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

rqt_mrta: Uma Interface Gráfica de Usuário baseada em ROS para configuração e supervisão de Arquiteturas de Alocação de Tarefas em Sistema Multirrobô

Adriano Henrique Rossette Leite

Itajubá, 9 de novembro de 2017

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

Adriano Henrique Rossette Leite

rqt_mrta: Uma Interface Gráfica de Usuário baseada em ROS para configuração e supervisão de Arquiteturas de Alocação de Tarefas em Sistema Multirrobô

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

9 de novembro de 2017
Itajubá

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

rqt_mrta: Uma Interface Gráfica de Usuário baseada em ROS para configuração e supervisão de Arquiteturas de Alocação de Tarefas em Sistema Multirrobô

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: Uma Interface Gráfica de Usuário baseada em ROS para configuração e supervisão de Arquiteturas de Alocação de Tarefas em Sistema Multirrobô/ Adriano Henrique Rossette Leite. – Itajubá, 9 de novembro de 2017-
39 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, 9 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: Uma Interface Gráfica de Usuário baseada em ROS para configuração e supervisão de Arquiteturas de Alocação de Tarefas em Sistema Multirrobô

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
9 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
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*
P3DX	*Adept MobileRobots Pioneer 3 DX*
ROS	*Robot Operating System*
UNIFEI	Universidade Federal de Itajubá

Lista de símbolos

n	Número inteiro
t	Tempo
T	Período

Sumário

1 Introdução

1.1 Motivação

1.2 Objetivos

1.3 Contribuições

1.4 Estrutura do Trabalho

2 Revisão teórica

Colocar texto aqui

2.1 Sistemas Multirrobô

Colocar texto aqui

2.1.1 Taxonomias

2.2 Alocação de Tarefas em Sistema Multirrobô

Um dos problemas mais desafiadores em aplicações multi-robôs é denominado como *alocação de tarefas*, (MRTA, acrônimo para *Multi-Robot Task Allocation*). 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

2.2.2 Taxonomias

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

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; 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.

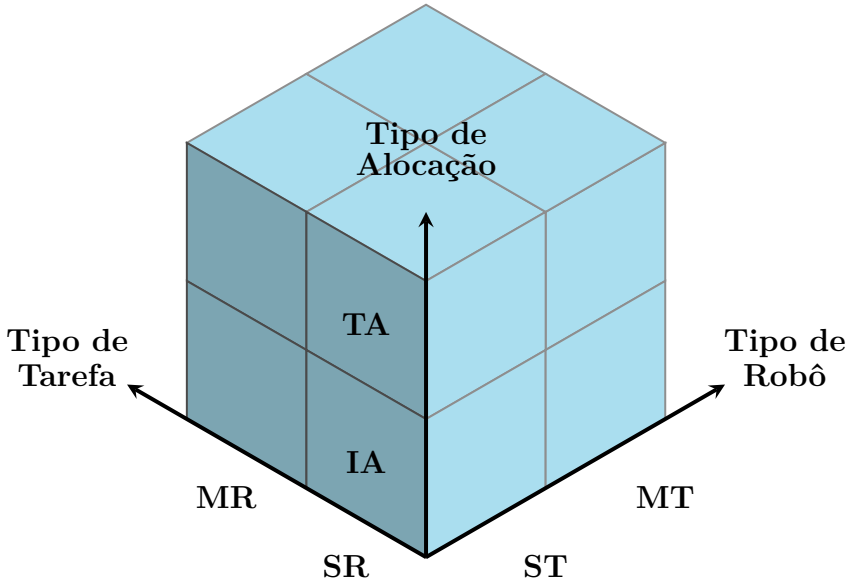


Figura 1 – Representação visual da taxonomia de três eixos sugerida por Gerkey e Matarić (2004).

É visto na Figura 1 uma representação gráfica da taxonomia de (GERKEY; 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.

dar exemplo

Uma nova taxonomia foi sugerida por ...

definir o escopo de problemas considerados neste trabalho

2.2.2.1 Arquiteturas MRTA

Possuem a função de solucionar problemas de alocação de tarefas em sistemas mutirrobô.

2.2.3 Arquiteturas baseadas em Comportamento

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

2.2.4 Arquiteturas baseadas em Mercado

- **Murdoch**: (GERKEY; MATARIC, 2002);
- **M+**: (BOTELHO; ALAMI, 1999);

2.3 ROS

Acrônimo para *Robot Operating System* (QUIGLEY et al., 2009), o ROS é uma *framework* para robótica que tem incentivado a comunidade de pesquisadores em robótica a trabalhar conjuntamente desde seu lançamento.

Primeiramente, ROS é flexível. Um projeto atômico baseado em ROS, denominado pacote, pode ser desenvolvido em diversas linguagens de programação. Deste modo, seus desenvolvedores podem tirar proveito das vantagens que cada linguagem suportada tem, sejam elas eficiência em tempo de execução, confiabilidade, recursos, síntaxe, semântica 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. Várias bibliotecas externas já foram adaptadas para ser 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.

[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>

Em adição, 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.

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.

Como resultado,

Over the years, it has been noticed that: (1) the number of contributors (academic researchers and industry) and projects have increased; (2) the applications have became more sophisticated; (3) the degree of difficulty of solved problems has ??? in different areas of robotics field; (4) the robotic industry has been more interested to contribute.

citar exemplo de middlewares para robótica

falar sobre o conteúdo deste capítulo

2.3.1 Conceitos Básicos

Sua concepção foi baseada em 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 que desempenham computação.
- **Nó Mestre**: em inglês *Master*,
- **Servidor de Parâmetros**: em inglês *Parameter Server*,
- **Tópicos**: em inglês *Topics*,
- **Serviços**: em inglês *Services*,
- **Bolsas**: em inglês *Bags*,

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.

- **ROS Wiki**[8]: 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**[9]: é uma página *web* de perguntas e respostas diretamente relacionada ao ROS.
- **Blog**[10]: providencia notícias regularmente com fotos e vídeos.

2.3.1.4 Nomes

Nomes de Recursos de Grafo no ROS

Nomes de Recursos de Pacote no ROS

Falar brevemente sobre *tf*, sobre ações e tarefas do *actionlib*, sobre a ontologia de mensagens, sobre *plugins*, sobre filtros e modelos de robôs

[8] <http://wiki.ros.org>
[9] <https://answers.ros.org/questions/>
[10] <http://www.ros.org/news/>

3 Desenvolvimento

3.1 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.

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

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 comportamento 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

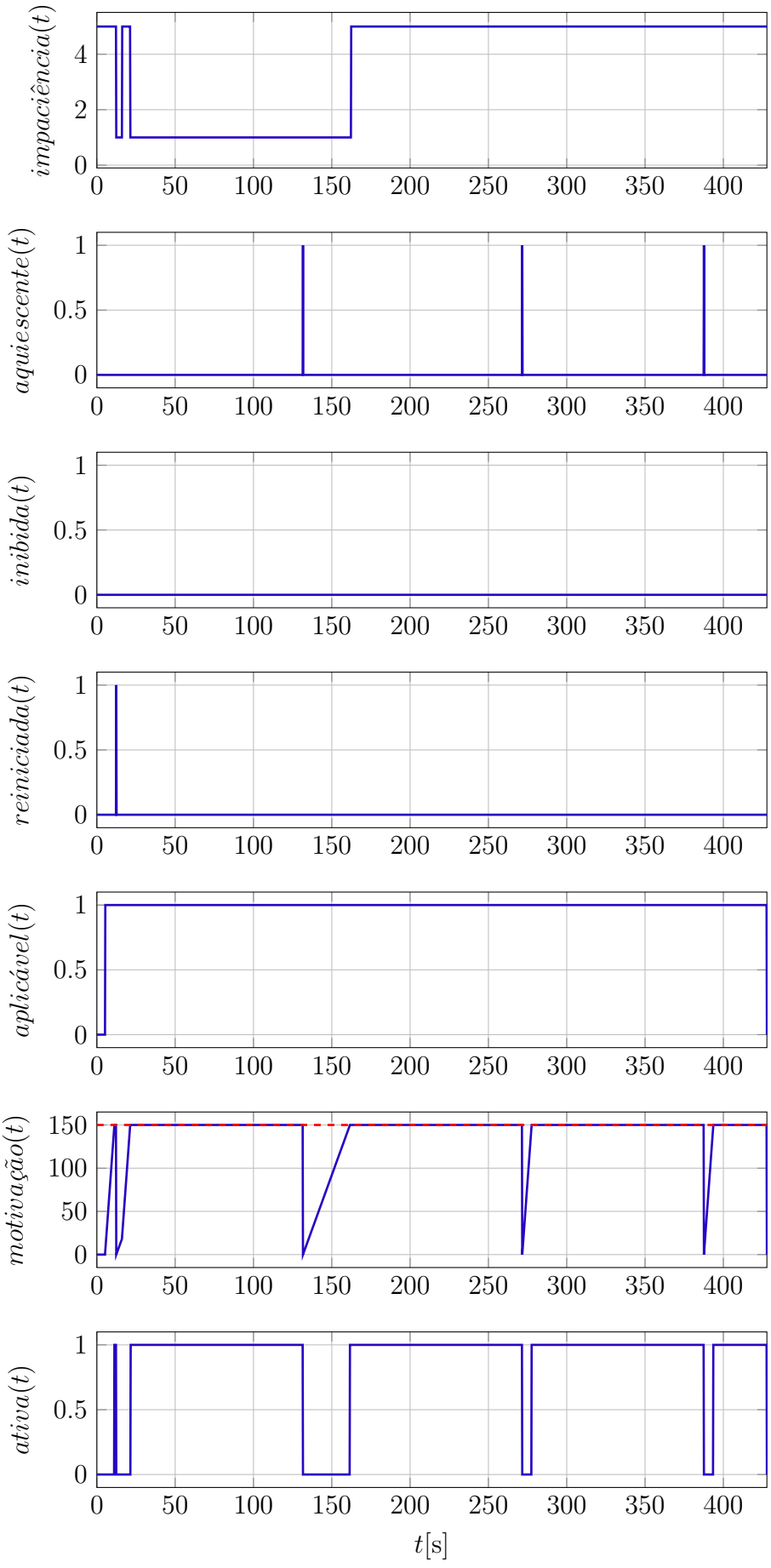


Figura 2 – Motivação da configuração de comportamento /robot1/wander.

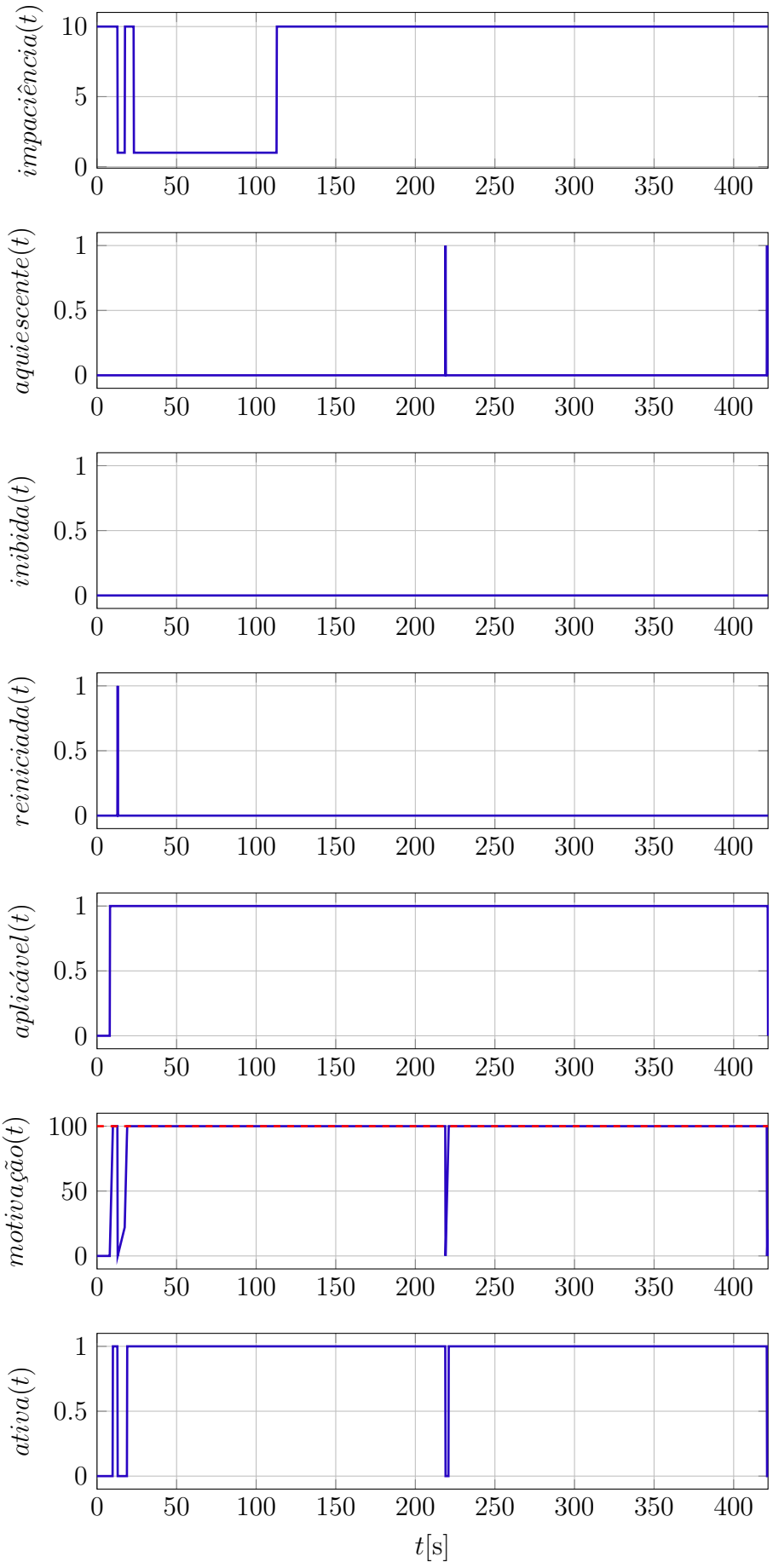


Figura 3 – Motivação da configuração de comportamento /robot2/wander.

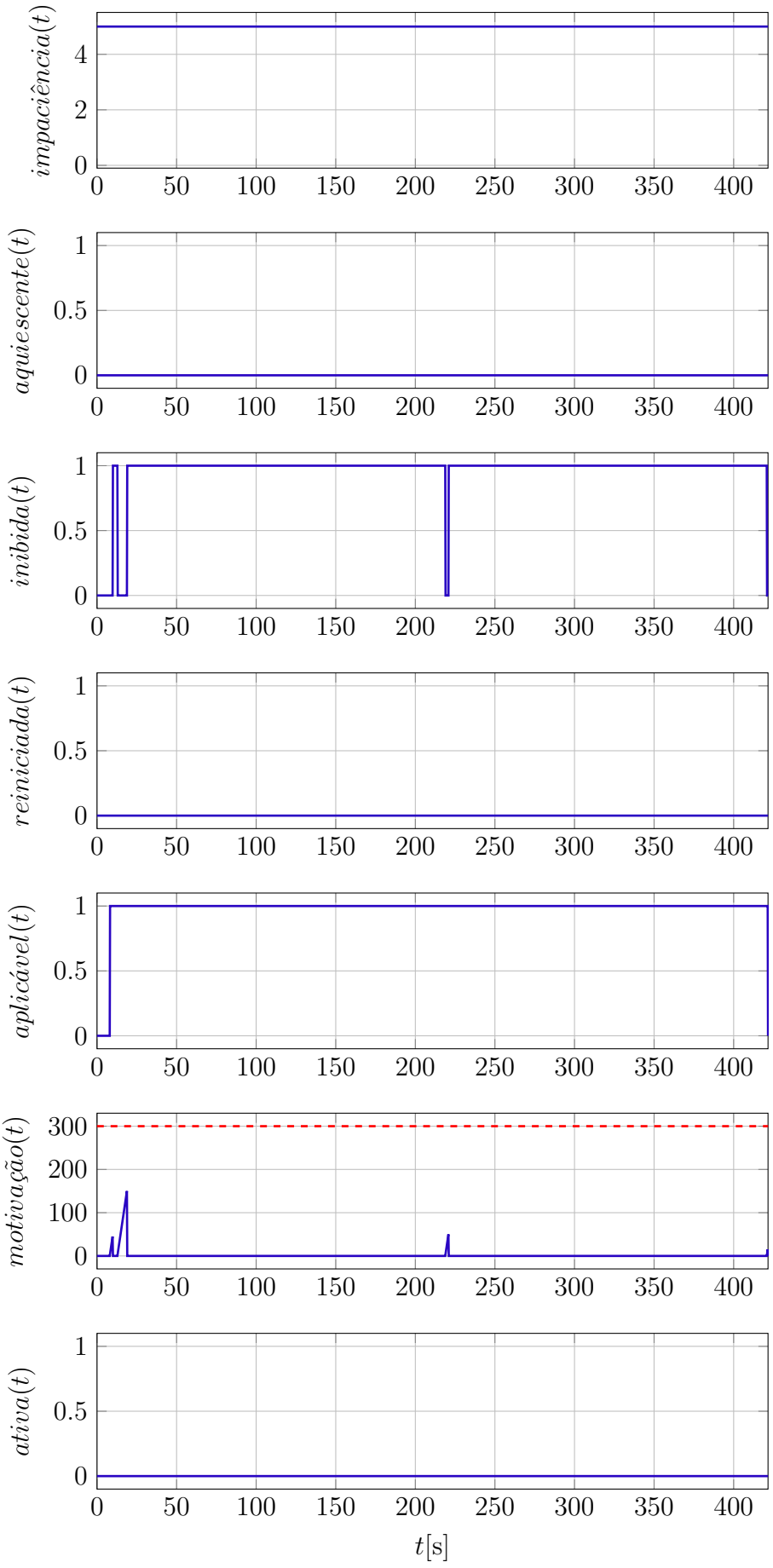


Figura 4 – Motivação da configuração de comportamento /robot2/border-protection.

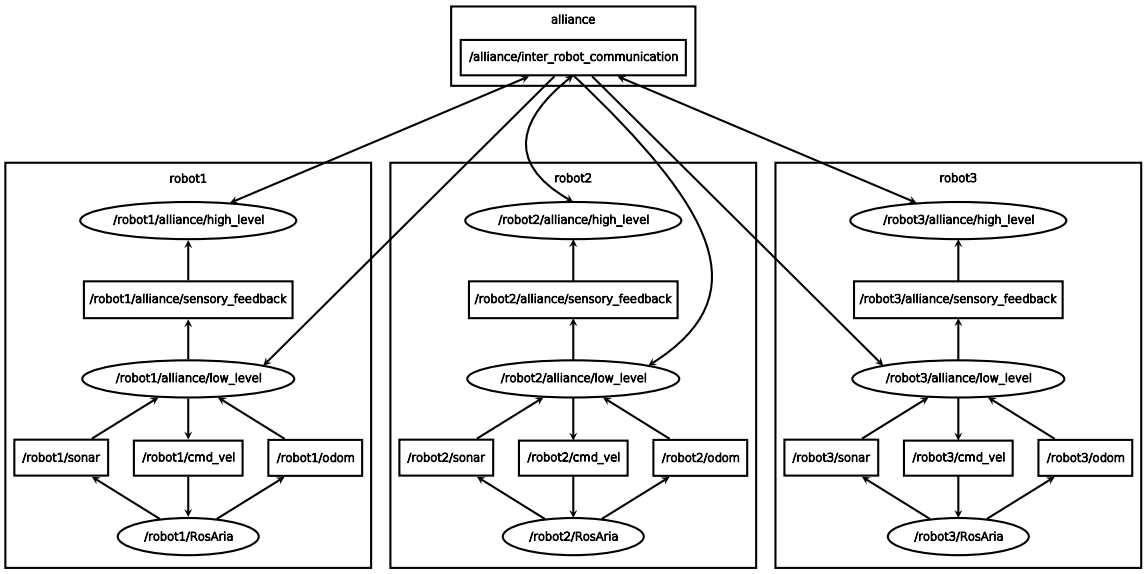


Figura 5 – Grafo do ALLIANCE no ROS para três robôs.

4 Conclusão e Trabalhos Futuros

4.1 Conclusão

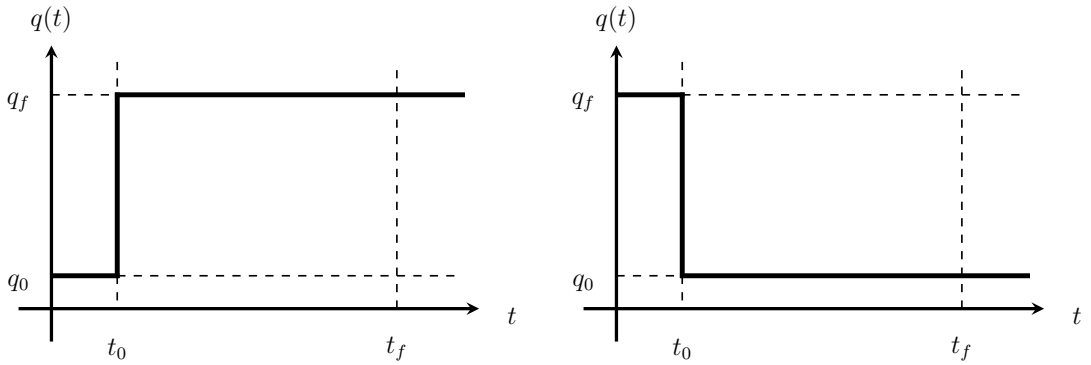
4.2 Trabalhos Futuros

Apêndices

APÊNDICE A – Funções Temporais

A.1 Funções Degraus

A Figura 6 mostra duas funções degraus: ascendente 6a e descendente 6b.



(a) Função degrau ascendente.

(b) Função degrau descendente.

Figura 6 – 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

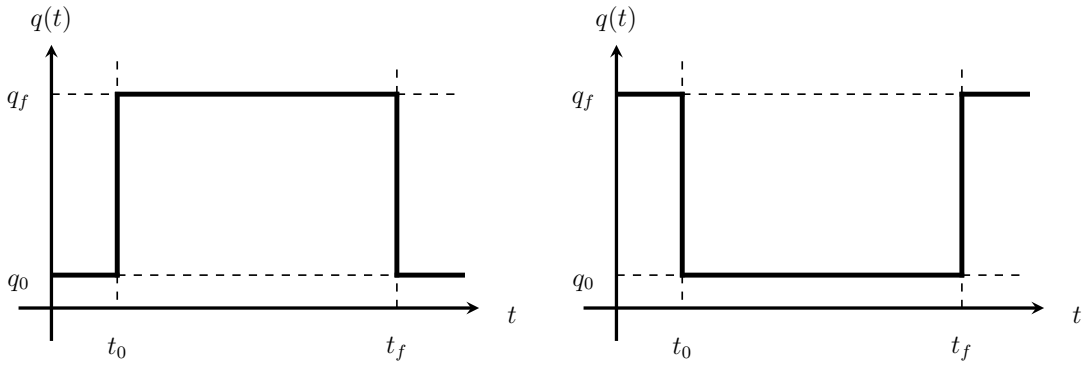
A Figura 7 mostra duas funções pulsos: ascendente 7a e descendente 7b.

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

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

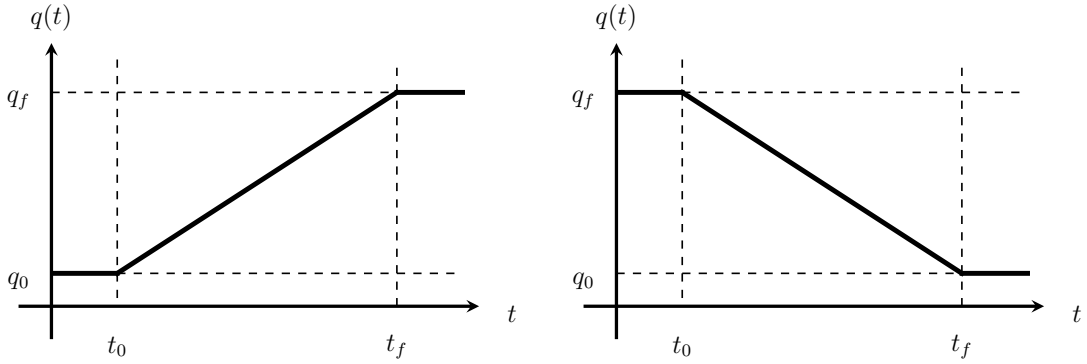
A.3 Funções Lineares

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



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

Figura 7 – Funções pulsos.



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

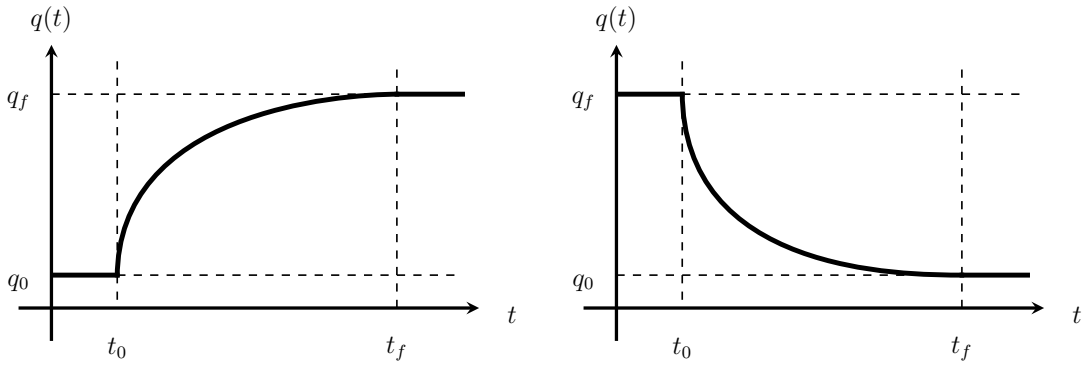
Figura 8 – Funções lineares.

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

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

A.4 Função Exponenciais

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



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

Figura 9 – 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. 18

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. 19

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. 23

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. 19

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. 17, 18

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. 19, 27

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. 19, 23

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. 19, 20