UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

rqt_mrta: Um Pacote ROS para Configuração e Supervisão de Arquiteturas MRTA

Adriano Henrique Rossette Leite

Itajubá, 14 de novembro de 2017

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Adriano Henrique Rossette Leite

rqt_mrta: Um Pacote ROS para Configuração e Supervisão de Arquiteturas MRTA

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Área de Concentração: Automação e Sistemas Elétricos Industriais

Orientador: Prof. Dr. Guilherme Sousa Bastos

14 de novembro de 2017
Itajubá

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

rqt_mrta: Um Pacote ROS para Configuração e Supervisão de Arquiteturas MRTA

Adriano Henrique Rossette Leite

Dissertação aprovada por banca examinadora em 15 de Dezembro de 2017, conferindo ao autor o título de **Mestre em Ciências em Engenharia Elétrica.**

Banca Examinadora:
Prof. Dr. Guilherme Sousa Bastos (Orientador)
Prof. Dr. Edson Prestes
Prof. Dr. Laércio Augusto Baldochi Júnior

Itajubá
2017

Adriano Henrique Rossette Leite
rqt_mrta: Um Pacote ROS para Configuração e Supervisão de Arquiteturas MRTA/ Adriano Henrique Rossette Leite. – Itajubá, 14 de novembro de 2017-
50 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Guilherme Sousa Bastos

Dissertação (Mestrado)
Universidade Federal de Itajubá
Programa de Pós-Graduação em Engenharia Elétrica, 14 de novembro de 2017.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 07:181:009.3

Adriano Henrique Rossette Leite

rqt_mrta: Um Pacote ROS para Configuração e Supervisão de Arquiteturas MRTA

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Trabalho aprovado. Itajubá, 15 de Dezembro de 2017:

Prof. Dr. Guilherme Sousa Bastos
Orientador

Prof. Dr. Edson Prestes

Prof. Dr. Laércio Augusto Baldochi Júnior

Itajubá
14 de novembro de 2017

Agradecimentos

À Deus ...

À meus amigos e familiares ...

Ao meu orientador ...

À banca examinadora, ...

Aos amigos e colegas do LRO, ...

À Capes pelo apoio financeiro durante estes 2 anos.

À Fapemig pelo financiamento do projeto de pesquisa TEC-APQ-00666-12, o qual possibilitou a compra do robô utilizado neste trabalho (Pioneer-3DX).

"Colocar a frase aqui."
(Autor)

Resumo

Este trabalho apresenta o desenvolvimento do pacote baseado em ROS *rqt_mrta*, o qual fornece um *plugin* de interface gráfica de usuário para a parametrização amigável de arquiteturas para a resolução de problemas de alocação de tarefas em um sistema multirrobô. Além disso, em tempo de execução, o *plugin* dispõe elementos gráficos para a supervisão e monitoramento da arquitetura e do sistema multirrobô.

Palavras-chaves: MRS. MRTA. ROS.

Abstract

This work presents ...

The *rqt_mrta* provides a GUI plugin for configuring and monitoring multi-robot task allocation architectures durings runtime.

Key-words: MRS. MRTA. ROS.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
API	*Application Programming Interface*
CBR	Competição Brasileira de Robótica
CNP	*Contract Net Protocol*
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
LRO	Laboratório de Robótica
MAS	*Multi-Agent System*
MRS	*Multi-Robot System*
MRTA	*Multi-Robot Task Allocation*
OAP	*Optimal Assignment Problem*
P3DX	*Adept MobileRobots Pioneer 3 DX*
ROS	*Robot Operating System*
UNIFEI	Universidade Federal de Itajubá
XML	*Extensible Markup Language*

Lista de símbolos

n	Número inteiro
t	Tempo
T	Período

Sumário

1 Introdução

1.1 Motivação

Aplicações de robótica onde vários robôs interagem entre si e também com o ambiente em que estão inseridos são chamadas de sistemas multirrobô, do inglês *Multi-robot systems* (MRS). Um sistema multirrobô possui diversas vantagens sobre sistemas com apenas um robô. Entre elas se encontram o ganho de flexibilidade, a simplificação de tarefas complexas e o aumento da eficiência no uso de recursos, de desempenho do sistema como um todo e da robustez através de redundâncias (CAO; FUKUNAGA; KAHNG, 1997; DUDEK et al., 1996; ZLOT et al., 2002). Entretanto, aplicações dessa natureza demandam arquiteturas complexas para o controle da coordenação dos robôs envolvidos e, intrinsecamente, possuem problema de escalabilidade nos processos computacionais e na rede de comunicação.

Um dos problemas mais desafiadores em aplicações de vários robôs é denominado *alocação de tarefa* (MRTA, acrônimo para *Multi-Robot Task Allocation*), que busca atribuir a execução de um conjunto de tarefas para um grupo de robôs sujeitos à limitações de forma que o desempenho geral do sistema seja otimizado. Esse tipo de problema pode ser resolvido por arquiteturas que se baseiam em modelos de organização que podem ser encontrados no cotidiano. Suas premissas limitam a abrangência de problemas que podem ser resolvidos pela a arquitetura. Com isso, há uma grande quantidade de arquiteturas formuladas. E com o intuito de classificá-las, Gerkey e Matarić (2004) sugeriu uma taxonomia independente do domínio para a classificação de problemas MRTA a partir da análise de várias delas (PARKER, 1998; GERKEY; MATARIC, 2002; BOTELHO; ALAMI, 1999; WERGER; MATARIĆ, 2000; FRANK, 2005; STENTZ; DIAS, 1999; CHAIMOWICZ; CAMPOS; KUMAR, 2002).

Com o advento do ROS (do inglês *Robot Operating System*) (QUIGLEY et al., 2009), vários sistemas inteligentes puderam ser reutilizados em diversas aplicações de robótica, tais como: localização (LI; BASTOS, 2017), navegação robótica, gerenciamento de largura de banda (JULIO; BASTOS, 2015), planejamento e escalonamento de ações e tarefas (FOX; LONG, 2003; MANNE, 1960), algoritmos de inteligência artificial (SCHNEIDER et al., 2015; WATKINS; DAYAN, 1992), entre outros. Sendo um *middleware* dedicado para aplicações robóticas, ele possibilitou a integração de trabalhos desenvolvidos por equipes distintas de pesquisa em robótica, pois ele simplifica o desenvolvimento de processos e dá suporte à comunicação e interoperabilidade deles. Desta forma, pesquisadores de robótica podem ater-se ao desenvolvimento de projetos dentro da sua especialização, necessitando apenas configurar os demais pacotes para a execução da aplicação. Problemas

que anteriormente possuíam difícil solução em termos de *software*, foram simplificados a partir da modularidade proporcionada pelo ROS.

Apesar da vasta existência de arquiteturas de alocação de tarefa para sistemas multirrobôs, houveram poucas tentativas de aproximação genérica delas em projetos baseados em ROS. Isto é, mediante a pesquisa realizada na literatura e no repositório do ROS, não foram encontrados projetos baseados em ROS que se aproximam de uma arquitetura MRTA configurável para qualquer problema MRTA que satisfaça suas premissas. Reis e Bastos (2015) mostraram as facilidades que o ROS oferece na implementação da arquitetura ALLIANCE, proposta por Parker (1998). Contudo, essa aproximação atende apenas o problema aplicado nesse trabalho. Já em *auction_methods_stack*[1], ainda que não possui documentação alguma, está contido um grupo de pacotes que foi desenvolvido em uma versão antiga do ROS e nunca mais foi atualizado.

Com isso, verifica-se a necessidade de ferramentas que facilitem a utilização de arquiteturas de alocação de tarefa em sistemas multirrobô para o *framework* ROS para incentivar o desenvolvimento de aproximações genéricas dessas arquiteturas.

1.2 Objetivos

Esse trabalho propõe desenvolver um pacote ROS, denominado *rqt_mrta*, que facilite a utilização de arquiteturas de alocação de tarefa no ROS para sistemas multirrobô. Esse pacote fornece uma interface gráfica que foi desenvolvida com o intuito de disponibilizar serviços para dois tipos de clientes: (1) desenvolvedor e (2) usuário de arquitetura MRTA. Seus serviços são:

- Cadastro de novas arquiteturas no ROS para seu uso na solução de problemas de alocação de tarefa em sistemas multirrobô;
- Criação de novos projetos contendo a definição de um problema de alocação de tarefa;
- Configuração da arquitetura escolhida para resolver o problema MRTA;
- Armazenamento dos dados de configuração no projeto criado;
- Monitoramento da comunicação dos robôs do sistema no ROS em tempo de execução;
- Monitoramento das atividades dos robôs no sistema em tempo de execução.

Além disso, será apresentado neste trabalho uma aproximação genérica da arquitetura Alliance, a qual foi desenvolvida em um pacote ROS chamado *alliance*.

[1] <https://github.com/joaoquintas/auction_methods_stack>

1.3 Contribuições

A partir da elaboração deste trabalho, os seguintes pacotes baseados em ROS foram obtidos e disponibilizados para a comunidade ROS:

- ***rqt_mrta***[2]: uma interface gráfica usuário que facilita a utilização de arquiteturas que resolvem o problema de alocação de tarefa em um sistema multirrobô.
- ***alliance***[3]: uma aproximação genérica da arquitetura Alliance.
- ***alliance_msgs***[4]: contém definição das mensagens utilizadas na comunicação entre os robôs na arquitetura Alliance pelo pacote *alliance*.
- ***rqt_alliance***[5]: uma interface gráfica de usuário que monitora as variáveis de motivação dos robôs em um sistema que utiliza o pacote *alliance* para a alocação de tarefas.

1.4 Estrutura do Trabalho

No capítulo 2, ... Prosseguindo para o capítulo 3, ... A seguir, ..., no capítulo 4. Por fim, o capítulo 5 ...

[2] <http://wiki.ros.org/rqt_mrta>
[3] <http://wiki.ros.org/alliance>
[4] <http://wiki.ros.org/alliance_msgs>
[5] <http://wiki.ros.org/rqt_alliance>

2 Revisão teórica

2.1 Sistema Multirrobô

2.1.1 Sistema: cooperativo *versus* competitivo

2.1.2 Composição: homogêneo *versus* heterogêneo

2.1.3 Coordenação: deliberativa *versus* reativa

2.1.4 Comunicação: implícita *versus* explícita

2.1.5 Arquitetura: centralizada *versus* distribuída

2.2 Alocação de Tarefa em Sistema Multirrobô

Um dos problemas mais desafiadores em aplicações multirrobô leva o nome *alocação de tarefa*, na língua inglesa, *Multi-Robot Task Allocation*(MRTA). Problemas dessa natureza buscam como solução atribuir otimamente um conjunto de robôs para um conjunto de tarefas de maneira que o desempenho geral de um sistema sujeito a um conjunto de limitações seja otimizado.

falar sobre o conteúdo desta seção

2.2.1 Definição Formal

Zlot e Stentz (2006) define o problema de alocação de tarefa em um sistema multirrobô conforme o seguinte problema de atribuição ótima (OAP - *Optimal Assignment Problem*) estático.

Definição 2.1. (*Alocação de Tarefa em um Sistema Multirrobô*) Sejam dados um conjunto T, um conjunto R e uma função de custo para cada subconjunto de robots $r \in R$ que especifique o custo de performance para cada subconjunto de tarefas, $c_r : 2^T \rightarrow \mathbb{R}_+ \cup \{\infty\}$: procure a alocação $A^* \in R^T$ que minimiza a função objetivo global $C : R^T \rightarrow \mathbb{R}_+ \cup \{\infty\}$. —

Note que para que um algoritmo consiga encontrar uma solução ótima para este problema, é necessário levar em consideração todo o espaço de alocação R^T, cujo tamanho aumenta exponencialmente em função do número de tarefas e robôs no sistema. No entanto, como um problema MRTA possui natureza dinâmica, que varia com o tempo mudanças do ambiente, a solução de um OAP estático pode ser mais aplicável.

2.2.2 Taxonomia

Gerkey e Matarić (2004) sugeriram uma taxonomia de três eixos independente do domínio para a classificação de problemas de alocação de tarefas em sistemas multirrobôs.

O primeiro eixo determina o *tipo dos robôs* que compõem o problema. Os tipos de robôs possíveis são: *ST* (acrônimo para *Single-Task*) ou *MT* (acrônimo para *Multi-Task*). Problemas que envolvem robôs que só podem executar uma tarefa por vez são compostos por robôs do tipo *ST*. Entretanto, se houver pelo menos um robô capaz de executar mais de uma tarefa simultaneamente, então esse problema é composto por robôs do tipo *MT*.

O segundo eixo da taxonomia determina o *tipo das tarefas* que compõem o problema. Nesse caso, são possíveis os tipos: *ST* (acrônimo para *Single-Robot*) ou *MR* (acrônimo para *Multi-Robot*). Problemas cujo tipo das tarefas é *SR*, diz-se que todas as tarefas envolvidas só podem ser executadas por um robô. Porém, quando o tipo das tarefas envolvidas é *MR*, diz-se que existe tarefas que podem ser executadas por mais de um robô.

O terceiro eixo, por sua vez, determina o *tipo da alocação* do problema, o qual pode assumir os valores: *IA* (acrônimo para *Instantaneous Assignment*) ou *TA* (acrônimo para *Time-extended Assignment*). O primeiro caso, *IA*, diz repeito à problemas MRTA onde as alocações das tarefas para os robôs são realizadas instantaneamente, sem levar em consideração o estado futuro do sistema. Por outro lado, em problemas cujo tipo de alocação é *TA*, além de conhecido o estado atual de cada rôbo e do ambiente, também é conhecido o conjunto de tarefas que precisarão ser alocadas no futuro. Neste último caso, diversas tarefas são alocadas para um robô, o qual deve executar cada alocação conforme seu agendamento. De acordo com Bastos, Ribeiro e Souza (2008), quando o tipo de alocação do problema MRTA é *IA*, o número de robôs é superior ao número de tarefas alocadas e quando *TA*, o oposto acontece. Isso se deve ao fato de que, em problemas MRTA cujo tipo de alocação é *IA*, o número de robôs no sistema é capaz de suprir a taxa de tarefas a serem atribuídas, de modo que é muito provável que haverão robôs ociosos no sistema; enquanto, naqueles cujo tipo de alocação é *TA*, o número de robôs que compõem o sistema não é suficiente para atender a taxa de tarefas a serem alocadas no sistema.

É visto na Figura 1 uma representação gráfica da taxonomia de Gerkey e Matarić (2004) para a classificação de problemas MRTA (*Multi-Robot Task Allocation*), onde pode-se notar que existem oito classes de problemas MRTA bem definidos.

2.2.2.1 Arquitetura MRTA

Possue a função de solucionar o problema de alocação de tarefas em um dado sistema multirrobô.

Basicamente existem duas formas de implementação de uma arquitetura: iterativa e instantânea. As aproximações iterativas apresentam uma dinâmica progressiva para que

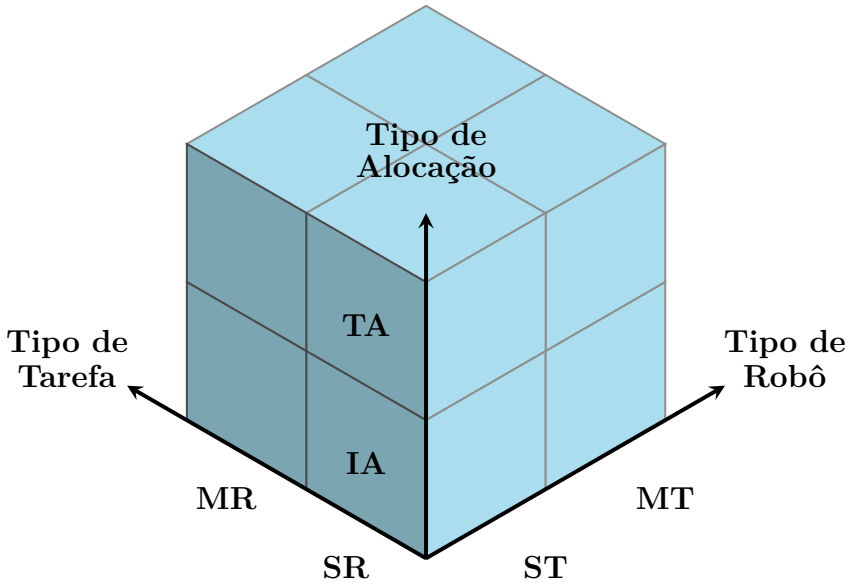


Figura 1 – Representação visual da taxonomia de três eixos sugerida por Gerkey e Matarić (2004).

ocorra um alocação, enquanto as arquiteturas que esperam uma resposta instantânea dos robôs do sistema ...

2.2.3 Arquiteturas baseadas em Comportamento

- **ALLIANCE**: (PARKER, 1998);
- **L-ALLIANCE**: (PARKER, 1996);

2.2.4 Arquiteturas baseadas em Negociação

Muitas arquiteturas baseadas em regras de negociação são variações do Protocolo de Rede de Contrato, do inglês *Contract Net Protocol* (CNP), sugerido por Smith (1980). Este mecanismo é utilizado para a atribuição de tarefas com controle distribuído por meio de um processo de negociação em sistemas multiagente. A Tabela 1 mostra uma comparação realizada por Yan, Jouandeau e Cherif (2013) de três abordagens clássicas do CNP. A primeira delas, abordagens baseada em regras de mercado, é composta por indivíduos competitivos cujo objetivo é se beneficiar maximizando o seu lucro e minimizando seus custos mesmo quando se trata de seus colegas de trabalho (ZLOT; STENTZ, 2006). Na sequência, abordagens baseadas em regras de leilão, uma excelente escolha para a alocação de recursos escassos (GERKEY; MATARIC, 2002). E, por fim, abordagens baseadas em regras de comércio, que são compostas por compradores e vendedores, cuja relação consiste em trocas: o comprador usa dinheiro para adquirir bens e serviços dos vendedores e os vendedores recebem o dinheiro para a entrega dos bens ou serviços (YAN; JOUANDEAU; CHERIF, 2011).

Tabela 1 – Comparação de três variações do CNP (YAN; JOUANDEAU; CHERIF, 2013).

	Abordagens baseadas em mercado	**Abordagens baseadas em leilão**	**Abordagens baseadas em comércio**
Modelo de comunicação na negociação	*publish / subscribe*	*publish / subscribe*	*apply / allocate*
Algoritmo de alocação de tarefa	algoritmo guloso	algoritmo guloso	algoritmo guloso
Abilidade de alocação de tarefa por iteração	uma tarefa	uma tarefa	várias tarefas
Determinação do papel dos robôs	voluntária	voluntária	negociação
Consideração de utilidade	custo e benefício	custo	custo
Reatribuição de tarefa	permitida	não permitida	permitida
Complexidade de comunicação	$O(1)$/licitante, $O(n)$/leiloeiro	$O(1)$/licitante, $O(n)$/leiloeiro	$O(1)$/comprador, $O(n)$/vendedor
Complexidade de computação	$O(n)$	$O(n)$	$O(n)$

Segue abaixo, exemplos de arquiteturas de alocação de tarefa em sistema multirrobô baseadas em modelos de negociação.

- **M+**: foi a primeira variação do CNP para a alocação e realização de tarefas em sistemas multirrobô (BOTELHO; ALAMI, 1999);
- **FMS**: do inglês *The Free Market System*, é uma abordagem baseada em regras de mercado
- **Murdoch**: cada atribuição de tarefa é tratada como um processo de leilão, em que o robô vencedor é aquele que oferece o maior lance (GERKEY; MATARIC, 2002);

2.3 ROS - Robot Operating System

Acrônimo para *Robot Operating System* (QUIGLEY et al., 2009), o ROS é um *framework* para robótica que tem incentivado a comunidade de pesquisadores desta área do conhecimento a trabalhar conjuntamente desde seu lançamento. Ao observar o grande

avanço desta ferramenta de comunicação, muitos fabricantes de manipuladores industriais iniciaram a investir em pesquisas para integrar seus robôs com o ROS.

Uma lacuna que antes existia na nova geração de aplicações robóticas foi preenchida com o lançamento do ROS. Como um fornecedor de serviços de *middleware*, ele (1) simplifica o desenvolvimento de processos, (2) suporta comunicação e interoperabilidade, (3) oferece e facilita serviços frequentemente utilizados em robótica e, ainda, oferece (4) utilização eficiente dos seus recursos disponíveis, (5) abstrações heterogênicas e (6) descoberta e configuração automática de recursos (QUIGLEY et al., 2009). No intuito de cobrir todas exigências de um *middleware*, ROS 2.0 tenta dar suporte à sistemas embarcados e dispositivos de baixo recurso.

No ROS, projetos atômicos são chamados *pacotes* e podem ser desenvolvido em diversas linguagens de programação. Isso mostra que o ROS é flexível, pois seus usuários podem tirar proveito das vantagens que cada linguagem suportada tem, sejam elas eficiência em tempo de execução, confiabilidade, recursos, síntaxe, semântica, suporte ou documentação existente. Atualmente, as linguagens de programação suportadas são C++, Python e Lisp. As linguagens Java e Lua ainda estão em fase de desenvolvimento.

Projetos de robótica possuem rotinas que poderia ser reutilizadas em outros projetos. Por esta razão, ROS é também modular, pois pacotes configuráveis existentes podem ser combinados para realizar uma aplicação especifica de robótica. Várias bibliotecas externas já foram adaptadas para serem usadas no ROS: aruco[1], gmapping[2], interfaces de programação para aplicações de robôs[3], sensores[4] e simuladores[5], planejadores[6], reconhecimento de voz[7], entre outros. Isso evidencia que os usuários de ROS podem focar no desenvolvimento de pesquisa de sua área e contribuir da melhor forma com essa comunidade.

Enfim, ROS disponibiliza diversas ferramentas para auxiliar no desenvolvimento de projetos e, também, verificar o funcionamento de aplicação. Suas ferramentas típicas são: *get* e *set* de parâmetros de configuração, vizualização da topologia de conexão *peer-to-peer*, medição de utilização de banda, gráficos dos dados de mensagem e outras mais. É altamente recomendado o uso dessas ferramentas para garantir a estabilidade e confiança dos pacotes desenvolvidos, que normalmente têm alta complexidade.

Esta seção apresenta conceitos básicos para entender o funcionamento desta *framework*. Em seguida, são expostas as regras de nomenclatura dos recursos do ROS. E,

[1] <http://wiki.ros.org/ar_sys>
[2] <http://wiki.ros.org/gmapping>
[3] <http://wiki.ros.org/Robots>
[4] <http://wiki.ros.org/Sensors>
[5] <http://wiki.ros.org/gazebo>
[6] <http://kcl-planning.github.io/ROSPlan/>
[7] <http://wiki.ros.org/Sensors#Audio_.2BAC8_Speech_Recognition>

então, é brevemente dado suporte sobre a construção de aplicações gráficas integradas com o ROS.

2.3.1 Conceitos Básicos

Sua concepção foi fundada sobre conceitos divididos em três níveis: (1) Sistema de Arquivos do ROS, (2) Grafo de Computação do ROS e (3) Comunidade do ROS. A seguir será explicado cada um desses níveis, cada um com seu respectivo conjunto de conceitos. Além disso, também serão detalhados os dois tipos de nomes definidos no ROS: nomes de recursos de pacote e nomes de recursos de grafo.

2.3.1.1 Sistema de Arquivos do ROS

Os conceitos envolvidos no nível do *Sistema de Arquivos do ROS* se referem aos arquivos armazenados em disco. São eles:

- **Pacotes**: em inglês *Packages*, é uma forma atômica de organização de criação e lançamento de *software* no ROS. Um pacote contém definições de processos (nós), de dependência de bibliotecas, de tipos de mensagens, ações e serviços, de estruturas de dados e, por fim, de configuração.
- **Meta-Pacotes**: em inglês *Metapackages*, é um tipo especial de pacote que tem por objetivo agrupar pacotes relacionados.
- **Manifestos de Pacote**: em inglês *Package Manifests*, arquivo nomeado *package.xml* contido na raíz de cada pacote. Seu papel é fornecer meta-informações sobre seu pacote: nome, versão, descrição, informações de licença, dependências, entre outras.
- **Tipos de Mensagem**: em inglês *Message Types*, arquivos de extensão *.msg*, localizados dentro da pasta *msg* de um dado pacote. Seu conteúdo define a estrutura de dados de uma mensagem que poderá ser enviado pelo ROS.
- **Tipos de Serviço**: em inglês *Service Types*, arquivos de extensão *.srv*, localizados dentro da pasta *srv* de um dado pacote. Seu conteúdo define a estrutura de dados das mensagens de requisito e resposta de um serviço, as quais poderão ser enviadas pelo ROS.

2.3.1.2 Grafo de Computação do ROS

O *Grafo de Computação do ROS* é uma rede ponto-a-ponto de processos que processam dados conjuntamente. Os conceitos presentes neste nível são:

- **Nós**: em inglês *Nodes*, são processos computacionais que são executados para desempenhar o controle de atuadores, realizar leitura e filtragem de sinais sensoriais ou implementar algoritmos avançados de planejamento e tomada de decisão. É desejável que os nós sejam desenvolvidos da forma mais genérica possível, para sua reutilização em outros projetos. Cada linguagem de programação suportada encapsula as funcionalidades do ROS em uma biblioteca. Para a escrita de um nó na linguagem C++, é utilizada a biblioteca do pacote *roscpp*[8] e, para escrever um nó em Python, é utilizada a biblioteca contida no pacote *rospy*[9];

- **Nó Mestre**: em inglês *Master*, fornece cadastro e pesquisa de nome no Grafo de Computação do ROS, ou seja, este nó é responsável por garantir a comunicação entre os nós. Sem a sua execução, não existe comunicação entre os nós.

- **Servidor de Parâmetros**: em inglês *Parameter Server*, parte do Nó Mestre que centraliza a consulta e o armazenamento de dados indexados por uma cadeia de caracteres.

- **Mensagens**: em inglês *Messages*, a comunicação entre os nós no ROS consiste no transporte de mensagens, as quais são estruturas de dados que possuem campos tipados. Os campos de uma mensagem podem ser do tipo primitivo (booleano, inteiro, ponto flutuante, caracter, enumerado, cadeia de caracteres), aninhar outras mensagens ou vetores desses tipos.

- **Tópicos**: em inglês *Topics*, são canais que ligam os nós para o transporte de mensagens utilizando a semântica de comunicação *publish/subscribe*. Assim, nós que enviam mensagens para o sistema, as publica no tópico e nós recebem as mensagens ao assinar o tópico. Cada tópico possui um tipo, o que lhe permite transportar apenas este tipo de mensagem. Como característica da sua semântica, vários nós podem publicar e se inscrever no mesmo tópico. E um nó pode publicar e se inscrever em vários tópicos;

- **Serviços**: em inglês *Services*, é um sistema de comunicação no ROS que obedece a semântica *request/reply*. Neste caso, um nó cliente solicita um serviço através de um pedido para um nó servidor que, por sua vez, retorna uma resposta ao nó cliente ao finalizar o serviço prestado.

- **Bolsas**: do inglês *Bags*, são arquivos de extensão *.bag* que contêm dados de mensagens do ROS.

A Figura 2 ilustra os tipos básicos de comunicação entre nós no ROS. Nessa figura, nós são representados por elipses, tópicos por retângulos, conexões entre nó e tópico por

[8] <http://wiki.ros.org/roscpp>
[9] <http://wiki.ros.org/rospy>

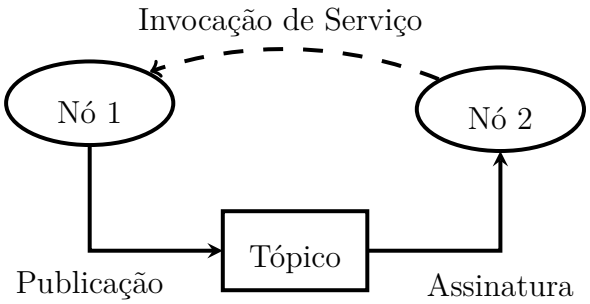


Figura 2 – Conceitos básicos de comunicação do ROS.

setas de linha contínua e invocações de serviço por setas de linha tracejada. Verifica-se assim que o *Nó 1* publica no *Tópico* e o *Nó 2* o subscreve. Além disso, o *Nó 1* é servidor do *Serviço* e o *Nó 2* é seu cliente.

Nós que publicam mensagens em um tópico só estão interessados em disponibilizar a informação, não importando com quem irá utilizá-lo. Da mesma forma, um nó que assina um tópico está apenas interessado em receber a informação disponível no tópica sem se importar com sua fonte. Deste modo, é aconselhado utilizar esse tipo de comunicação na troca de dados de fluxo contínuo, por exemplo, dados de sensores e sinais de atuação e controle.

Uma invocação de serviço é equivalente a chamada remota de um procedimento. Quando um cliente de serviço solicita um pedido ao seu servidor, ambos ficam aguardando o procedimento finalizar. Com isso, é recomendado o uso desse tipo de comunicação em casos onde o serviço prestado é rápido, como alterações do estado de alguma variável interna.

Vale salientar que muitas mensagens e serviços já foram padronizadas em pacotes do ROS[10].

2.3.1.3 Comunidade do ROS

De modo que comunidades separadas possam trocar código fonte e conhecimento, vários recursos foram criados na *Comunidade do ROS*. Tais como:

- **Distribuições**: agrupa coleções de pacotes versionados para facilitar a instalação do ROS. Além disso, é mantido uma versão consistente de cada conjunto de pacotes relacionados.
- **Repositórios**: uma rede federada de repositórios de código permite que instituições diferentes possam desenvolver e lançar componentes de *software* para seus próprios robôs.

[10] <http://wiki.ros.org/common_msgs>

- **ROS Wiki**[11]: é o principal fórum para informações de documentação sobre o ROS. Qualquer pessoa pode solicitar uma conta para contribuir com sua própria documentação, ou ainda fornecer correções e atualizações, bem como, escrever tutoriais.
- **Listas de endereços eletrônicos**: é o meio de comunicação primário entre os usuários de ROS para perguntar sobre questões de *software* do ROS e para receber notificações de novas atualizações.
- **ROS Answers**[12]: é uma página *web* de perguntas e respostas diretamente relacionada ao ROS.
- **Blog**[13]: providencia notícias regularmente com fotos e vídeos.

2.3.1.4 Nome de Recurso de Grafo

Os recursos de grafo presentes no ROS são: nós, parâmetros, tópicos e serviços. Com o uso adequado da sintaxe de nomes, é possível obter encapsulamento desses recursos através do mecanismo que os nomeia, pois ele gera uma estrutura hierárquica de nomes. Em outras palavras, cada recurso no ROS possui um *namespace* que pode ser compartilhado com vários outros recursos. Normalmente, recursos podem criar outros recursos dentro do seu próprio *namespace* e acessar recursos que estão dentro ou acima dele. Contudo, recursos em camadas inferiores podem ser acessados através da integração de código em *namespaces* superiores. Abaixo, seguem exemplos de nomes de recurso no ROS.

- /
- /rqt_mrta
- /lro/p3dx/pose
- /lro/amigobot/pose
- /lro/alliance

O primeiro exemplo mostra o *namespace* global (/). Todos os recursos com seu respectivo *namespace* estão sob ele. O exemplo seguinte mostra um recurso denominado *rqt_mrta* cujo *namespace* se encontra no nível mais alto. Em seguida, verifica-se três exemplos de recursos que estão sob o *namespace lro*. Entretanto, os recursos mostrados nos terceiro e quarto exemplos ainda estão sob um outro *namespace*, *p3dx* e *amigobot*, respectivamente. Note que neste caso, o nome de ambos recursos são iguais (*pose*), porém

[11] <http://wiki.ros.org>
[12] <https://answers.ros.org/questions/>
[13] <http://www.ros.org/news/>

eles são diferenciados pelos seus *namespaces* (*/lro/p3dx* e */lro/amigobot*, respectivamente). Por último, é dado o recurso cujo nome é *alliance* que se encontra sob o *namespace /lro*.

Existem quatro tipos de resolução de nomes de recursos no ROS: *base*, *relativa*, *global* e *privada*.

- base
- relativa/nome
- /global/nome
- ~privada/nome

Nomes são resolvidos relativamente, então recursos não necessitam estar cientes de qual *namespace* eles se encontram. Isso simplifica a programação como nós que trabalham em conjunto podem ser escritos como se eles estivessem todos no nível de *namespace* mais alto.

Tabela 2 – Exemplos de resolução de nomes no ROS

Nó	**Relativa**	**Global**	**Privada**
/no	img→/no/img	/img→/img	~img→/no/img
/no	img/raw→/no/img/raw	/img/raw→/img/raw	~img/raw→/no/img/raw
/ns/no	img→/ns/no/img	/img→/img	~img→/ns/no/img

Esses conceitos possuem extrema importância em sistemas multirrobô, principalmente naqueles cuja frota de robôs é homogênea. Neste último caso, a partir de replicação das configurações de um robô, todo o sistema pode ser iniciado, variando apenas o *namespace* de cada robô do sistema.

2.3.1.5 Nome de Recursos de Pacote

2.3.2 Interface Gráfica de Usuário do ROS

Além de ferramentas disponíveis em terminal via comando de linha, o ROS também disponibiliza ferramentas gráficas cujas funcionalidades são controladas por um *plugin*. Estes são desenvolvidos através do *rqt*[14] que disponibiliza uma interface de programação de aplicação (do inglês, *Application Programming Interface* - API) em C++ e Python para a criação de interface gráfica de usuário (GUI, acrônimo para *Graphical User Interface*) integrada com o ROS. Por sua vez, esta API utiliza o Qt (lê-se *cute*) como seu *kit* de

[14] <http://wiki.ros.org/rqt>

desenvolvimento de *software* (SDK - *Software Development Kit*). A Figura 3 foi extraida da página do metapacote *rqt* e mostra a aplicação de vários *plugins* que foram acoplados em uma mesma janela através do *rqt_gui*[15].

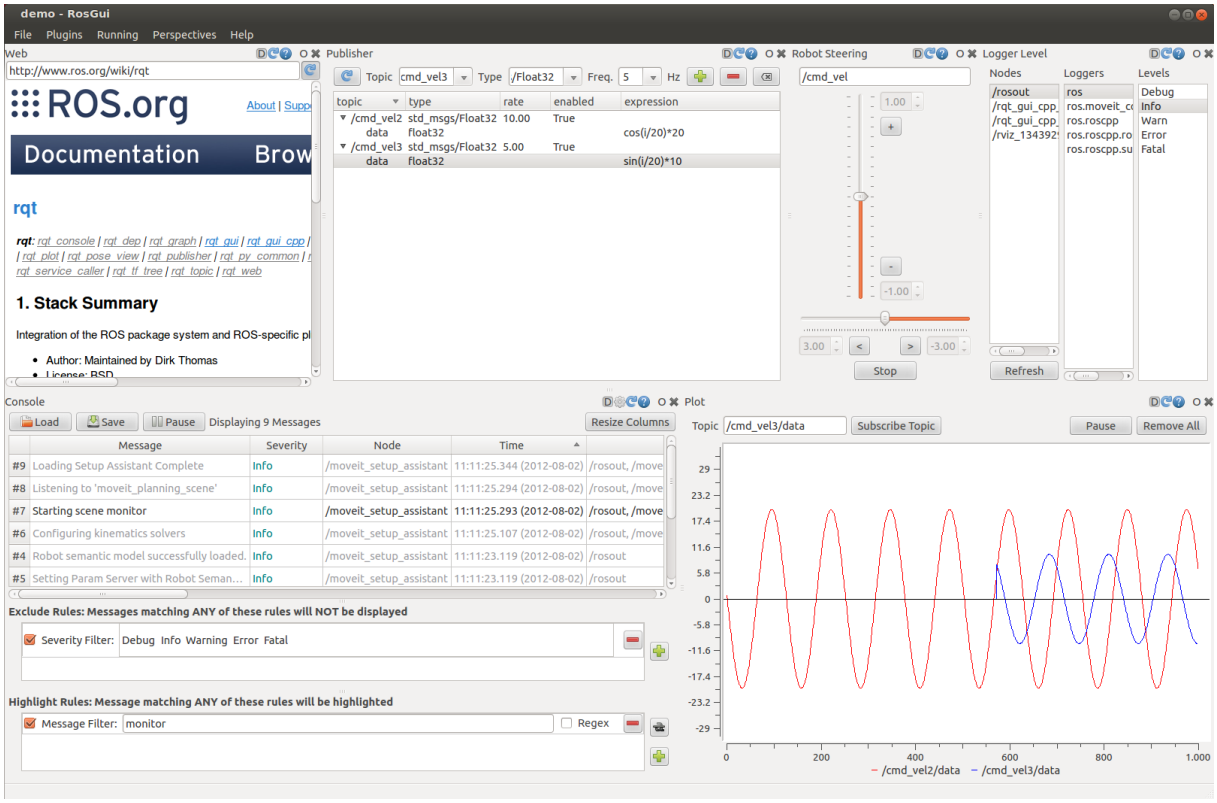


Figura 3 – Exemplo de ferramentas gráficas existentes no ROS.

Essas ferramentas são agrupadas em categorias. Entre elas estão:

- **Configuração**: reune ferramentas relacionadas a execução e configuração de nós, os *plugins* *rqt_launch*[16] e *rqt_reconfigure*[17] são exemplos disso;
- **Introspecção**: junta *plugins* para a análise do Grafo de Computação e das dependências entre pacotes;
- ***Logging***: agrupa ferramentas para alternar o nível de *log* nos nós e para filtrar *logs*;
- **Tópicos**: são reunidas ferramentas diretamente relacionadas tópicos no ROS, como a publicação de mensagens, monitor de tópico e navegador para definições de mensagem;
- **Serviços**: cliente de serviços e navegador para definições de serviços, são exemplo de ferramentas relacionadas com serviços;

[15] <http://wiki.ros.org/rqt_gui>
[16] <http://wiki.ros.org/rqt_launch>
[17] <http://wiki.ros.org/rqt_reconfigure>

- **Visualização**: agrupa ferramentas que traçam gráficos de dados numéricos no tempo, mostram de imagens publicadas em tópicos e sistemas supervisórios, são exemplos de *plugins* que pertencem a esta categoria *rqt_image_view*[18], *rqt_multiplot*[19] e *rqt_rviz*[20];

- e muitas outras.

2.4 Trabalhos Relacionados

Reis e Bastos (2015) elaborou uma aproximação da arquitetura Alliance dedicada para uma aplicação de patrulhamento

[18] <http://wiki.ros.org/rqt_image_view>
[19] <http://wiki.ros.org/rqt_multiplot>
[20] <http://wiki.ros.org/rqt_rviz>

3 Desenvolvimento

3.1 *rqt_mrta*

Esta interface gráfica tem por finalidade garantir que usuários de arquitetura MRTA configurem corretamente arquiteturas d

Este pacote

Sabendo que *rqt*[1] é um *framework* que auxilia desenvolvedores de interface gráfica em Qt (YAFEI, 2012) para usuários (GUI) do ROS.

Este pacote ROS fornece um *plugin* de interface gráfica de usuário (GUI) escrito em C++ para o *framework rqt*[2] baseada em Qt(YAFEI, 2012). escrito na linguagem de programação C++, desenvolvido a partir da *framework*

Para isso, existem dois arquivos de configuração, um para a configuração da arquitetura e outro para a configuração do problema. Ambos arquivos possuem a extensão XML (Extensible Markup Language). Ao serem carregados pela aplicação, esta se adapta para tratar o problema MRTA definido, utilizando a arquitetura escolhida.

3.1.1 Arquivo de configuração de arquitetura

Seu propósito é:

- classificar a arquitetura segundo a taxonomia sugerida por Gerkey e Matarić (2004);
- identificar os parâmetros necessários para sua configuração inicial;
- identificar os nós que
- associar os arquivos de parâmetro

auxiliar os usuários de arquitetura de alocação de tarefa em sistema multirrobô.

3.1.2 Arquivo de configuração de aplicação

3.1.3 Modelo

MVC (*Model-View-Control*

[1] <http://wiki.ros.org/rqt>
[2] <http://wiki.ros.org/rqt>

3.2 ALLIANCE

Esta é uma arquitetura totalmente distribuída, tolerante à falhas, que visa atingir controle cooperativo e atender os requisitos de uma missão à ser desempenhada por um grupo de robôs heterogêneos (PARKER, 1998). Cada robô é modelado usando uma aproximação baseada em comportamentos. A partir do estado do ambiente e dos outros robôs cooperadores, uma configuração de comportamento é selecionada conforme sua respectiva função de realização de tarefa na camada de alto nível de abstração. Cada configuração de comportamento permite controlar os atuadores do robô em questão de um modo diferente.

Sejam $R = \{r_1, r_2, \cdots, r_n\}$, o conjunto de n robôs heterogêneos, e $A = \{a_1, a_2, \cdots, a_m\}$, o conjunto de m sub-tarefas independentes que compõem uma dada missão. Na arquitetura ALLIANCE, cada robô r_i possui um conjunto de p configurações de comportamento, dado por $C_i = \{c_{i1}, c_{i2}, \cdots, c_{ip}\}$. Cada configuração de comportamento fornece ao seu robô uma função de realização de tarefa em alto nível, conforme definido em (BROOKS, 1986). Por fim, é possível saber qual tarefa em A é executada por r_i quando sua configuração de ativação c_{ik} é ativa. Tal informação é obtida através da função $h_i(c_{ik})$, a qual pertence ao conjunto de n funções $H : C_i \rightarrow A$, $H = \{h_1(c_{1k}), h_2(c_{2k}), \cdots, h_n(c_{nk})\}$.

A ativação de uma dada configuração de comportamento c_{ij} do robô r_i para a execução da tarefa $h_i(c_{ij})$ em um dado instante, é dada pelo cálculo de motivação do seu comportamento motivacional. Por sua vez, cada comportamento motivacional possui um conjunto de módulos que têm a responsabilidade de monitorar alguma informação relevante sobre o sistema. A seguir, será detalhado o papel de cada uma desses módulos e suas contribuições para o cálculo de motivação.

A primeira função, definida pela Equação 3.1, tem como responsabilidade identificar quando a configuração de comportamento c_{ij} é aplicável. Esta função lógica é implementada no módulo de *feedback* sensorial, o qual observa constantemente as condições do ambiente por meio de sensores e, então, verifica se o sistema é favorecido se c_{ij} estiver ativada.

$$aplicável_{ij}(t) = \begin{cases} 1, & \text{se o módulo de } \textit{feedback} \text{ sensorial da configuração de comportamento } c_{ij} \text{ do robô } r_i \text{ indicar que esta configuração é aplicável mediante ao estado atual do ambiente no instante } t; \\ 0, & \text{caso contrário.} \end{cases} \tag{3.1}$$

A Equação 3.2 mostra uma das funções lógicas que também compõe o cálculo para ativação de c_{ij}. Seu papel, neste cálculo, é garantir que o robô r_i só tenha uma configuração de comportamento ativa por vez. Essa função é implementada pelo módulo

de supressão, o qual observa a ativação das demais configurações de comportamento de r_i.

$$inibida_{ij}(t) = \begin{cases} 1, & \text{se outra configuração de comportamento } c_{ik} \\ & \text{(com } k \neq j\text{) está ativa no robô } r_i \text{ no instante} \\ & t; \\ 0, & \text{caso contrário.} \end{cases} \tag{3.2}$$

Cada configuração de comportamento c_{ij} possui um módulo de comunicação que auxilia vários outros módulos de c_{ij} por meio do monitoramento da comunicação entre os robôs do sistema. Este módulo mantém o histórico das atividades dos demais robôs do sistema no que diz respeito à execução da tarefa $h_i(c_{ij})$. Deste modo, os demais módulos de c_{ij} podem consultar se os outros robôs estavam executando a tarefa $h_i(c_{ij})$ em um dado intervalo de tempo $[t_1; t_2]$, conforme mostra a Equação 3.3. Existem dois parâmetros no ALLIANCE que influenciam diretamente no módulo de comunicação de cada comportamento motivacional. O primeiro parâmetro, ρ_i, define a frequência com que r_i atualiza suas configurações de comportamento e publica seu estado atual, no que diz respeito à arquitetura. O segundo parâmetro, τ_i, indica a duração de tempo máxima que o robô r_i permite ficar sem receber mensagens do estado de qualquer outro robô do sistema. Quando esta duração é excedida para um dado robô r_k, o robô r_i passa considerar que r_k cessou sua atividade. A utilização deste parâmetro visa prever falhas de comunicação e de mal funcionamento.

$$recebida_{ij}(k, t_1, t_2) = \begin{cases} 1, & \text{se o robô } r_i \text{ recebeu mensagem do robô } r_k \\ & \text{referente à tarefa } h_i(c_{ij}) \text{ dentro do intervalo} \\ & \text{de tempo } [t_1; t_2], \text{ em que } t_1 < t_2; \\ 0, & \text{caso contrário.} \end{cases} \tag{3.3}$$

A próxima função tem a incumbência de reiniciar o cálculo para a ativação da configuração de comportamento c_{ij}. Essa função lógica é impulsionada apenas uma vez para cada robô que tenta executar a tarefa $h_i(c_{ij})$. Isto é, no instante em que acontece a primeira rampa de subida na Equação 3.3 para cada robô r_k, está função retorna um nível lógico alto. Essa condição evita que problemas de falhas persistentes não comprometam a completude da missão.

$$reiniciada_{ij}(t) = \exists x, (recebida_{ij}(x, t - dt, t) \wedge \neg recebida_{ij}(x, 0, t - dt)) \tag{3.4}$$

onde dt é o tempo decorrido desde a última verificação de comunicação.

A Equação 3.5 auxilia o módulo de aquiescência no cálculo de desistência para a desativação de c_{ij}. Baseando-se no histórico de ativação de c_{ij}, o módulo de comporta-

mento motivacional disponibiliza essa função lógica que verifica se c_{ij} ficou mantida ativa por um dado período de tempo até o instante desejado.

$$ativa_{ij}(\Delta t, t) = \begin{cases} 1, & \text{se a configuração de comportamento } c_{ij} \text{ do} \\ & \text{robô } r_i \text{ estiver ativa por mais de } \Delta t \text{ unidades} \\ & \text{de tempo no instante } t; \\ 0, & \text{caso contrário.} \end{cases} \tag{3.5}$$

O módulo de aquiescência monitora o tempo decorrido após a ativação da configuração de comportamento c_{ij} do robô r_i com o auxílio da Equação 3.5. São duas as suas preocupações: (1) verificar se c_{ij} permaneceu ativa por mais tempo que o esperado e (2) verificar se o tempo decorrido após um outro robô r_k ter iniciado a execução da tarefa $h_i(c_{ij})$, enquanto c_{ij} estava ativa, tenha excedido o tempo configurado para r_i passar sua vez para esse outro robô. A Equação 3.6 define as condições em que r_i está aquiescente à desativação de c_{ij}.

$$\begin{aligned} aquiescente_{ij}(t) = (&ativa_{ij}(\psi_{ij}(t), t) \wedge \exists x, recebida_{ij}(x, t - \tau_i, t)) \\ &\vee ativa_{ij}(\lambda_{ij}(t), t) \end{aligned} \tag{3.6}$$

onde $\psi_{ij}(t)$ é a duração de tempo que r_i deseja manter a configuração de comportamento c_{ij} ativa antes de dar preferência para outro robô executar a tarefa $h_i(c_{ij})$; e $\lambda_{ij}(t)$ é a duração de tempo que r_i deseja manter c_{ij} ativa antes de desistir para possivelmente tentar outra configuração de comportamento.

A impaciência de r_i para a ativação de c_{ij} cresce linearmente mediante a taxa de impaciência instantânea. Assim, o módulo de impaciência de c_{ij} é responsável por identificar falhas de execução da tarefa $h_i(c_{ij})$ por outros robôs do sistema e quantificar a insatisfação de r_i concernente à essa tarefa, conforme visto na Equação 3.7. Para isso, três parâmetros são utilizados: (1) $\phi_{ij}(k, t)$, o qual estabelece o tempo máximo que r_i permite a um outro robô r_k executar a tarefa $h_i(c_{ij})$ antes dele próprio iniciar sua tentativa; (2) $\delta_{slow_{ij}}(k, t)$, que determina a taxa de impaciência do robô r_i com respeito à configuração de comportamento c_{ij} enquanto o robô r_k está executando a tarefa correspondente à c_{ij}; e (3) $\delta_{fast_{ij}}(t)$, que determina a taxa de impaciência de r_i com relação à c_{ij} quando nenhum outro robô está executando a tarefa $h_i(c_{ij})$.

$$impaciência_{ij}(t) = \begin{cases} \min\limits_{x} \delta_{slow_{ij}}(x, t), & \text{se } recebida_{ij}(x, t-\tau_i, t) \wedge \neg recebida_{ij}(x, 0, t- \\ & \phi_{ij}(x, t); \\ \delta_{fast_{ij}}(t), & \text{caso contrário.} \end{cases} \tag{3.7}$$

Note que o método usado incrementa a motivação à uma taxa que permita que o robô mais lento r_k continue sua tentativa de execução de $h(c_{ij})$, desde que seja respeitada a duração máxima estipulada pelo parâmetro $\phi_{ij}(k, t)$.

A Equação 3.8 mostra a função de motivação, a qual combina todas as funções mencionadas anteriormente para a ativação da configuração de comportamento c_{ij}. Seu valor inicial é nulo e aumenta mediante a taxa de impaciência instantânea de r_i para ativar c_{ij} quando satisfeitas as seguintes condições: (1) c_{ij} seja aplicável, (2) mas não tenha sido inibida, (3) nem reiniciada; (4) e, ainda, r_i não seja aquiescente em desistir de manter c_{ij} ativa. Quando uma das condições citadas não é satisfeita, seu valor volta a ser nulo.

$$\begin{aligned} motivação_{ij}(0) &= 0 \\ motivação_{ij}(t) &= (motivação_{ij}(t-dt) + impaciência_{ij}(t)) \\ &\quad \times aplicável_{ij}(t) \times inibida_{ij}(t) \\ &\quad \times reiniciada_{ij}(t) \times aquiescente_{ij}(t). \end{aligned} \tag{3.8}$$

Assim que a motivação de r_i para ativar c_{ij} ultrapassa o limite de ativação, essa configuração de comportamento é ativada, conforme a Equação 3.9:

$$ativa_{ij}(t) = motivação_{ij}(t) \geq \theta \tag{3.9}$$

onde θ é o limite de ativação.

Fazendo uma análise das equações acima, verifica-se que, enquanto sua motivação cresce, é possível estimar quanto tempo resta para que a configuração de comportamento c_{ij} se torne ativa.

$$\overline{\Delta t}_{ativação_{ij}} = \frac{\theta - motivação_{ij}(t)}{impaciência_{ij}(t)\rho_i} \tag{3.10}$$

onde ρ_i é a frequência aproximada, em [Hz], com que r_i atualiza as motivação das configurações de comportamento em C_i e, ainda, publica seu estado comportamental. Como a taxa de impaciência não é constante, a Equação 3.10 é apenas uma estimativa, dada em [s].

Em conformidade com o que foi exposto, pode-se observar que é possível normalizar todas as funções de motivação, de modo que a imagem de cada uma delas pertença ao intervalo $[0;1] \subset \mathbb{R}_+$. Para isso, é necessário: (1) parametrizar o módulo de impaciência de cada configuração de comportamento c_{ij}, de maneira que a imagem da sua função de taxa de impaciência instantânea pertença ao intervalo $(0;1) \subset \mathbb{R}_+^*$; além disso, (2) atribuir o valor unitário ao limite de ativação; bem como, (3) saturar a função de motivação no limite de ativação. Como resultado, as Equações 3.9 e 3.10 podem ser rescritas como as Equações 3.11 e 3.12, respectivamente.

$$ativa_{ij}(t) = motivação_{ij}(t) == 1 \tag{3.11}$$

$$\overline{\Delta t}_{ativação_{ij}} = \frac{1 - motivação_{ij}(t)}{impaciência_{ij}(t)\rho_i} \tag{3.12}$$

Parker (1996) desenvolveu também uma variação do ALLIANCE, chamada L-ALLIANCE, capaz de estimar alguns parâmetros do ALLIANCE durante a fase de aprendizado.

Classificar essa arquitetura segundo as taxonomias revisadas

Comentar sobre o motivo de ter falado sobre essa arquitetura com tanto detalhe

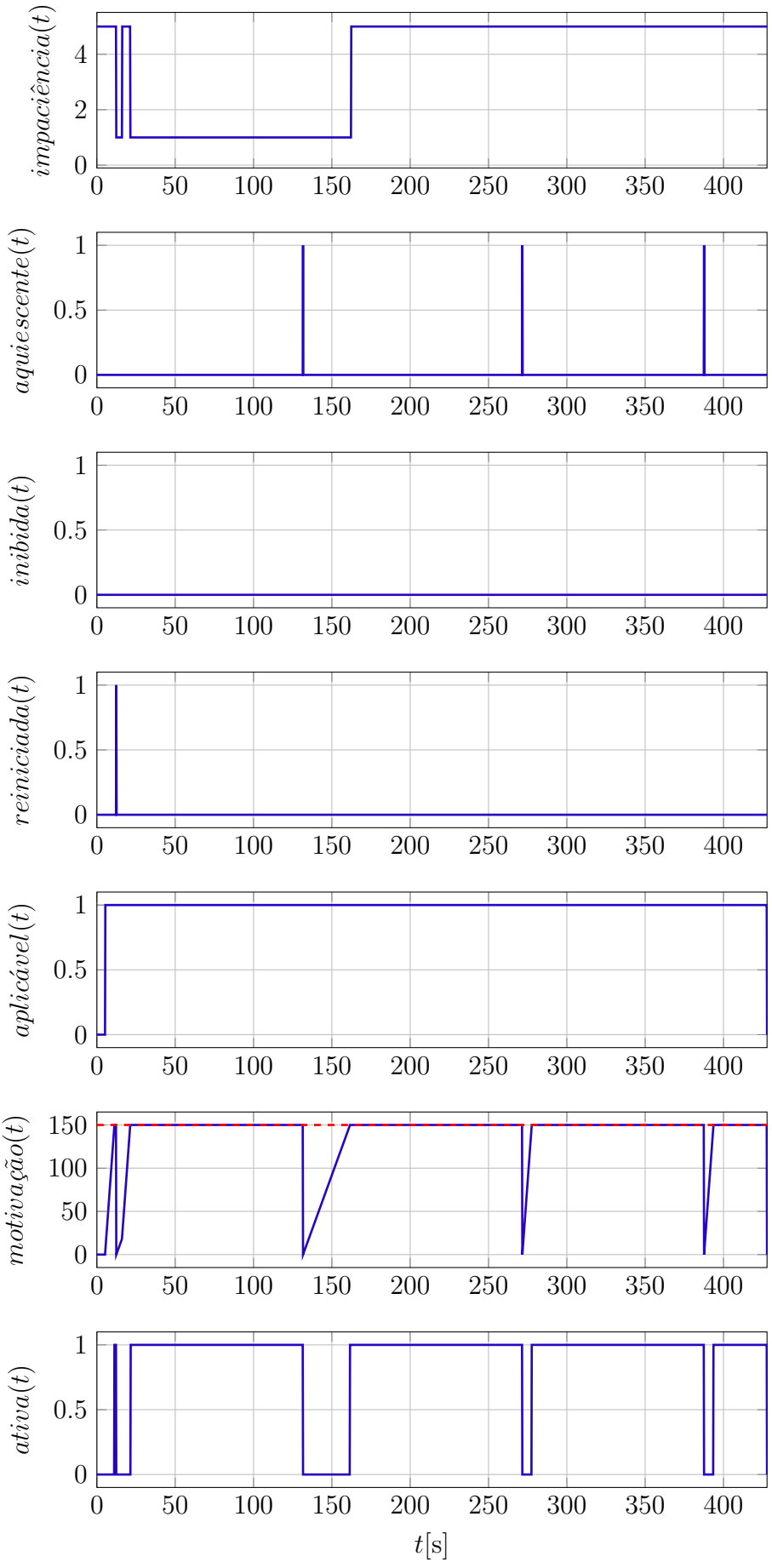


Figura 4 – Motivação da configuração de comportamento /robot1/wander.

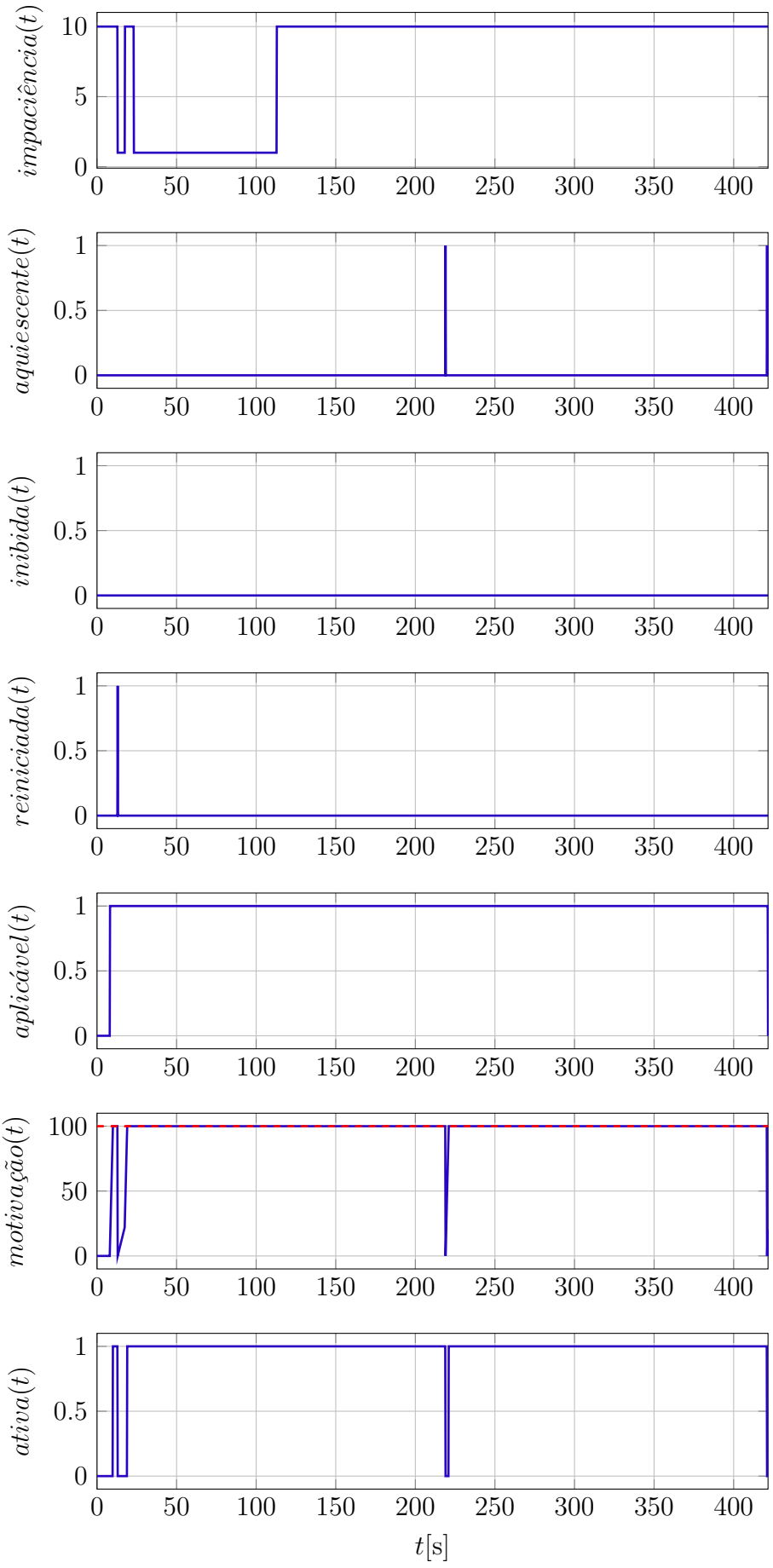


Figura 5 – Motivação da configuração de comportamento /robot2/wander.

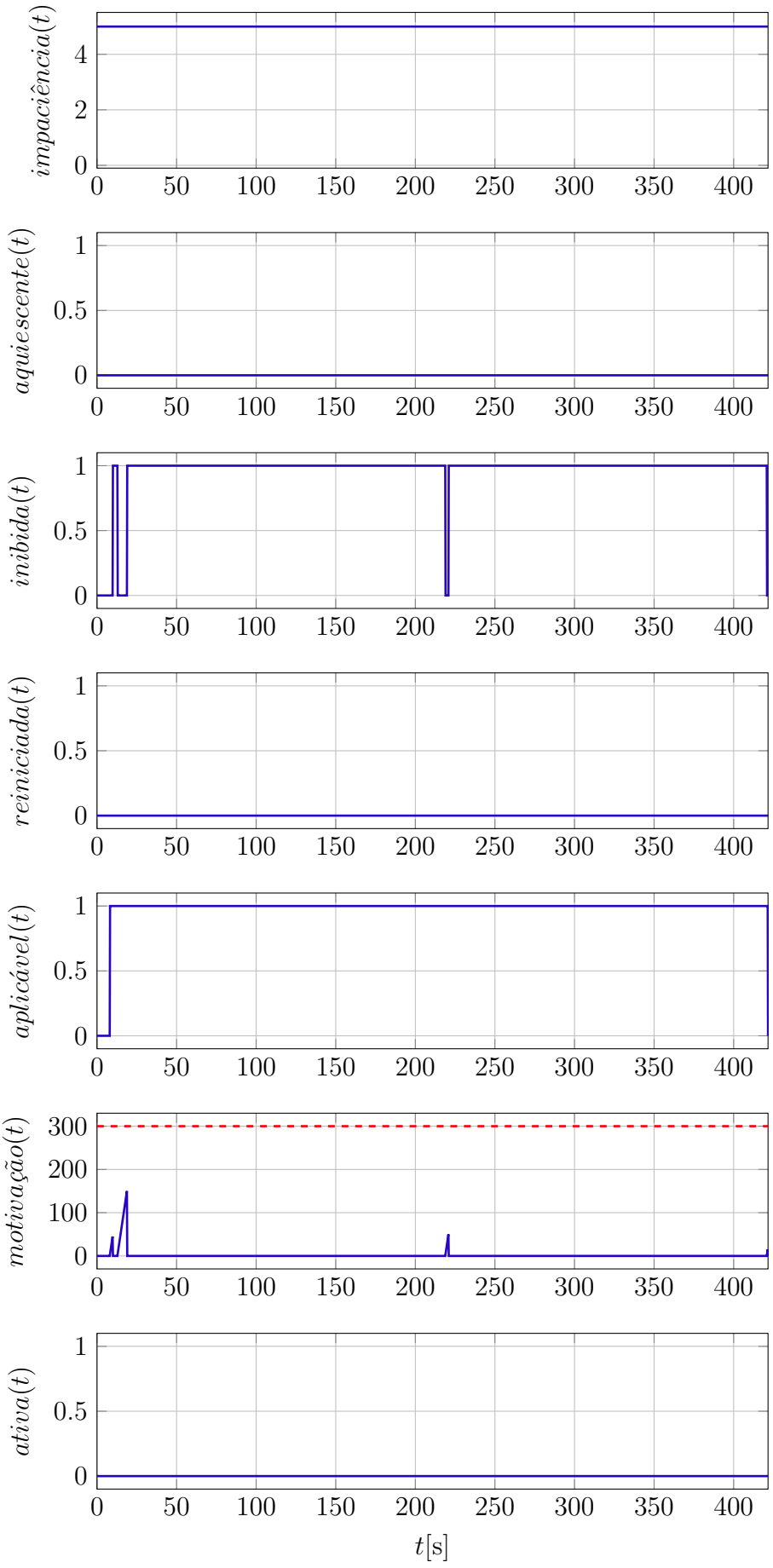


Figura 6 – Motivação da configuração de comportamento /robot2/border-protection.

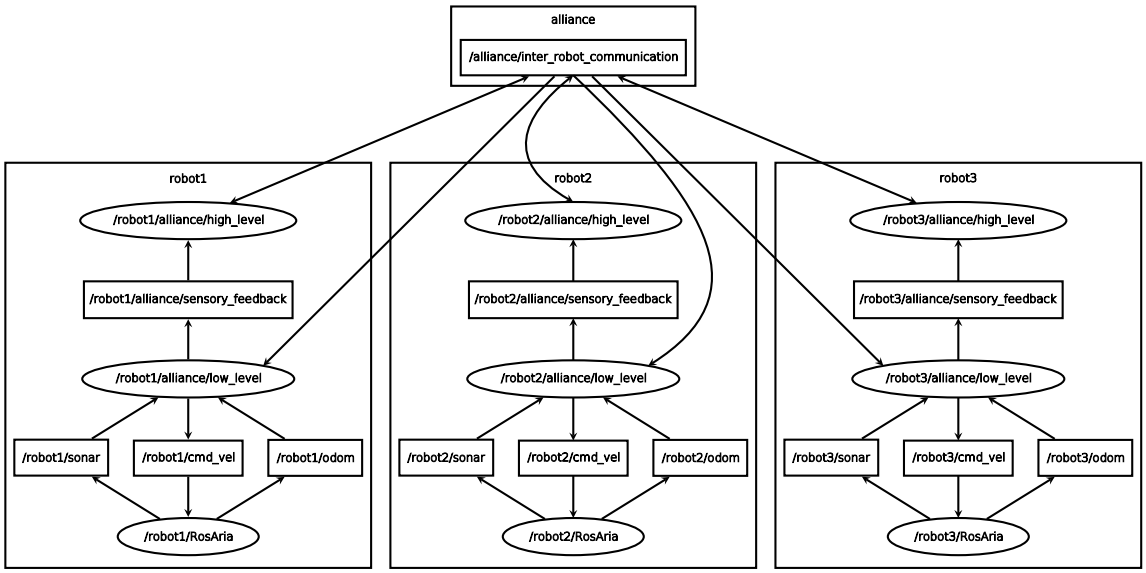


Figura 7 – Grafo do ALLIANCE no ROS para três robôs.

4 Experimentos e Resultados

5 Conclusão e Trabalhos Futuros

5.1 Conclusão

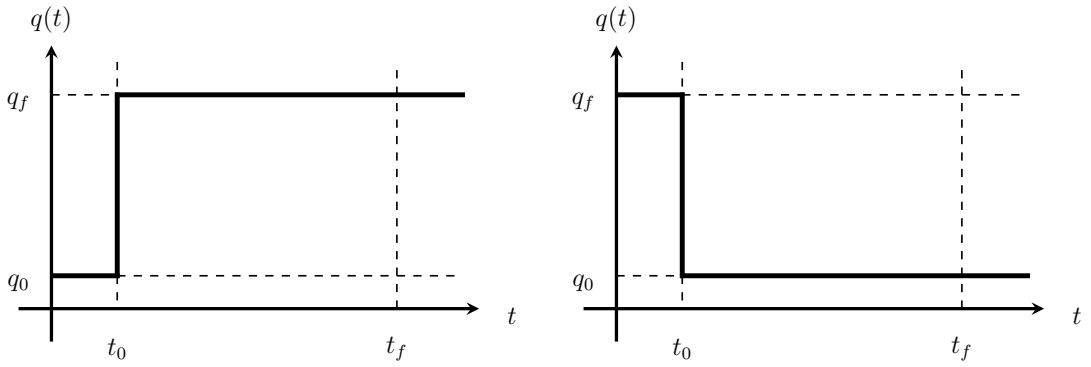
5.2 Trabalhos Futuros

Apêndices

APÊNDICE A – Funções Temporais

A.1 Funções Degraus

A Figura 8 mostra duas funções degraus: ascendente 8a e descendente 8b.



(a) Função degrau ascendente.

(b) Função degrau descendente.

Figura 8 – Funções degraus.

$$q(t) = \begin{cases} q_0, & t \leq t_0 \\ q_f, & t > t_0 \end{cases} \tag{A.1}$$

$$q(t) = \begin{cases} q_f, & t \leq t_0 \\ q_0, & t > t_0 \end{cases} \tag{A.2}$$

A.2 Funções Pulsos

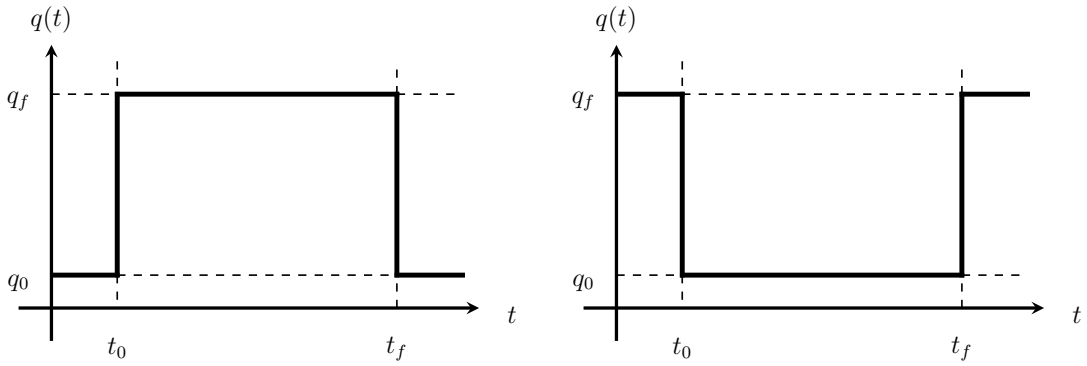
A Figura 9 mostra duas funções pulsos: ascendente 9a e descendente 9b.

$$q(t) = \begin{cases} q_0, & t \leq t_0 \\ q_f, & t > t_0 \end{cases} \tag{A.3}$$

$$f(t) = \begin{cases} q_f, & t \leq t_0 \\ q_0, & t > t_0 \end{cases} \tag{A.4}$$

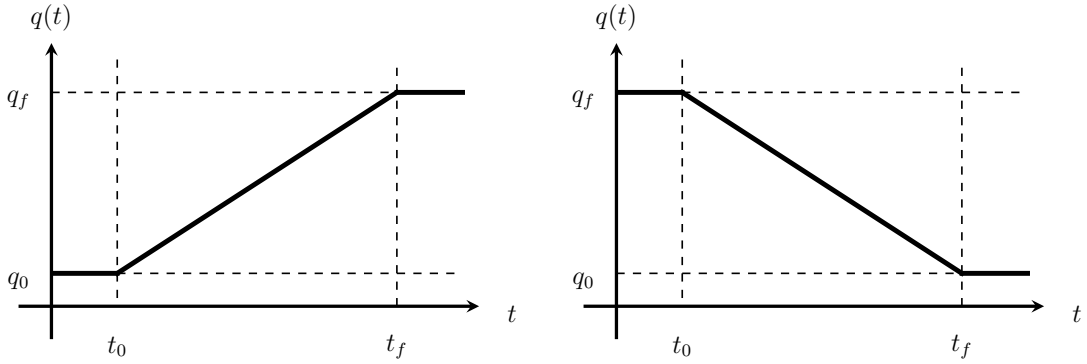
A.3 Funções Lineares

A Figura 10 mostra duas funções lineares: ascendente 10a e descendente 10b.



(a) Função pulso ascendente. (b) Função degrau descendente.

Figura 9 – Funções pulsos.



(a) Função linear ascendente. (b) Função linear descendente.

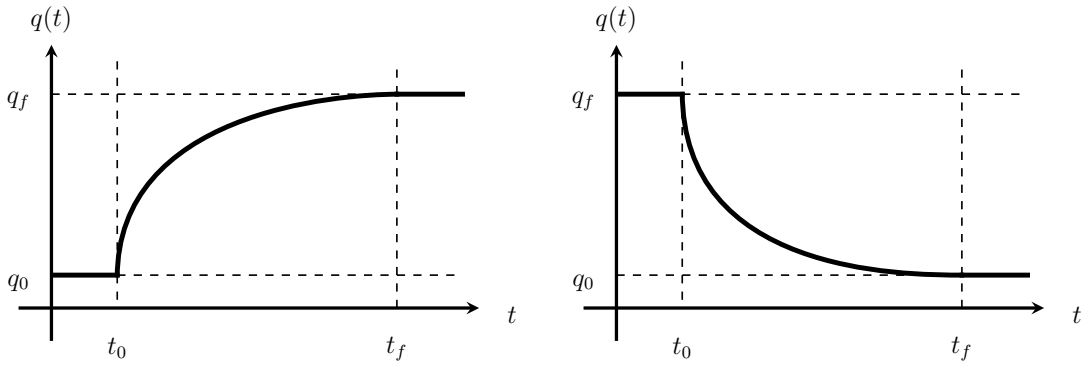
Figura 10 – Funções lineares.

$$q(t) = \begin{cases} q_0, & t \leq t_0 \\ (q_f - q_0)\dfrac{t - t_0}{t_f - t_0} + q_0, & t_0 < t \leq t_f \\ q_f, & t > t_f \end{cases} \tag{A.5}$$

$$q(t) = \begin{cases} q_f & t \leq t_0 \\ (q_0 - q_f)\dfrac{t - t_0}{t_f - t_0} + q_f, & t_0 < t \leq t_f \\ q_0 & t > t_f \end{cases} \tag{A.6}$$

A.4 Função Exponenciais

A Figura 11 mostra duas funções exponenciais: ascendente 11a e descendente 11b.



(a) Função exponencial ascendente. (b) Função exponencial descendente.

Figura 11 – Funções exponenciais.

$$q(t) = \begin{cases} q_0 & t \leq t_0 \\ q_f - (q_f - q_0)e^{-K\dfrac{t - t_0}{t_f - t_0}}, & t_0 < t \leq t_f \\ q_f & t > t_f \end{cases} \tag{A.7}$$

$$q(t) = \begin{cases} q_f & t \leq t_0 \\ q_0 - (q_0 - q_f)e^{-K\dfrac{t - t_0}{t_f - t_0}}, & t_0 < t \leq t_f \\ q_0 & t > t_f \end{cases} \tag{A.8}$$

Anexos

ANEXO A – Artigo publicado

Sed mattis, erat sit amet gravida malesuada, elit augue egestas diam, tempus scelerisque nunc nisl vitae libero. Sed consequat feugiat massa. Nunc porta, eros in eleifend varius, erat leo rutrum dui, non convallis lectus orci ut nibh. Sed lorem massa, nonummy quis, egestas id, condimentum at, nisl. Maecenas at nibh. Aliquam et augue at nunc pellentesque ullamcorper. Duis nisl nibh, laoreet suscipit, convallis ut, rutrum id, enim. Phasellus odio. Nulla nulla elit, molestie non, scelerisque at, vestibulum eu, nulla. Ut odio nisl, facilisis id, mollis et, scelerisque nec, enim. Aenean sem leo, pellentesque sit amet, scelerisque sit amet, vehicula pellentesque, sapien.

Referências

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. 20

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. 16, 22

BROOKS, R. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, IEEE, v. 2, n. 1, p. 14–23, 1986. 32

CAO, Y. U.; FUKUNAGA, A. S.; KAHNG, A. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, Kluwer Academic Publishers, v. 4, n. 1, p. 7–27, 1997. 16

CHAIMOWICZ, L.; CAMPOS, M. F.; KUMAR, V. Dynamic role assignment for cooperative robots. In: IEEE. *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. [S.l.], 2002. v. 1, p. 293–298. 16

DUDEK, G. et al. A taxonomy for multi-agent robotics. *Autonomous Robots*, Springer, v. 3, n. 4, p. 375–397, 1996. 16

FOX, M.; LONG, D. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 2003. 16

FRANK, A. On kuhn's hungarian method—a tribute from hungary. *Naval Research Logistics (NRL)*, Wiley Online Library, v. 52, n. 1, p. 2–5, 2005. 16

GERKEY, B. P.; MATARIC, M. J. Sold!: Auction methods for multirobot coordination. *IEEE transactions on robotics and automation*, IEEE, v. 18, n. 5, p. 758–768, 2002. 16, 21, 22

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. 16, 20, 21, 31

JULIO, R. E.; BASTOS, G. S. Dynamic bandwidth management library for multi-robot systems. In: IEEE. *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. [S.l.], 2015. p. 2585–2590. 16

LI, A. W.; BASTOS, G. S. A hybrid self-adaptive particle filter through kld-sampling and samcl. In: IEEE. *Advanced Robotics (ICAR), 2017 18th International Conference on*. [S.l.], 2017. p. 106–111. 16

MANNE, A. S. On the job-shop scheduling problem. *Operations Research*, INFORMS, v. 8, n. 2, p. 219–223, 1960. 16

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. 21, 36

PARKER, L. E. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, IEEE, v. 14, n. 2, p. 220–240, 1998. 16, 17, 21, 32

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, p. 5. 16, 22, 23

REIS, W. P. N. dos; BASTOS, G. S. Multi-robot task allocation approach using ros. In: IEEE. *Robotics Symposium (LARS) and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR), 2015 12th Latin American*. [S.l.], 2015. p. 163–168. 17, 30

SCHNEIDER, D. G. et al. Robot navigation by gesture recognition with ros and kinect. In: IEEE. *Robotics Symposium (LARS) and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR), 2015 12th Latin American*. [S.l.], 2015. p. 145–150. 16

SMITH, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, IEEE, n. 12, p. 1104–1113, 1980. 21

STENTZ, A.; DIAS, M. B. *A free market architecture for coordinating multiple robots*. [S.l.], 1999. 16

WATKINS, C. J.; DAYAN, P. Q-learning. *Machine learning*, Springer, v. 8, n. 3-4, p. 279–292, 1992. 16

WERGER, B. B.; MATARIĆ, M. J. Broadcast of local eligibility for multi-target observation. In: *Distributed autonomous robotic systems 4*. [S.l.]: Springer, 2000. p. 347–356. 16

YAFEI, H. *Qt creator quick start*. [S.l.]: Beijing: Beihang University Press, 2012. 31

YAN, Z.; JOUANDEAU, N.; CHERIF, A. A. Multi-robot decentralized exploration using a trade-based approach. In: *ICINCO (2)*. [S.l.: s.n.], 2011. p. 99–105. 21

YAN, Z.; JOUANDEAU, N.; CHERIF, A. A. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, SAGE Publications Sage UK: London, England, v. 10, n. 12, p. 399, 2013. 21, 22

ZLOT, R. et al. Multi-robot exploration controlled by a market economy. In: IEEE. *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. [S.l.], 2002. v. 3, p. 3016–3023. 16

ZLOT, R. M.; STENTZ, A. An auction-based approach to complex task allocation for multirobot teams. Carnegie Mellon University, The Robotics Institute, 2006. 19, 21