualizadas no *Parameter Server*.

4.6 Gerenciamento de Canais de Comunicação Locais

Como apresentado no capítulo sobre ROS, os nós são projetados para operar em uma ”escala fina” onde um sistema que controla um robô terá, na maioria das vezes, diversos nós, cada um cuidando de um ponto específico da aplicação. Estes nós se comunicam entre si utilizando os tópicos. Se um desses tópicos tem a sua frequência controlada pela biblioteca, esse tópico é tratado como um canal de comunicação gerenciado. Entretanto, a questão é: como gerenciar canais que enviam mensagens apenas para outros nós que estão executando na mesma máquina? Nesses casos, o tópico não tem nenhum impacto na utilização da banda e deveria ser ignorado pelo algoritmo de otimização.

De forma a tratar esse problema, o *DBMOptimizer* decide quais tópicos devem ser gerenciados pelo sistema em um dado tempo verificando se não existem nós externos comunicando com o tópico. Se não existe nenhum nó externo inscrito no tópico, ele não é tratado como um canal de comunicação gerenciado, é desconsiderado da otimização e tem a sua frequência ajustada para o valor máximo definido.

Outra questão importante é quando existem nós executando em máquinas diferentes inscritos em um mesmo tópico e pelo menos um desses nós está executando na mesma máquina de onde o tópico está sendo publicado. Por exemplo, o nó 1 e o nó 2 estão executando nas máquinas A e B respectivamente e estão inscritos no tópico */camera*. O tópico */camera* está sendo publicado pelo nó 3 que também está sendo executado na máquina B. A Figura 4.5 ilustra esse exemplo.

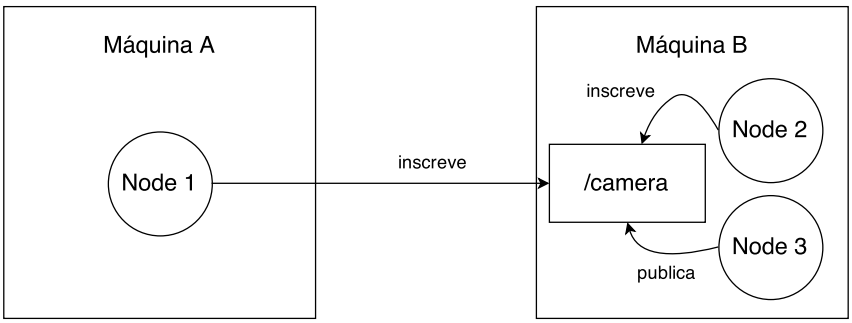

Figura 4.5: Problema de nós inscritos em um mesmo topic.

Se o sistema limita a taxa de publicação do tópico, um processamento local, não sujeito a restrições banda, não teria a sua disposição a taxa máxima de dados do tópico. Para resolver esse problema, o sistema cria dois tópicos para cada canal de comunicação gerenciado: um tópico com a taxa máxima de publicação (*/topic_name/*) e um tópico com a taxa de comunicação gerenciada pelo sistema levando em consideração a otimização das

frequências (*[topic_name]/managed*).

Dessa forma, um node se inscreve no tópico que possui a taxa máxima de comunicação (*[topic_name]*) caso esteja executando na mesma máquina onde o tópico está sendo publicado. Entretanto, se o node está executando em uma máquina remota, a inscrição é feita no tópico com a taxa de comunicação gerenciada (*[topic_name]/managed*). Assim, tópicos com uma taxa de comunicação máxima continuam disponíveis para processamentos locais e operações de *log* (registro de eventos relevantes).

A decisão de qual tópico se inscrever é implementada pelo *DBMSubscriber*. Essa classe herda todas as carecterísticas da classe padrão do ROS *Subscriber* registrando em um tópico específico. A Figura 4.6 apresenta um esquema ilustrando o comportamento descrito acima.

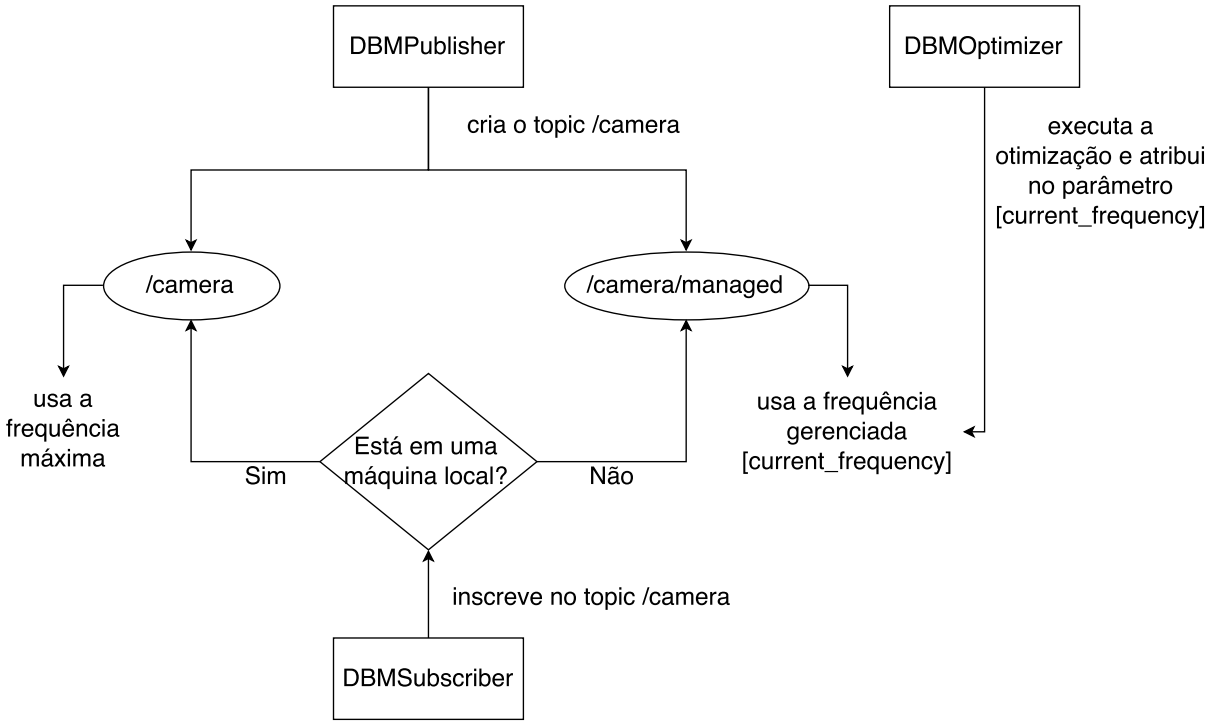

Figura 4.6: Funcionamento de tópicos locais e remotos.

4.7 Parâmetros do Sistema

Os parâmetros do sistema são um conjunto de parâmetros utilizados de forma a melhorar a customização da biblioteca permitindo a adaptação para novas aplicações sem a necessidade de mudar o código-fonte da biblioteca. Os parâmetros são armazenados

no *Parameter Server* e são compartilhados entre os nós. Os parâmetros do sistema são descritos abaixo:

- */dbm/topics*: Lista que contém o nome de todos os tópicos gerenciados pelo sistema. É atualizada pelo *DBMOptimizer* no início da execução do algoritmo de otimização levando em consideração se o tópico possui nós inscritos executando em máquinas remotas;
- [*topic_name*]/*dbm*/$frequency$/*current_value*: Valor da frequência atual do tópico [*topic_name*];
- [*topic_name*]/*dbm*/$frequency$/*min*: Frequência mínima para o tópico [*topic_name*];
- [*topic_name*]/*dbm*/$frequency$/*max*: Frequência máxima para o tópico [*topic_name*];
- [*topic_name*]/*dbm*/*priority*: Prioridade do tópico baseado nos eventos do ambiente [*topic_name*];
- [*topic_name*]/*dbm*/*message_size_in_bytes*: Tamanho da mensagem enviada pelo tópico [*topic_name*];
- */dbm/max_bandwidth_in_mbit*: Total de banda do sistema;
- */dbm/max_bandwidth_utilization*: Porcentagem da banda disponível para a aplicação (valores entre $[0 : 100]$);
- */dbm/optimization_rate_in_seconds*: Frequência em que o algoritmo de otimização é executado.

Em uma aplicação, podem existir mensagens sendo transmitidas na rede que não são gerenciadas pelo DBM. Serviços e outros tópicos não gerenciados podem ser utilizados, assim como outros tipos de comunicação entre os elementos do sistema. Exemplos de comunicações não gerenciadas podem ser alocação de tarefas para os robôs, comandos diversos ou qualquer outro tipo de funcionalidade que dependa da utilização da rede. Nesses casos, a banda do sistema definida pelo parâmetro */dbm/max_bandwidth_in_mbit* não deve ser utilizada totalmente pelos canais de comunicação gerenciados pelo DBM. Uma parcela dessa banda deve ser reservada para as comunicações não gerenciadas pela biblioteca. Isso pode ser feito através do parâmetro */dbm/max_bandwidth_utilization* que garante que apenas uma parte da banda total do sistema seja utilizada no cálculo das frequências.

4.8 Fluxograma do Funcionamento Básico do Pacote

Como apresentado nas seções anteriores, a arquitetura do pacote foi dividida em três bibliotecas e um node: *DBMPublisher*, *DBMSubscriber*, *DBMOptimizer* e o *default_optimizer_node*, respectivamente. O *DBMPublisher* é uma classe que estende a função da classe básica do ROS *ros::Publisher* recebendo os valores para a frequência mínima e a frequência máxima e criando um tópico gerenciado pelo sistema. O *DBMSubscriber* é uma biblioteca utilizada para se inscrever em um tópico gerenciado criado pelo *DBMPublisher*. O *DBMOptimizer* facilita a criação de estratégias de otimização e o *default_optimizer_node* resolve um problema de otimização linear calculando as frequências de envio dos tópicos baseado nas informações do ambiente.

Um nó que deseja publicar informações utilizando uma frequência de comunicação gerenciada pelo sistema cria um tópico utilizando o *DBMPublisher* e informa os limites de frequência que devem ser respeitados (frequência mínima e frequência máxima). Junto com o tópico também são criados todos os parâmetros do sistema (descritos na seção anterior) no *Parameter Server*. A figura 4.7 apresenta o comportamento da biblioteca durante a criação de um tópico utilizando o *DBMPublisher*.

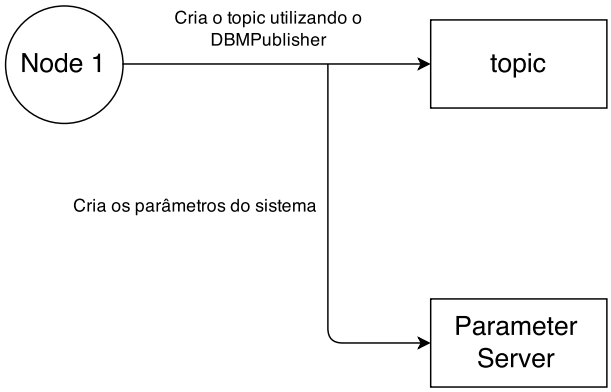

Figura 4.7: Criação de um tópico através do *DBMPublisher*.

Um outro nó que deseja receber as informações se inscreve no tópico criado utilizando o *DBMSubscriber*. O *DBMSubscriber* verifica se o nó está rodando na mesma máquina onde o tópico está sendo publicado e decide se deve se inscrever no tópico gerenciado ou no tópico com a taxa máxima de comunicação (figura 4.8).

O *default_optimizer_node* ou qualquer outro nó que esteja implementando uma estratégia de otimização utilizando o *DBMOptimizer* executa o algoritmo de otimização de acordo com o parâmetro */dbm/optimization_rate_in_seconds* e atualiza as frequências

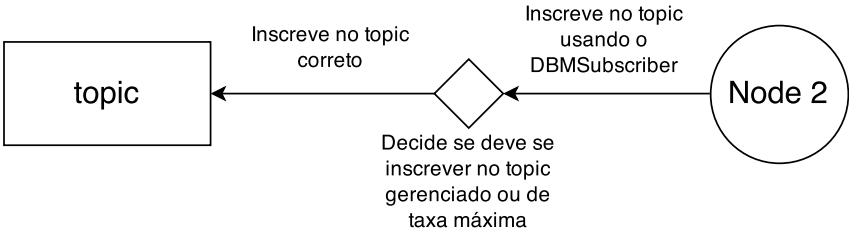

Figura 4.8: Inscrição em um tópico utilizando o *DBMSubscriber*.

dos tópicos no *Parameter Server*. O nó que está publicando o tópico é notificado e atualiza a frequência de publicação das mensagens (figura 4.9).

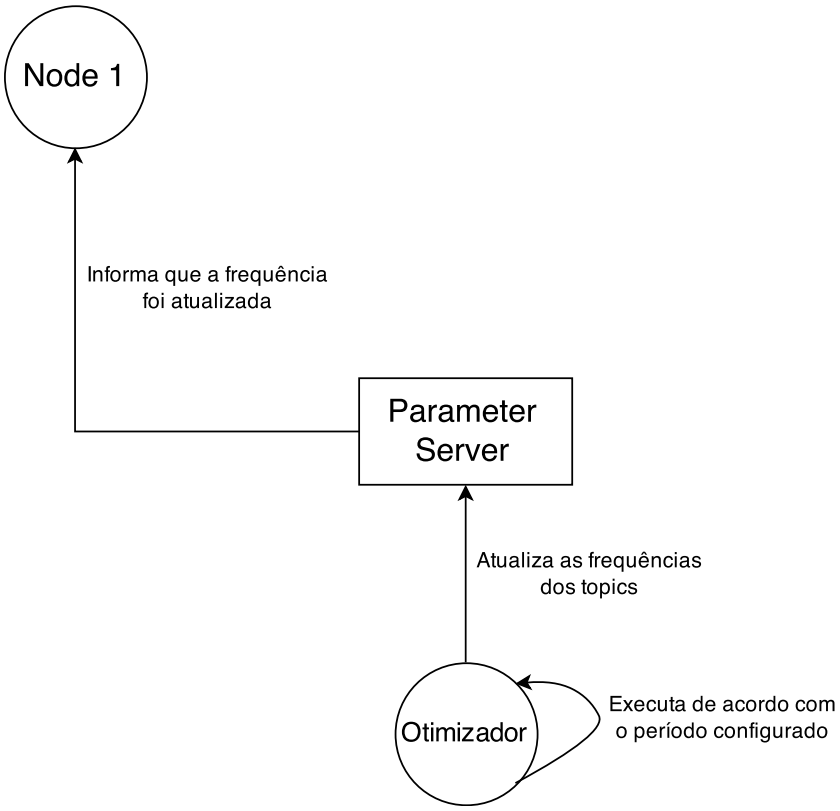

Figura 4.9: Atualização das frequências pelo otimizador.

A figura 4.10 mostra o resumo do funcionamento do pacote como descrito acima.

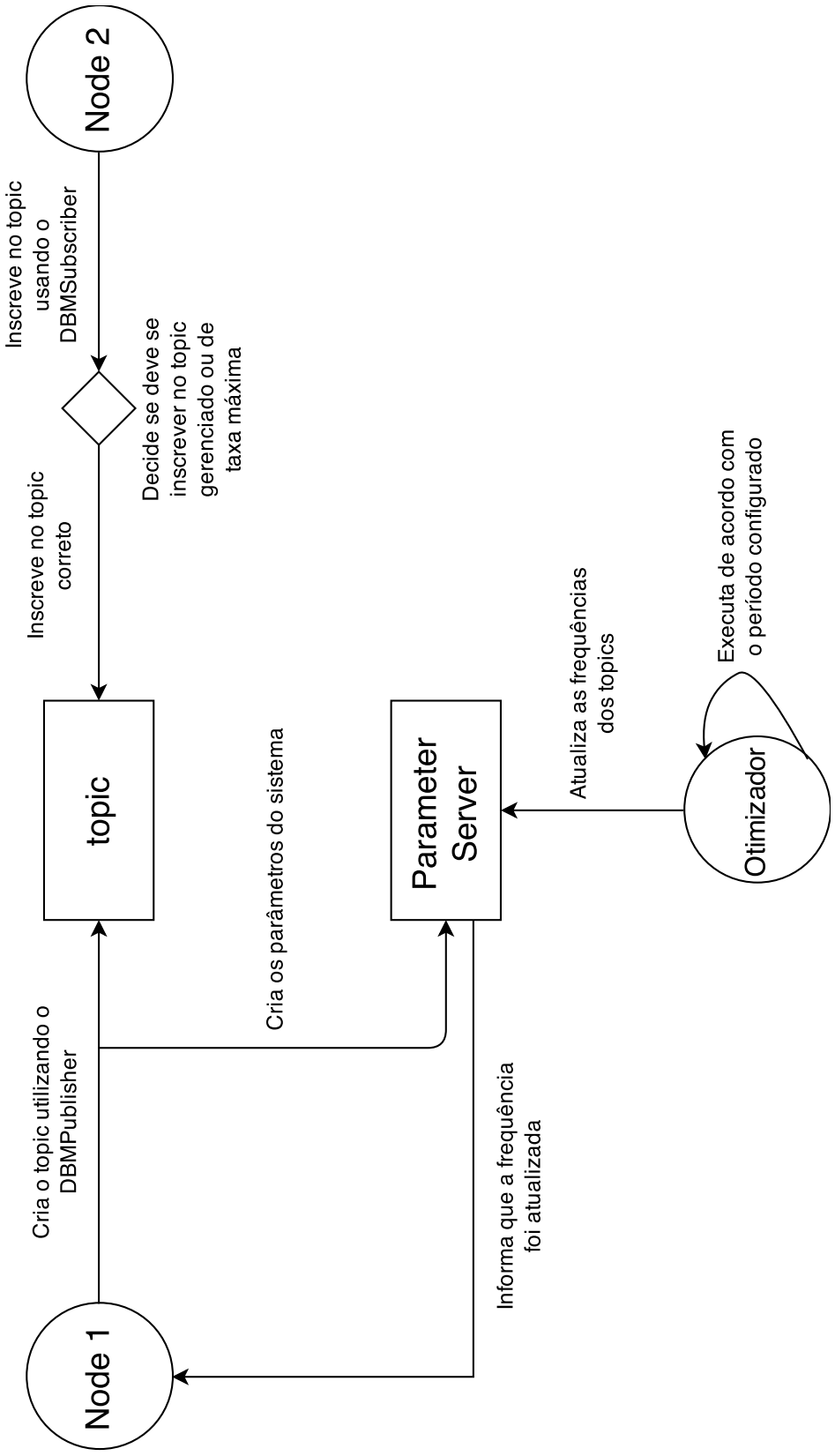

Figura 4.10: Resumo do funcionamento básico do pacote.

A figura 4.11 apresenta o diagrama com as classes da biblioteca e as suas dependências. As classes DBMPublisher, DBMSubscriber e DBMRate herdam das classes Publisher, Subscriber e Rate do ROS, respectivamente. A classe DBMPublisher utiliza a classe a DBMRate para controlar a taxa de publicação dinamicamente. A classe DBMParam é uma classe utilitária que contém os métodos de acesso e atribuição dos parâmetros do DBML como, por exemplo, a frequência mínima e máxima de um tópico. As classes DBMPublisher, DBMRate e DBMOptimizer utilizam a classe DBMParam para auxiliar na recuperação e configuração dos parâmetros da biblioteca. A classe DBMOptimizer auxilia os desenvolvedores a criarem novas estratégias de otimização. O nó *default_optimizer* utiliza a classe DBMOptimizer e implementa uma estratégia de otimização como descrito anteriormente.

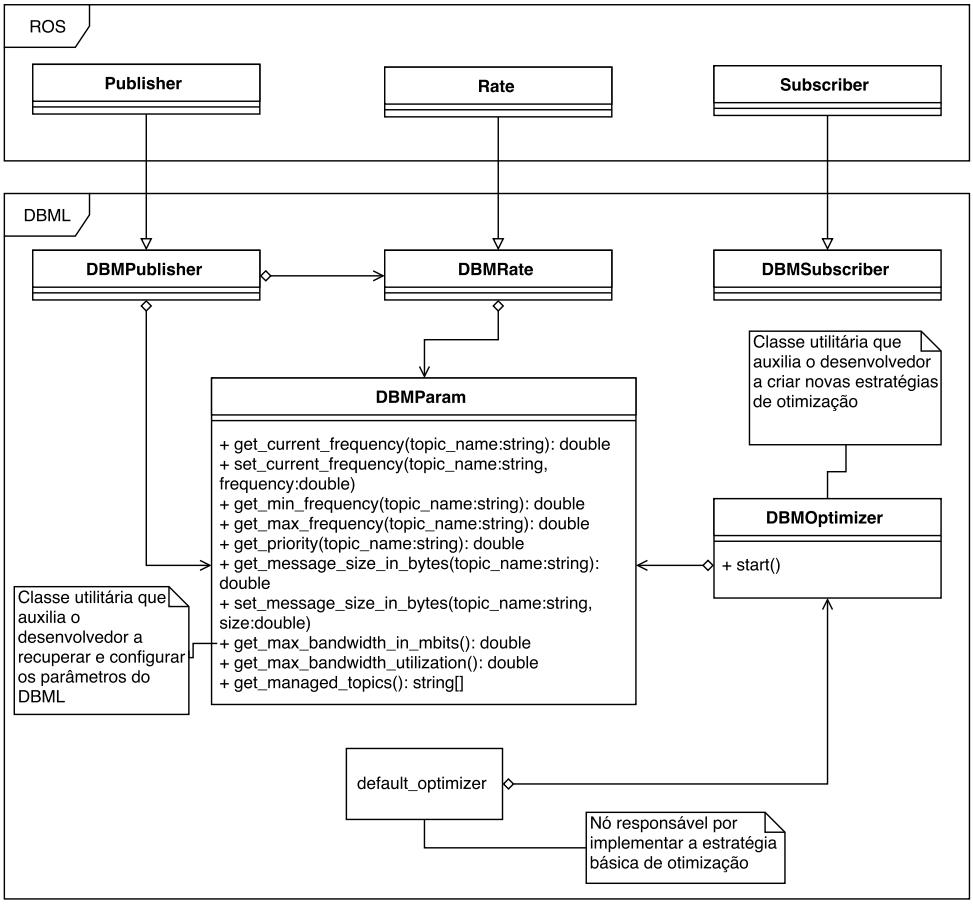

Figura 4.11: Diagrama com as classes e as suas dependências.

De forma a colaborar com divulgação do DBML e facilitar a sua utilização por outros pesquisadores, a DBML-Wiki (2015) apresenta um tutorial da utilização do DBML e um repositório com o seu código-fonte.

5 Simulações e Resultados

Esse capítulo apresenta uma aplicação de teleoperação com o gerenciamento dinâmico de banda utilizando a biblioteca proposta neste trabalho. O exemplo foi desenvolvido em um ambiente simulado executando em máquinas virtuais. A teleoperação foi escolhida como um exemplo de caso de uso por possuir elementos bem definidos com uma clara apresentação de como a comunicação pode depender do ambiente.

5.1 Ambiente de Simulação

Aplicações de teleoperação têm sido úteis em uma variedade de problemas utilizando robôs (SHERIDAN, 1992). Sirouspour e Setoodeh (2005) descrevem um sistema de teleoperação convencional com cinco elementos distintos: operador humano, dispositivo mestre, controlador e canal de comunicação, robô e o ambiente. O operador humano utiliza o dispositivo mestre para manipular o ambiente através do robô. Os controladores coordenam a operação utilizando os canais de comunicação (Figura 5.1).

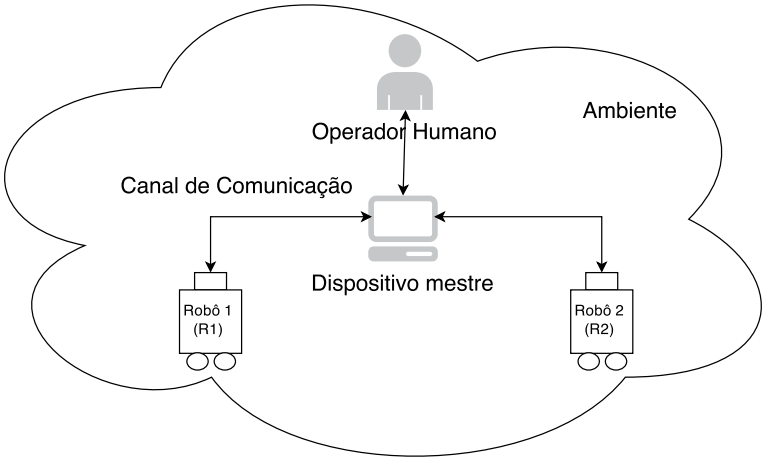

Figura 5.1: Aplicação de teleoperação

Na aplicação proposta como exemplo, os robôs são controlados por um operador humano utilizando uma estação de trabalho remota representando o dispositivo mestre. Os canais de comunicação são utilizados para enviar posição e outros comandos do dispositivo mestre para os robôs e informação visual (imagens da câmera) dos robôs para o dispositivo mestre.

As imagens da câmera dos robôs são enviadas para o dispositivo mestre onde o operador humano está controlando os robôs. O operador controla os robôs manipulando sua velocidade e direção. De forma a executar essa operação, o operador precisa receber *feedback* visual, ou seja, informações suficientes para distinguir os obstáculos no ambiente.

Chen, Haas e Barnes (2007) demonstraram que pessoas têm dificuldades em manter orientação espacial em um ambiente remoto com uma reduzida largura de banda. Se a taxa de transmissão das imagens reduz, o operador pode não conseguir evitar os obstáculos. Se a taxa aumenta, a quantidade de informações sendo transmitidas pelos canais de comunicação pode ultrapassar os limites de banda disponíveis. Dessa forma, os comandos enviados para o robô podem ser perdidos (ou chegarem atrasados) com a perda de pacotes na rede. Esses problemas justificam a utilização de um gerenciamento de banda nesse tipo de aplicação.

Para simular os robôs e o ambiente foram desenvolvidos quatro novos nós utilizando o DBML:

environment_simulator_node : Simula os sensores de distância e velocidade dos robôs. Através desse nó é possível publicar um valor de distância e velocidade ou ler um conjunto de valores de um arquivo e publicá-los através dos tópicos *distance* e *speed* utilizando uma frequência pré-configurada.

robot_camera_node : Captura a imagem da câmera e publica através do tópico *camera* criado utilizando o *DBMPublisher*.

robot_camera_priority_node : Lê os valores de velocidade e distância publicados pelo *environment_simulator_node* e calcula a prioridade de acordo com a equação 4.2.

monitor_node : Se inscreve no tópico *camera* e exibe a imagem publicada pelos robôs, simulando a estação controlada pelo operador humano.

De forma a ilustrar o posicionamento dos robôs no ambiente, foi criado um cenário

simulado contendo obstáculos onde os robôs devem ser guiados através de um caminho até atingirem um ponto de chegada. A figura 5.2 mostra o ambiente utilizado na simulação:

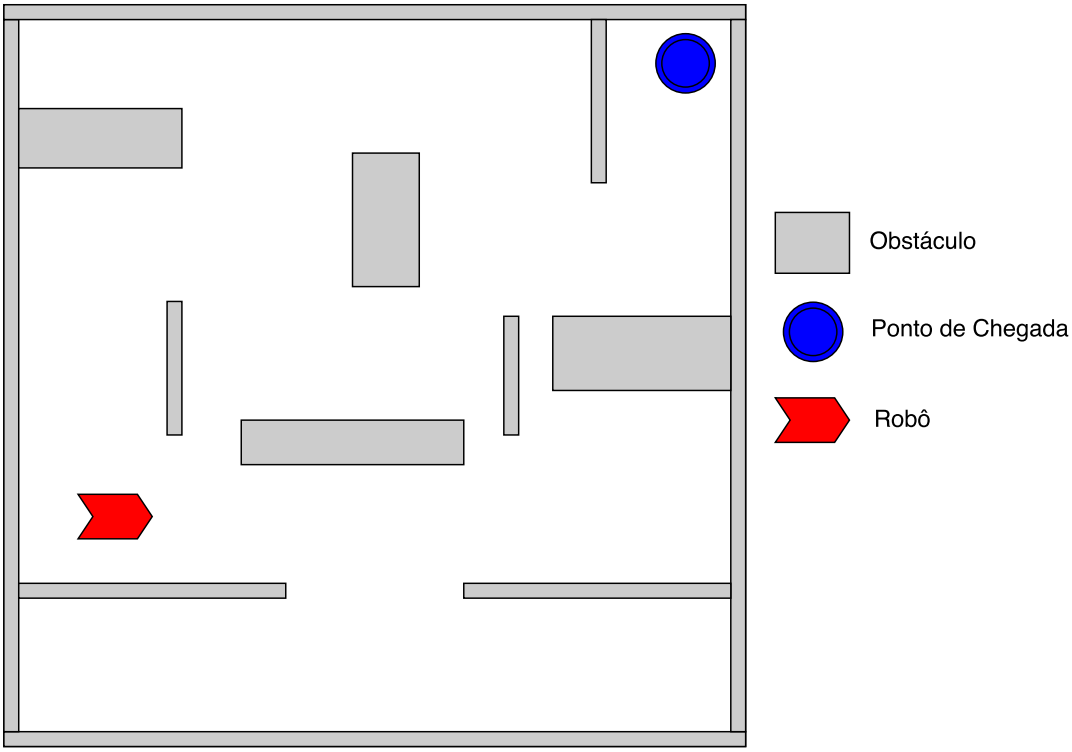

Figura 5.2: Ambiente de simulação

5.2 Canais de Comunicação

Os canais de comunicação entre o operador humano e os robôs são essenciais para uma percepção efetiva do ambiente remoto (FRENCH; GHIRARDELLI; SWOBODA, 2003). O teleoperador depende diretamente da qualidade do vídeo sendo transmitido: a percepção remota e a performance do operador em estimar a distância e o tamanho dos outros elementos do sistema podem ser comprometidas em ambientes com limitada largura de banda (ERP; PADMOS, 2003). Chen, Haas e Barnes (2007) estudaram formas comuns de degradação do vídeo causado por uma baixa largura de banda, incluindo a taxa de *frames por segundo* (fps) sendo transmitidos.

Chen, Haas e Barnes (2007) mostraram que o fps mínimo de 10 Hz deve ser utilizado de forma a evitar uma degradação do vídeo transmitido e não prejudicar a tarefa de teleoperação. Altos valores de fps como 16 Hz são sugeridos para alguns tipos de aplicações como a navegação. Por isso, nesse trabalho, será criado um canal de comunicação para

as imagens da câmera do robô (topic */camera*) com 1 Hz de frequência mínima (representando os casos onde não existem nenhuma necessidade do operador manipular o robô devido a uma estabilidade no ambiente) e 16 Hz para a frequência máxima. A aplicação utiliza uma largura de banda disponível de 11 Mbps. Isto representa uma taxa de transferência de dados de 1375 KBps. O algoritmo de otimização será executado uma vez a cada segundo e 100 % da banda disponível poderá ser utilizada pelos canais de comunicação gerenciados pelo DBML. A tabela 5.1 mostra um resumo das configurações utilizadas na aplicação de simulação:

Tabela 5.1: Configurações da aplicação de simulação.

/camera/dbm/frequency/min	1Hz
/camera/dbm/frequency/max	16Hz
/dbm/max_bandwidth_in_mbit	11 Mbps
/dbm/max_bandwidth_utilization	100%
/dbm/optimization_frequency_in_seconds	1 sec

5.3 Eventos do Ambiente

O capítulo 4 descreve a prioridade do canal de comunicação baseado nos eventos do ambiente. Essa prioridade é calculada para cada canal de comunicação (tópico) e leva em consideração os eventos no ambiente que afetam a importância do canal para o sistema.

Na aplicação de teleoperação, a distância para os obstáculos e a velocidade são eventos do ambiente que serão monitorados de modo a calcular a prioridade do tópico que publica as imagens da câmera (tópico */camera*). O operador precisa de mais *feedback* visual ao operar remotamente um robô com uma velocidade mais alta ou mais perto dos obstáculos.

Mansour et al. (2012) definem, na aplicação de testes utilizada em seu trabalho, uma máxima distância detectada pelos sensores dos robôs de 200 cm e uma velocidade máxima de 50 cm/s. Nesse trabalho, serão definidos os mesmos parâmetros de forma a utilizar valores reais no ambiente simulado. Dessa forma, a prioridade baseada na distância dos obstáculos e na velocidade dos robôs pode ser definida pelas funções:

$$t_c = \frac{distância}{velocidade} \tag{5.1}$$

$$p_{camera} = \begin{cases} 1, & \text{se } T < 3 \\ 0, & \text{se } T > 20 \\ \dfrac{20 - t_c}{17}, & \text{caso contrário,} \end{cases} \tag{5.2}$$

onde $0 <= distancia <= 200$ (cm) e $0 <= velocidade <= 50$ (cm/s).

A equação (5.1) define o tempo restante até o robô colidir com o obstáculo. Assim, quanto mais próximo o robô está de atingir o obstáculo e quanto maior a sua velocidade, menor será o tempo restante até a colisão. A equação (5.2) define a prioridade como uma função do tempo esperado até a colisão do robô com um obstáculo. Caso um robô esteja a mais de 20 segundos de uma colisão, a prioridade do seu canal de comunicação é ajustada para o mínimo possível, representando uma situação onde o operador possui um tempo coniderável para desviar do obstáculo. Caso o tempo para colisão esteja abaixo de 3 segundos, o canal de comunicação deve enviar dados com a máxima prioridade possível para que o operador tenha o *feedback* visual necessário para desviar do obstáculo.

O gráfico 5.3 mostra visualmente a prioridade baseada no tempo esperado até a colisão. Existem diversas formas de calcular a prioridade e as funções descrevendo os eventos do ambiente para uma mesma aplicação. Essas funções são modeladas dependendo de onde a biblioteca está sendo utilizada e de como os eventos do ambiente afetam a prioridade. Por exemplo, os limites de 20 segundos e 3 segundos utilizados na equação 5.2 poderiam ser ajustados para outros valores de acordo com as necessidades de projeto.

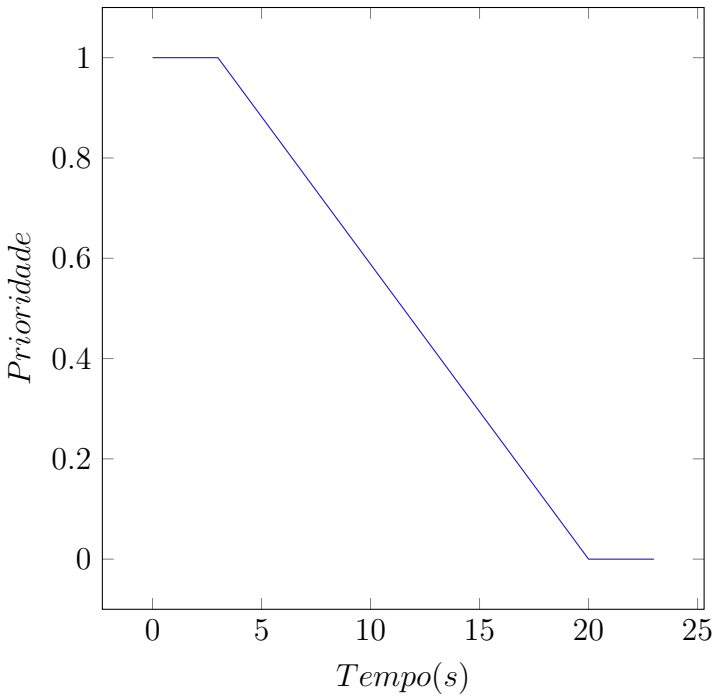

Figura 5.3: Prioridade baseada no tempo esperado até a colisão.

5.4 Gerenciamento de Banda Utilizando Dois Robôs

Em uma primeira simulação, dois robôs (R_1 e R_2) são operados no ambiente definido na figura 5.2 utilizando uma aplicação desenvolvida com o DBML. A resolução das imagens das câmeras dos robôs é de 640×480 para R_1 e 160×120 para R_2, implicando em um tamanho de mensagem de 84 KBytes e 21 KBytes, respectivamente, para os *frames* enviados (MANSOUR et al., 2012). A tabela 5.2 mostra os parâmetros de tamanho de mensagem para os robôs:

Tabela 5.2: Tamanho das mensagens na simulação com dois robôs.

$robot1/camera/dbm/message\_size\_in\_bytes$	84 000
$robot2/camera/dbm/message\_size\_in\_bytes$	21 000

Os robôs são controlados através de um ambiente com obstáculos e devem chegar até um ponto estabelecido. Utilizando o DBML, a cada tempo da simulação, a velocidade do robô e a distância do obstáculo a frente são utilizados para calcular a prioridade do tópico */camera* no instante de tempo. O *default_optimizer_node* executa o problema de otimização linear a cada um segundo como descrito no parâmetro

/dbm/optimization_rate_in_seconds. As tabelas 5.3 e 5.4 mostram a posição e a velocidade dos robôs no ambiente, respectivamente, em cada instante de tempo.

Tabela 5.3: Posição e velocidade de R_1.

	Robô 1 (R_1)		
Instante	Distância (cm)	Velocidade (cm/s)	Tempo para colisão (s)
1	200	10	20
2	150	5	30
3	30	5	6
4	100	10	10
5	50	10	5
6	150	10	15
7	50	20	2,5
8	180	5	36
9	200	5	40
10	200	4	50

Tabela 5.4: Posição e velocidade de R_2.

	Robô 2 (R_2)		
Instante	Distância (cm)	Velocidade (cm/s)	Tempo para colisão (s)
1	100	10	10
2	200	10	20
3	200	2	100
4	10	0	∞
5	150	0	∞
6	150	10	15
7	200	0	∞
8	100	10	10
9	70	10	7
10	30	10	3

Os resultados do algoritmo proposto podem ser comparados com um algoritmo de taxa fixa. O algoritmo estático divide a banda disponível entre os canais de comunicação na

proporção do tamanho das mensagens enviadas por eles. Desse modo, com uma largura de banda disponível de 1375 KBps (11 Mbps), é reservado 1100 KBps para R_1 e 275 KBps para R_2. Respeitando os limites de banda, o tópico */camera* irá enviar as mensagens a uma frequência de 13.09 Hz. Esse valor está acima do mínimo definido de 10 Hz de forma a evitar a degradação do vídeo (como definido na seção 5.2). Ou seja, os operadores conseguem manipular os robôs, mas em nenhuma situação a qualidade do vídeo irá atingir o valor de frequência desejado de 16 Hz. Esse valor de frequência pode ser útil nos casos onde uma colisão está próxima de acontecer.

De forma a avaliar o algoritmo de gerenciamento de banda proposto nesse trabalho, a tabela 5.5 e a figura 5.4 apresentam alguns resultados da simulação utilizando a biblioteca de gerenciamento de banda. O sistema prioriza os canais de comunicação aumentando a frequência e utilizando o máximo da banda disponível (11 Mbps), evitando desperdício de recursos. Entretanto, diferente do algoritmo estático, o BDML leva em consideração a necessidade de comunicação do tópico no momento.

Tabela 5.5: Frequências na aplicação de teleoperação para a simulação com dois robôs (t_c = Tempo para colisão; p = Prioridade; f = Frequência).

	Robô 1 (R_1)			Robô 2 (R_2)		
Instante	t_c (s)	P	f (Hz)	t_c (s)	p	f (Hz)
1	20	0,00	12,37	10	0,59	16
2	30	0,00	12,37	20	0,00	16
3	6	0,82	16	100	0,00	1,48
4	10	0,59	16	∞	0,00	1,48
5	5	0,88	16	∞	0,00	1,48
6	15	0,29	13	15	0,29	13,48
7	2,5	1,00	16	∞	0,00	1,48
8	36	0,00	12,37	10	0,59	16
9	40	0,00	12,37	7	0,76	16
10	50	0,00	12,37	3	1,00	16

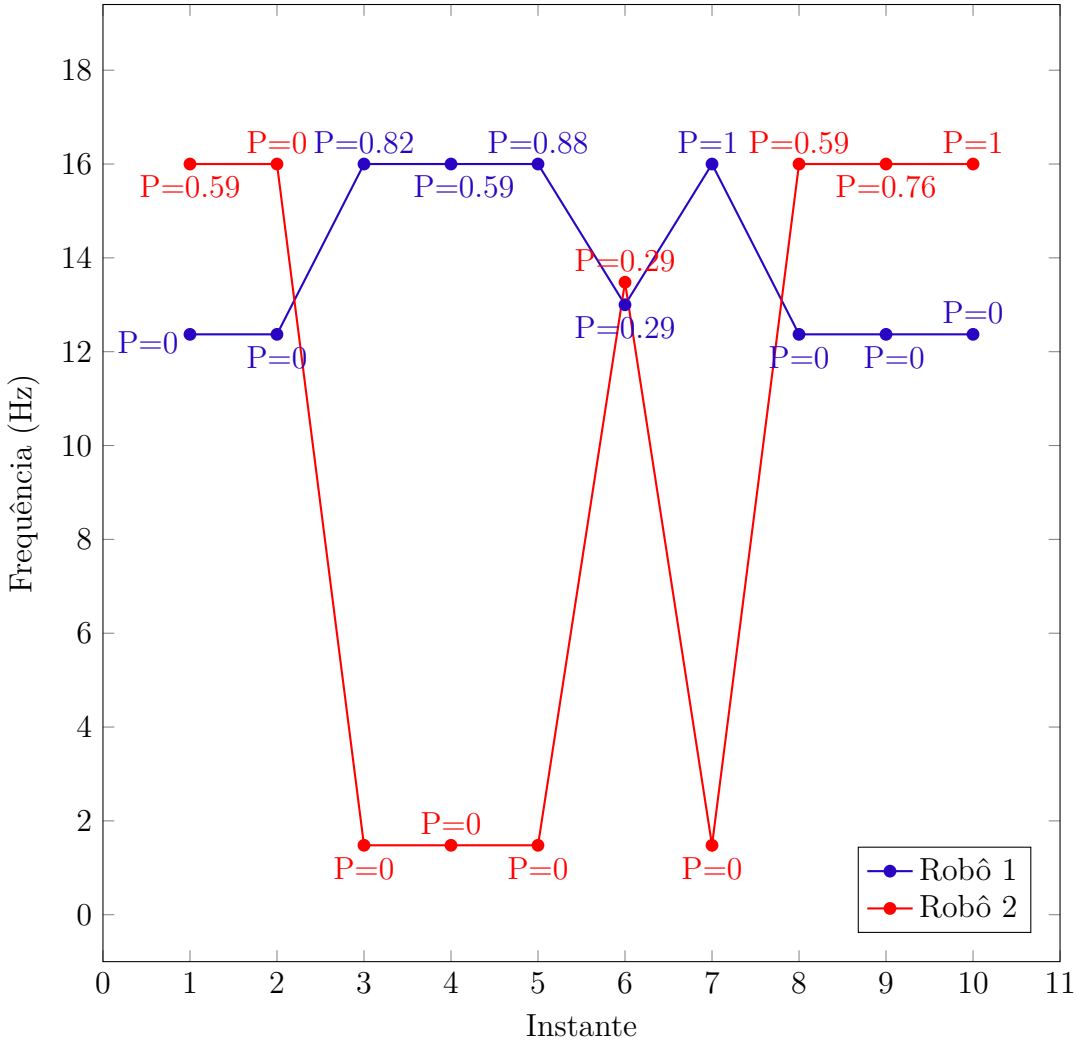

Figura 5.4: Prioridades na aplicação de teleoperação.

A figura 5.5 mostra no ambiente a posição de cada robô, sua velocidade (V), a distância para os obstáculos (D) e a frequência de envio do tópico */camera* (f) em cada um dos instantes utilizados na simulação.

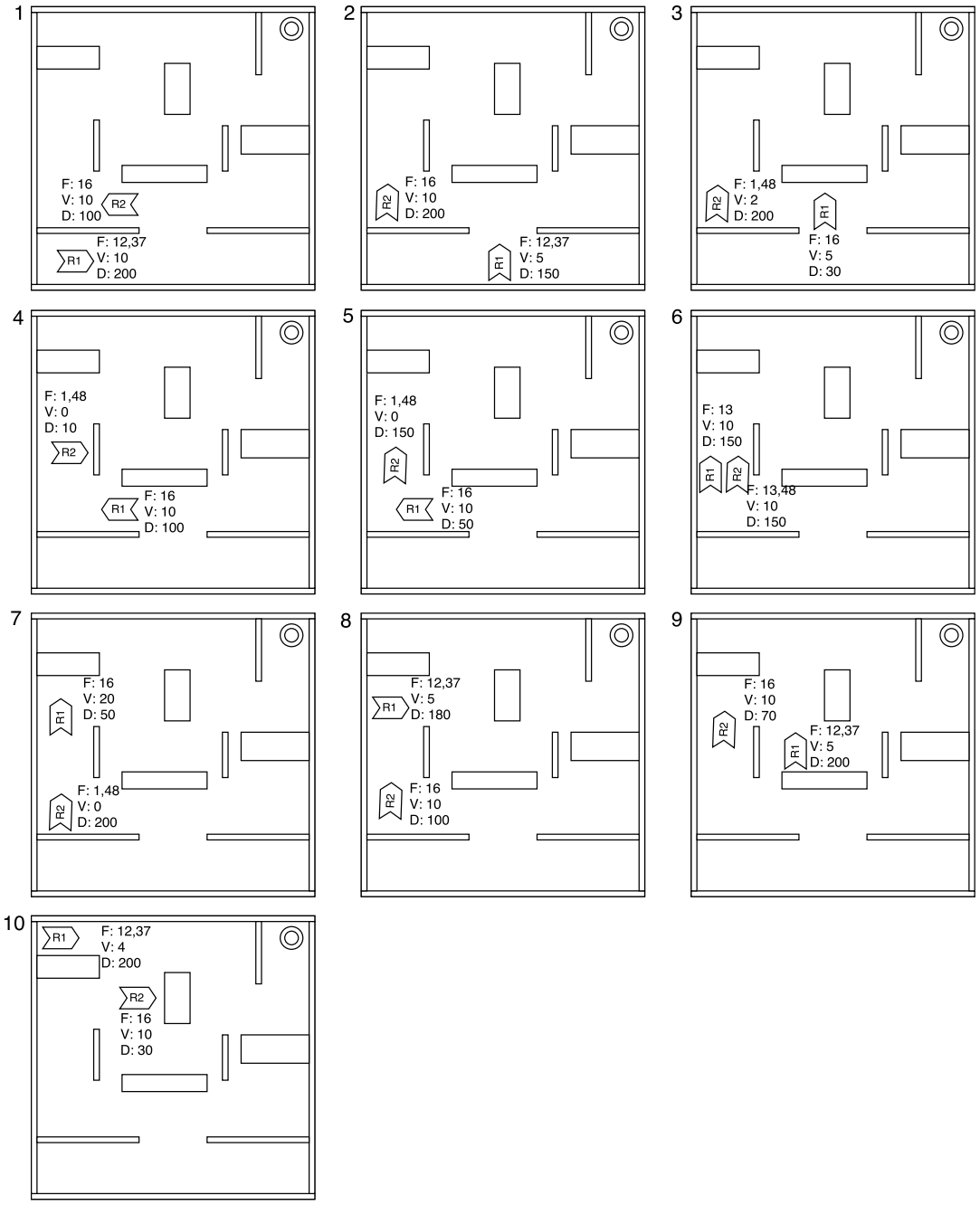

Figura 5.5: Posição, distância (D), velocidade (V) e frequência (f) dos robôs nos instantes da simulação.

No instante 1 pode-se notar como a biblioteca atribui uma maior frequência para os canais de comunicação mais prioritários enquanto garante o máximo de utilização da banda disponível para o sistema. Nesse caso, o robô R_1 encontra-se distante de um obstáculo com um tempo estimado para a colisão de 20 segundos e o robô R_2 possui uma prioridade mais alta uma vez que o seu tempo estimado para a colisão é de 10 segundos.

A biblioteca atribiu a frequência máxima para o robô R_2 e, de forma a utilizar o máximo da banda disponível, reservou 12.37 Hz para o robô R_1.

O instante 2 mostra uma simulação com $P_1 = 0$; $P_2 = 0$; $f_1 = 12,37\ Hz$; $f_2 = 16\ Hz$. Esse tempo mostra um caso onde as frequências não são proporcionais às prioridades. Nesse caso, o resultado está correto uma vez que o objetivo do algoritmo de otimização proposto é maximizar a largura de banda utilizada por todos os canais de comunicação (Eq. 4.6). Entretanto, considerando o cenário da aplicação, uma divisão das frequências proporcionais às prioridades poderia ser mais adequado. Assim, uma simples modificação do algoritmo proposto pode dividir a banda entre os canais de comunicação quando as prioridades são iguais a zero, uma vez que o otimizador encontra muitas soluções nesse caso. Desse modo, as frequências no tempo 1 podem ser recalculadas para $f_1 = 13,09\ Hz$ e $f_2 = 13,09\ Hz$.

Os instantes 3, 4, 5 e 7 exibem casos onde o robô R_1 está próximo de um obstáculo e o robô R_2 está parado. Nesses casos, a frequência de envio do robô R_2 pode ser diminuída para o mínimo possível já que não está sendo operado no momento e não corre o risco de colidir com os obstáculos. Nesses instantes, o sistema disponiblizou uma frequência de 16 Hz (máximo disponível) para o robô R_1 e 1,48 Hz para o robô R_2. Assim, é possível operar o robô R_1 com o máximo de *feedback* visual possível evitando a colisão com os obstáculos do ambiente. O instante 6 mostra um caso onde as prioridades são iguais e as frequências alocadas foram divididas entre os robôs de forma aproximada.

Os instantes 8, 9 e 10 mostram casos onde as prioridades do robô R_2 são maiores do que as prioridades do robô R_1. Nesses casos os dois robôs estão sendo operados, mas o robô R_1 está mais distante dos obstáculos que o robô R_2. O sistema atribiu então, corretamente, uma maior frequência para o robô R_2 permitindo que ele seja operado com uma maior qualidade de informações.

A banda total utilizada pelos canais de comunicação gerenciados em todos os tempos da simulação foi igual ao total de banda disponível para o sistema (11 Mbps). Como citado anteriormente, elevando a utilização da banda para próximo do máximo disponível evita-se despedício de recursos sem ultrapassar os limites estabelecidos.

5.5 Normalização das Prioridades

A seção 4.2 mostrou que as prioridades calculadas em função dos eventos do ambiente devem ser normalizadas para valores entre $[0 : 1]$, levando em consideração o tamanho da

mensagem e as prioridades dos outros canais de comunicação. Essa normalização garante que o tamanho da mensagem seja levado em consideração no cálculo das frequências. Sem esse ajuste, o otimizador não atribui as frequências de forma proporcional à prioridade, gerando resultados incoerentes com o objetivo da biblioteca.

A simulação apresentada na seção 5.4 mostrou os resultados utilizando a normalização das prioridades. Mas de forma a enfatizar a importância da normalização e comparar os resultados, a tabela 5.6 apresenta uma comparação entre os resultados obtidos com uma simulação feita sem o cálculo da normalização das prioridades e a simulação feita na seção anterior, com o cálculo da normalização. As duas simulações foram feitas utilizando os mesmos parâmetros e a mesma disposição e velocidade dos robôs no ambiente.

Tabela 5.6: Comparação entre simulações com normalização e sem normalização das prioridades.

	Posição e Prioridade				Normalizado		Não Normalizado	
Instante	R_1		R_2		R_1	R_2	R_1	R_2
	t_c (s)	p	t_c (s)	p	f (Hz)	f (Hz)	f (Hz)	f (Hz)
1	20	0,00	10	0,59	12,3	16	13,91	9,82
2	30	0,00	20	0,00	12,3	16	16	1,48
3	6	0,82	100	0,00	16	1,48	16	1,48
4	10	0,59	∞	0,00	16	1,48	16	1,48
5	5	0,88	∞	0,00	16	1,48	16	1,48
6	15	0,29	15	0,29	13	13,48	15,02	5,41
7	2,5	1,00	∞	0,00	16	1,48	16	1,48
8	36	0,00	10	0,59	12,37	16	13,91	9,82
9	40	0,00	7	0,76	12,37	16	13,25	12,47
10	50	0,00	3	1,00	12,37	16	12,37	16

Os instantes 1, 8 e 9 mostram claramente a vantagem de utilizar a normalização das prioridades. No instante 1, os robôs R_1 e R_2 tem prioridades iguais a 0,00 e 0,59, respectivamente. Na simulação não normalizada, o otimizador atribuiu uma frequência maior para o robô R_1, mesmo com o robô R_2 tendo uma prioridade mais alta. O mesmo aconteceu nos instantes 8 e 9, onde o robô com menor prioridade recebeu o maior valor de frequência. Por outro lado, na simulação com as prioridades normalizadas, em todos os casos o robô com maior prioridade recebeu um maior valor de frequência, atingindo o objetivo da biblioteca.

O tempo 2 e o tempo 6 mostram casos onde as prioridades dos robôs são iguais. Em ambos os casos, os resultados utilizando a normalização das prioridades foi melhor que os resultados sem a normalização por apresentarem uma distribuição de frequências mais equilibrada entre os robôs.

Como no restante dos tempos os resultados das duas simulações foram iguais, podemos concluir que, levando em consideração as simulações feitas, a normalização afeta os resultados positivamente, distribuindo a banda entre os robôs de forma mais proporcional às prioridades dos canais de comunicação.

5.6 Máxima Utilização da Banda

Nas duas simulações feitas, toda a banda disponível para a aplicação foi utilizada pelos canais de comunicação gerenciados pelo DBML. Entretanto, na aplicação de teleoperação proposta como exemplo, é necessário que o operador envie comandos para os robôs, como posição e velocidade. Mesmo com esses comandos gastando pouco recurso da rede, é importante reservar parte da banda para esse tipo de comunicação não gerenciada, evitando assim sobrecarga da rede e perda de pacotes. Isso é feito através do parâmetro */dbm/max_bandwidth_utilization*. Esse parâmetro atribui uma porcentagem da banda que deve ser utilizada pelos canais de comunicação gerenciados reservando a outra parte para qualquer tipo de comunicação que não esteja utilizando o DBML.

Diversos tipos de comunicação podem não ser gerenciados pelo DBML: comandos, serviços ROS, *ActionLib* (seção 3.3), tópicos com frequência fixa de envio, alocação de tarefas, etc. Qualquer funcionalidade que envie dados pela rede e que não seja criado utilizando o *DBMPublisher* é considerado um canal de comunicação não gerenciado. Esses canais de comunicação não são levados em consideração pelo otimizador do DBML e, por isso, devem ter uma parte da banda reservada para a sua comunicação. Caso contrário, pode haver colisão de pacotes na rede porque o otimizador do DBML irá atribuir frequências aos canais de comunicação gerenciados de forma a utilizar toda a banda disponível.

Para demonstrar a utilização desse parâmetro, foi feita uma simulação utilizando o mesmo ambiente já definido, mas considerando uma utilização de 70% da banda disponível. Como a banda total configurada para a aplicação é de 1350 KBps, os canais de comunicação gerenciados pelo DBML poderão utilizar 962,5 KBps. Dessa forma, qualquer tipo de comunicação não gerenciada poderá utilizar a rede sem que haja sobrecarga. A

tabela 5.7 mostra os resultados obtidos na simulação:

Tabela 5.7: Frequências na aplicação de teleoperação para a simulação com 70% de utilização da banda (T = Tempo para colisão; P = Prioridade; F = Frequência).

	Robô 1 (R_1)			Robô 2 (R_2)		
Instante	t_c (s)	p	f (Hz)	t_c (s)	p	f (Hz)
1	20	0,00	7,46	10	0,59	16
2	30	0,00	7,46	20	0,00	16
3	6	0,82	-	100	0,00	-
4	10	0,59	-	∞	0,00	-
5	5	0,88	-	∞	0,00	-
6	15	0,29	-	15	0,29	-
7	2,5	1,00	-	∞	0,00	-
8	36	0,00	7,46	10	0,59	16
9	40	0,00	7,46	7	0,76	16
10	50	0,00	7,46	3	1,00	16

Nos instantes 3, 4, 5, 6 e 7 o otimizador não encontrou nenhuma solução dentro das restrições de frequência mínima e máxima e da banda total disponível. Nesses casos, o DBML avisa ao projetista do sistema que não encontrou solução viável. A correção desse problema pode ser feita através do aumento da banda disponível para a aplicação ou do ajuste das frequências mínimas e máximas dos canais de comunicação gerenciados.

5.7 Tamanho de Mensagem Variável

Nas simulações apresentadas, o tamanho das mensagens varia entre os robôs, mas é fixo em todos os tempos da simulação. Como exemplo, o robô R_1 sempre envia as imagens da câmera com 84 Kbytes. Entretando, diversos tipos de comunicação não possuem esse comportamento: as mensagens podem variar no decorrer do tempo e um sistema dinâmico precisa tratar essas alterações.

Todos os canais de comunicação gerenciados pelo DBML tem o seu tamanho de mensagem monitorado pela biblioteca. Sempre que uma mensagem é publicada em um tópico, o DBML verifica se o tamanho da mensagem foi alterado e, caso isso aconteça, atualiza o tamanho da mensagem dinamicamente no *Parameter Server* (parâmetro

[nome_do_topico]/dbm/message_size_in_bytes). Esse ajuste do tamanho da mensagem em tempo real possibilita que o otimizador do DBML calcule as prioridades levando em consideração o tamanho real da mensagem naquele instante.

O resultado obtido pelo DBML em uma simulação com um tamanho variável das mensagens se mostrou-se condizente com os objetivos da biblioteca. Em todos os casos, as frequências atribuídas para os canais de comunicação respeitaram as prioridades e a utilização da banda foi maximizada (em todos os instantes, a utilização da banda foi o máximo disponível). A tabela 5.8 mostra uma simulação feita utilizando os mesmos parâmetros das simulações anteriores, porém com o envio de mensagens de tamanho variável (coluna M). Como pode ser visto, os resultados obtidos com a utilização da biblioteca independem da variação no tamanho da mensagem.

Tabela 5.8: Frequências na aplicação de teleoperação para a simulação com tamanhos de mensagens variáveis (M = Tamanho da mensagem; t_c = Tempo para colisão; p = Prioridade; f = Frequência).

	Robô 1 (R_1)				Robô 2 (R_2)			
Instante	M (Kbytes)	t_c (s)	p	f (Hz)	M (Kbytes)	t_c (s)	p	f (Hz)
1	12	20	0,00	11,92	77	10	0,59	16
2	11	30	0,00	16	21	20	0,00	16
3	7	6	0,82	16	21	100	0,00	16
4	6	10	0,59	16	32	∞	0,00	16
5	76	5	0,88	16	19	∞	0,00	8,36
6	13	15	0,29	16	8	15	0,29	16
7	21	2,5	1,00	16	67	∞	0,00	15,51
8	7	36	0,00	16	13	10	0,59	16
9	11	40	0,00	16	13	7	0,76	16
10	11	50	0,00	16	12	3	1,00	16

5.8 Simulações com Quatro Robôs

De forma mostrar o comportamento da biblioteca em um cenário contendo mais robôs, foi feita uma simulação no mesmo cenário descrito anteriormente (com a prioridade normalizada) adicionando dois outros robôs: R_3 e R_4.

Na simulação feita com quatro robôs, o robô R_3 envia imagens de uma câmera com resolução de 320×240 enquanto o robô R_4 envia as imagens da câmera com uma resolução de 480×360. Dessa forma, cada robô envia mensagens para a central de tamanho 42 KBytes e 63 KBytes, respectivamente. Essa simulação tem como objetivo verificar o comportamento do DBML em um ambiente com mais robôs enviando mensagens de tamanho fixo. Os outros parâmetros continuam os mesmos da simulação anterior. A tabela 5.9 mostra as configurações utilizadas na aplicação de simulação:

Tabela 5.9: Configurações da aplicação de simulação com quatro robôs.

$robot1/camera/dbm/frequency/min$	1Hz
$robot1/camera/dbm/frequency/max$	16Hz
$robot2/camera/dbm/frequency/min$	1Hz
$robot2/camera/dbm/frequency/max$	16Hz
$robot3/camera/dbm/frequency/min$	1Hz
$robot3/camera/dbm/frequency/max$	16Hz
$robot4/camera/dbm/frequency/min$	1Hz
$robot4/camera/dbm/frequency/max$	16Hz
$robot1/camera/dbm/message\_size\_in\_bytes$	84 000
$robot2/camera/dbm/message\_size\_in\_bytes$	21 000
$robot3/camera/dbm/message\_size\_in\_bytes$	42 000
$robot4/camera/dbm/message\_size\_in\_bytes$	63 000
$/dbm/max\_bandwidth\_in\_mbit$	11 Mbps
$/dbm/max\_bandwidth\_utilization$	100%
$/dbm/optimization\_frequency\_in\_seconds$	1 sec

A distância dos robôs para os obstáculos, sua velocidade e o tempo estimado para a colisão do robô com o obstáculo mais próximo podem ser vistos nas tabelas 5.10, 5.11, 5.12 e 5.13.

Tabela 5.10: Posição e velocidade de R_1 em uma simulação com quatro robôs.

	Robô 1 (R_1)		
Instante	Distância (cm)	Velocidade (cm/s)	Tempo para colisão (s)
1	200	10	20
2	150	15	15
3	100	10	10
4	50	0	∞
5	50	0	∞
6	50	0	∞
7	50	0	∞
8	50	5	10
9	25	3	8,33
10	10	5	2

Tabela 5.11: Posição e velocidade de R_2 em uma simulação com quatro robôs.

	Robô 2 (R_2)		
Instante	Distância (cm)	Velocidade (cm/s)	Tempo para colisão (s)
1	100	5	20
2	75	5	15
3	50	5	10
4	25	0	∞
5	25	0	∞
6	25	0	∞
7	25	0	∞
8	25	2	12,5
9	15	2	7,5
10	5	2	2,5

Tabela 5.12: Posição e velocidade de R_3 em uma simulação com quatro robôs.

	Robô 3 (R_3)		
Instante	Distância (cm)	Velocidade (cm/s)	Tempo para colisão (s)
1	200	0	∞
2	200	0	∞
3	200	0	∞
4	200	10	20
5	150	10	15
6	100	10	10
7	50	3	16,67
8	50	0	∞
9	50	0	∞
10	50	20	2,5

Tabela 5.13: Posição e velocidade de R_4 em uma simulação com quatro robôs.

	Robô 4 (R_4)		
Intante	Distância (cm)	Velocidade (cm/s)	Tempo para colisão (s)
1	100	0	∞
2	100	0	∞
3	100	0	∞
4	100	10	10
5	15	8	6,25
6	10	1	10
7	5	1	5
8	5	0	∞
9	5	0	∞
10	5	3	1,67

Como a largura de banda disponível para a aplicação não é suficiente para o envio das imagens da câmera em uma taxa que permita que o operador controle os quatro robôs de forma simultânea, os robôs foram operados de dois a dois. Nos instantes 1, 2 e 3 os robôs R_1 e R_2 foram operados enquanto os robôs R_3 e R_4 ficaram parados na mesma posição

e nos instantes 4, 5, 6 e 7 os robôs R_3 e R_4 foram operados enquanto os robôs R_1 e R_2 ficaram parados. Esse revezamento continuou nos tempos restantes.

Essa estratégia de operar apenas um conjunto de robôs enquanto um outro conjunto fica parado é importante para permitir que o operador receba uma taxa de *feedback* aceitável para que possa operar o robô com qualidade mesmo em um ambiente com uma baixa largura de banda disponível. Esse comportamento não seria possível em uma aplicação sem um gerenciamento dinâmico já que as taxas de transmissão dos tópicos seria fixa.

As tabelas 5.14, 5.15, 5.16 e 5.17 mostram as prioridades e as frequências de cada robô durante os tempos de simulação:

Tabela 5.14: Frequências na aplicação de teleoperação para R_1 em uma simulação com quatro robôs.

	Robô 1 (R_1)		
Instante	t_c (s)	p	f (Hz)
1	20	0,00	1
2	15	0,29	8,5
3	10	0,59	8,5
4	∞	0,00	1
5	∞	0,00	1
6	∞	0,00	1
7	∞	0,00	1
8	10	0,59	9,57
9	8,33	0,69	8,24
10	2	1,00	4,75

Tabela 5.15: Frequências na aplicação de teleoperação para R_2 em uma simulação com quatro robôs.

	Robô 2 (R_2)		
Instante	t_c (s)	p	f (Hz)
1	20	0,00	1
2	15	0,29	8,5
3	10	0,59	8,5
4	∞	0,00	1
5	∞	0,00	1
6	∞	0,00	1
7	∞	0,00	1
8	12,5	0,44	7,43
9	7,5	0,74	8,76
10	2	1,00	4,75

Tabela 5.16: Frequências na aplicação de teleoperação para R_3 em uma simulação com quatro robôs.

	Robô 3 (R_3)		
Instante	t_c (s)	p	f (Hz)
1	∞	0,00	3,83
2	∞	0,00	1
3	∞	0,00	1
4	20	0,00	3,83
5	15	0,29	5
6	10	0,59	8,5
7	16,67	0,20	3,83
8	∞	0,00	1
9	∞	0,00	1
10	2,5	1,00	4,75

Tabela 5.17: Frequências na aplicação de teleoperação para R_4 em uma simulação com quatro robôs.

	Robô 4 (R_4)		
Instante	t_c (s)	p	f (Hz)
1	∞	0,00	16
2	∞	0,00	3,83
3	∞	0,00	3,83
4	10	0,59	16
5	6,5	0,81	14,83
6	10	0,59	11,33
7	5	0,88	16
8	∞	0,00	3,83
9	∞	0,00	3,83
10	1,67	1,00	7,56

O gráfico 5.6 mostra um resumo dos resultados obtidos com a simulação utilizando quatro robôs:

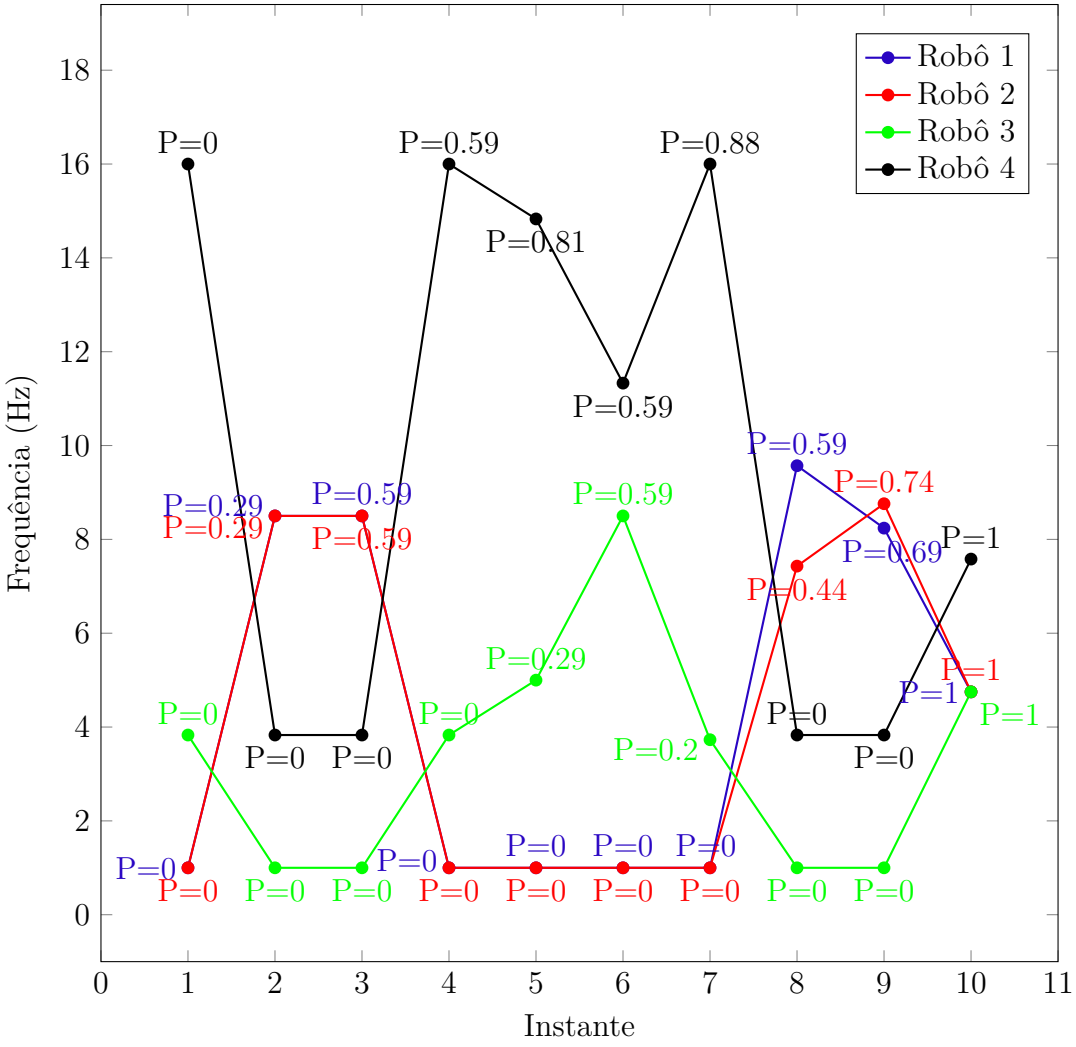

Figura 5.6: Prioridades na aplicação de teleoperação utilizando quatro robôs.

Como pode ser visto nos resultados acima, apesar do sistema ter distribuído a banda de acordo com as prioridades dos robôs em cada tempo da simulação, as frequências atribuídas ainda se mostraram baixas para que o operador receba o nível de *feedback* necessário. Os limites de frequência mínima e máxima configurados para a aplicação foram respeitados, mas apenas nos tempos 4, 5 e 7 o robô mais prioritário recebeu uma taxa de comunicação próximo do sugerido (16 Hz). Isso acontece porque os limites de banda disponíveis para a aplicação são baixos para permitir o nível de comunicação desejado (nessa simulação foram mantidas as mesmas restrições da simulação feita na seção 5.4 adicionando dois outros robôs). Para corrigir esse problema, é necessário aumentar a banda disponível.

É importante verificar que, mesmo a aplicação tendo uma deficiência de banda disponível para a comunicação, os resultados obtidos com o DBML são melhores que os

resultados obtidos em um algoritmo de banda fixa onde as frequências são distribuídas de forma proporcional ao tamanho da mensagem. Em uma solução desse tipo, os quatro robôs enviariam imagens da câmera a uma taxa de 6,54 Hz, bem abaixo do mínimo de 10 Hz definido na seção 5.2. Utilizando o DBML, os robôs com uma prioridade mais alta ficaram mais próximos ou ultrapassaram esse valor mínimo.

6 Conclusões e Trabalhos Futuros

Este trabalho apresentou o desenvolvimento de uma biblioteca de gerenciamento de banda em sistemas multirrobôs utilizando o ROS. O sistema prioriza os canais de comunicação de acordo com eventos do ambiente e oferece uma maior largura de banda para os canais mais importantes. Um exemplo de caso de uso demostrando como modelar um sistema e utilizar a biblioteca foi apresentado juntamente com os dados da simulação em uma aplicação de teleoperação.

A primeira parte do trabalho apresentou um estudo sobre sistemas multirrobôs e suas principais aplicações. Diversos trabalhos encontrados na bibliografia foram discutidos de forma a mostrar a importância de estudos nessa área. Pesquisas relacionadas ao gerenciamento de banda em SMR foram apresentadas para servirem de base para a construção do trabalho. Uma análise do ROS foi feita apresentando os principais conceitos e o funcionamento do framework sobre o qual a biblioteca foi construída. Essa parte do trabalho pode servir como uma fonte de consulta e pesquisa em trabalhos futuros utilizando o ROS.

O capítulo de desenvolvimento mostrou a construção da biblioteca de gerenciamento de banda utilizando o ROS. A forma padrão como uma taxa de comunicação de um tópico ROS é controlada através do uso da classe *ros::Rate* não permite um gerenciamento dinâmico da frequência de envio. A construção de uma classe que extende as funcionalidades do *ros::Rate* permitindo o gerenciamento dinâmico das frequências é a primeira contribuição desse trabalho.

A biblioteca proposta permite que os canais de comunicação representados pelos tópicos sejam priorizados de acordo com os eventos no ambiente. Essa priorização permite que estratégias de gerenciamento da banda calculem as frequências de envio dos canais de comunicação de acordo com a importância do canal naquele momento.

A biblioteca foi construída de forma a isolar o algoritmo de otimização da frequência em um módulo separado para que possa ser facilmente substituído por abordagens mais eficientes. Uma modelagem simples foi apresentada utilizando programação linear. Os

dados apresentados na simulação mostraram que o otimizador construído maximiza a utilização da banda respeitando as prioridades dos canais de comunicação, mas existem abordagens mais eficientes na literatura.

Para utilizar a biblioteca em sistemas existentes é necessário modificar o código-fonte desses sistemas. Como trabalho futuro, seria importante alterar a biblioteca proposta de forma a utilizá-la em sistemas existentes (nós e tópicos) sem a necessidade de mudanças em seu código-fonte.

Uma outra melhoria é criar um algoritmo de otimização e um gerenciamento dinâmico para imagens e vídeos da câmera. Imagens das câmeras dos robôs são informações transmitidas em diversos tipos de aplicações que impactam diretamente na utilização da banda. Esse tipo de informação (transmissão de vídeo e imagens) geralmente possui um tamanho expressivo para a aplicação e merecem ser tratados de forma exclusiva. Por exemplo, a biblioteca poderia reduzir a resolução da imagem de 640×480 para 160×120 quando existe uma redução na banda disponível.

Um desafio que deve ser abordado em trabalhos futuros é a criação de uma melhor capacidade do sistema em tratar os eventos em tempo real. A biblioteca proposta executa o algoritmo de otimização da banda em uma taxa fixa configurada por um parâmetro do sistema. Entretanto, rápidas mudanças no ambiente ou em situações nas quais os robôs encontram com outros robôs podem necessitar de uma resposta mais rápida de forma a garantir que o sistema não esteja aplicando limites de banda que estão desatualizados com a situação dos robôs no ambiente.

Como desenvolvido no presente trabalho, a biblioteca assume que o total de banda disponível para a aplicação é conhecida de antemão. Isso pode degradar a performance do sistema em ambientes reais, por exemplo em redes *wireless*, onde a largura de banda depende das localização física dos nós da rede e dos obstáculos do ambiente.

Por fim, experimentos em cenários reais podem ser executados de forma a acrescentar novas análises da utilização da biblioteca em aplicações reais.

Referências Bibliográficas

ANDREWS, M. et al. Providing quality of service over a shared wireless link. *IEEE Communications magazine*, Citeseer, v. 39, n. 2, p. 150–154, 2001.

ARAI, T.; PAGELLO, E.; PARKER, L. E. Editorial: Advances in multi-robot systems. *IEEE Transactions on robotics and automation*, v. 18, n. 5, p. 655–661, 2002.

ARKIN, R. C. Motor schema—based mobile robot navigation. *The International journal of robotics research*, Sage Publications, v. 8, n. 4, p. 92–112, 1989.

ARKIN, R. C. et al. Buzz, an instantiation of a schema-based reactive robotic system. Georgia Institute of Technology, 1993.

BALCH, T.; ARKIN, R. C. Communication in reactive multiagent robotic systems. *Autonomous Robots*, Springer, v. 1, n. 1, p. 27–52, 1994.

BALCH, T.; HYBINETTE, M. Social potentials for scalable multi-robot formations. In: IEEE. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. [S.l.], 2000. v. 1, p. 73–80.

BASTOS, G. S.; RIBEIRO, C. H. C.; SOUZA, L. E. de. Variable utility in multi-robot task allocation systems. In: IEEE. *Robotic Symposium, 2008. LARS'08. IEEE Latin American*. [S.l.], 2008. p. 179–183.

BENI, G. The concept of cellular robotic system. In: IEEE. *Intelligent Control, 1988. Proceedings., IEEE International Symposium on*. [S.l.], 1988. p. 57–62.

BOTELHO, S. C.; ALAMI, R. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In: IEEE. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. [S.l.], 1999. v. 2, p. 1234–1239.

BRAUN, C. W.; PETR, D. W. *Performance evaluation of dynamic and static bandwidth management methods for ATM networks*. Dissertação (Mestrado) — University of Kansas, Electrical Engineering and Computer Sciences, 1994.

CHASKAR, H. M.; MADHOW, U. Fair scheduling with tunable latency: a round-robin approach. *IEEE/ACM Transactions on Networking (TON)*, IEEE Press, v. 11, n. 4, p. 592–601, 2003.

CHEN, J. Y.; HAAS, E. C.; BARNES, M. J. Human performance issues and user interface design for teleoperated robots. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, IEEE, v. 37, n. 6, p. 1231–1245, 2007.

CHOI, S.-i.; HUH, J.-d. Dynamic bandwidth allocation algorithm for multimedia services over ethernet pons. *ETRI journal*, Electronics and Telecommunications Research Institute, v. 24, n. 6, p. 465–468, 2002.

CRAWLEY, E.; NAIR, R. Rfc2386-1998. *A framework for QoS-based routing in the internet. USA: Network Working Group*, v. 5, 1998.

DBML-WIKI. 2015. *http : //wiki.ros.org/dynamic_bandwidth_manager/*. Online; acessado em 14-Junho-2015.

DORIGO, M. et al. *Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008, Brussels, Belgium, September 22-24, 2008, Proceedings*. [S.l.]: Springer, 2008.

ERP, J. B. V.; PADMOS, P. Image parameters for driving with indirect viewing systems. *Ergonomics*, Taylor & Francis, v. 46, p. 1471–1499, 2003.

FARINELLI, A.; IOCCHI, L.; NARDI, D. Multirobot systems: a classification focused on coordination. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, IEEE, v. 34, n. 5, p. 2015–2028, 2004.

FERRARI, C. et al. Multirobot motion coordination in space and time. *Robotics and autonomous systems*, Elsevier, v. 25, n. 3, p. 219–229, 1998.

FONG, T.; THORPE, C.; BAUR, C. Multi-robot remote driving with collaborative control. *Industrial Electronics, IEEE Transactions on*, IEEE, v. 50, n. 4, p. 699–704, 2003.

FOX, D. et al. A probabilistic approach to collaborative multi-robot localization. *Autonomous robots*, Springer, v. 8, n. 3, p. 325–344, 2000.

FRENCH, J.; GHIRARDELLI, T. G.; SWOBODA, J. The effect of bandwidth on operator control of an unmanned ground vehicle. In: NTSA. *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*. [S.l.], 2003. v. 2003.

FUKUDA, T.; NAKAGAWA, S. Dynamically reconfigurable robotic system. In: IEEE. *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*. [S.l.], 1988. p. 1581–1586.

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Education, 1994.

GANZ, A.; GANZ, Z.; WONGTHAVARAWAT, K. *Multimedia Wireless Networks: Technologies, Standards and QoS*. [S.l.]: Pearson Education, 2003.

GERKEY, B.; VAUGHAN, R. T.; HOWARD, A. The player/stage project: Tools for multi-robot and distributed sensor systems. In: *Proceedings of the 11th international conference on advanced robotics*. [S.l.: s.n.], 2003. v. 1, p. 317–323.

GERKEY, B. P.; MATARIC, M. J. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In: IEEE. *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. [S.l.], 2003. v. 3, p. 3862–3868.

GERKEY, B. P.; MATARIĆ, M. J. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, SAGE Publications, v. 23, n. 9, p. 939–954, 2004.

HOWARD, A. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, SAGE Publications, v. 25, n. 12, p. 1243–1256, 2006.

HULL, B.; JAMIESON, K.; BALAKRISHNAN, H. Bandwidth management in wireless sensor networks. In: ACM. *Proceedings of the 1st international conference on Embedded networked sensor systems*. [S.l.], 2003. p. 306–307.

KLAVINS, E. Communication complexity of multi-robot systems. In: *Algorithmic Foundations of Robotics V*. [S.l.]: Springer, 2004. p. 275–291.

KRAMER, J.; SCHEUTZ, M. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, Springer, v. 22, n. 2, p. 101–132, 2007.

LAURENT, S. S. et al. *Programming web services with XML-RPC*. [S.l.]: "O'Reilly Media, Inc.", 2001.

LIMA, P. U.; CUSTODIO, L. M. Multi-robot systems. In: *Innovations in robot mobility and control*. [S.l.]: Springer, 2005. p. 1–64.

LUENBERGER, D. G. *Introduction to linear and nonlinear programming*. [S.l.]: Addison-Wesley Reading, MA, 1973.

LUMELSKY, V. J.; HARINARAYAN, K. Decentralized motion planning for multiple mobile robots: The cocktail party model. In: *Robot colonies*. [S.l.]: Springer, 1997. p. 121–135.

MACLENNAN, B. *Synthetic ethology: An approach to the study of communication*. [S.l.]: University of Tennessee, Computer Science Department, 1990.

MANSOUR, C. et al. Event-based dynamic bandwidth management for teleoperation. In: IEEE. *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. [S.l.], 2011. p. 229–233.

MANSOUR, C. et al. Dynamic bandwidth management for teleoperation of collaborative robots. In: IEEE. *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*. [S.l.], 2012. p. 1861–1866.

MARTINEZ, A.; FERNÁNDEZ, E. *Learning ROS for robotics programming*. [S.l.]: Packt Publishing Ltd, 2013.

MOURIKIS, A. I.; ROUMELIOTIS, S. I. Optimal sensor scheduling for resource-constrained localization of mobile robot formations. *Robotics, IEEE Transactions on*, IEEE, v. 22, n. 5, p. 917–931, 2006.

NAHRSTEDT, K.; SHAH, S. H.; CHEN, K. Cross-layer architectures for bandwidth management in wireless networks. In: *Resource Management in Wireless Networking*. [S.l.]: Springer, 2005. p. 41–62.

PARKER, L. E. L-alliance: Task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics*, Taylor & Francis, v. 11, n. 4, p. 305–322, 1996.

PARKER, L. E. Current state of the art in distributed autonomous mobile robotics. In: *Distributed Autonomous Robotic Systems 4*. [S.l.]: Springer, 2000. p. 3–12.

PERKINS, C. E.; BHAGWAT, P. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 1994. v. 24, n. 4, p. 234–244.

PEYRAVIAN, M.; GÜN, L. Bandwidth efficiency of the networking broadband services architecture: A case study. *Telecommunication Systems*, Springer, v. 5, n. 2, p. 273–301, 1996.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: *ICRA workshop on open source software*. [S.l.: s.n.], 2009. v. 3, p. 5.

ROS-WIKI. 2015. Http://wiki.ros.org/. Online; acessado em 02-Fevereiro-2015.

RYBSKI, P. E. et al. Performance of a distributed robotic system using shared communications channels. *Robotics and Automation, IEEE Transactions on*, IEEE, v. 18, n. 5, p. 713–727, 2002.

SHEN, W.-M.; SALEMI, B.; WILL, P. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *Robotics and Automation, IEEE Transactions on*, IEEE, v. 18, n. 5, p. 700–712, 2002.

SHERIDAN, T. B. *Telerobotics, automation, and human supervisory control*. [S.l.]: MIT press, 1992.

SIROUSPOUR, S.; SETOODEH, P. Multi-operator/multi-robot teleoperation: an adaptive nonlinear control approach. In: IEEE. *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. [S.l.], 2005. p. 1576–1581.

SUGIYAMA, H.; TSUJIOKA, T.; MURATA, M. Qos routing in a multi-robot network system for urban search and rescue. In: IEEE. *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*. [S.l.], 2006. v. 2, p. 5–pp.

SUGIYAMA, H.; TSUJIOKA, T.; MURATA, M. Integrated operations of multi-robot rescue system with ad hoc networking. In: IEEE. *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*. [S.l.], 2009. p. 535–539.

ŠVESTKA, P.; OVERMARS, M. H. Coordinated path planning for multiple robots. *Robotics and autonomous systems*, Elsevier, v. 23, n. 3, p. 125–152, 1998.

THRUN, S.; BURGARD, W.; FOX, D. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In: IEEE. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. [S.l.], 2000. v. 1, p. 321–328.

TIPSUWAN, Y. et al. An auction-based dynamic bandwidth allocation with sensitivity in a wireless networked control system. *Computers & Industrial Engineering*, Elsevier, v. 57, n. 1, p. 114–124, 2009.

WANG, X. G. et al. A qos-based bandwidth management scheme in heterogeneous wireless networks. *International Journal of Simulation Systems, Science and Technology*, v. 5, n. 1-2, p. 9–17, 2004.

YAÏCHE, H.; MAZUMDAR, R. R.; ROSENBERG, C. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Transactions on Networking (TON)*, IEEE Press, v. 8, n. 5, p. 667–678, 2000.

YAMASHITA, A. et al. Motion planning for cooperative transportation of a large object by multiple mobile robots in a 3d environment. In: IEEE. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. [S.l.], 2000. v. 4, p. 3144–3151.

YINGYING, D.; YAN, H.; JINGPING, J. Multi-robot cooperation method based on the ant algorithm. In: IEEE. *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*. [S.l.], 2003. p. 14–18.