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

Projeto de um pacote ROS para a gestão de alocação de tarefas complexas em sistemas multi-robôs

Adriano Henrique Rossette Leite

Itajubá, 21 de outubro de 2017

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

Adriano Henrique Rossette Leite

Projeto de um pacote ROS para a gestão de alocação de tarefas complexas em sistemas multi-robôs

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

21 de outubro de 2017
Itajubá

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

Projeto de um pacote ROS para a gestão de alocação de tarefas complexas em sistemas multi-robôs

Adriano Henrique Rossette Leite

Dissertação aprovada por banca examinadora em 01 de Fevereiro de 2018, conferindo ao autor o título de **Mestre em Ciências em Engenharia Elétrica.**

Banca Examinadora:
Prof. Dr. Guilherme Sousa Bastos (Orientador)
(Coorientador)
Prof. Dr. Convidado1
Prof. Dr. Convidado2
Prof. Dr. Convidado3

Itajubá
2018

Adriano Henrique Rossette Leite

Projeto de um pacote ROS para a gestão de alocação de tarefas complexas em sistemas multi-robôs/ Adriano Henrique Rossette Leite. – Itajubá, 21 de outubro de 2017-

45 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, 21 de outubro 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

Projeto de um pacote ROS para a gestão de alocação de tarefas complexas em sistemas multi-robôs

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, 01 de Fevereiro de 2018:

Prof. Dr. Guilherme Sousa Bastos
Orientador

Coorientador

Prof. Dr. Convidado1

Prof. Dr. Convidado2

Prof. Dr. Convidado3

Itajubá
21 de outubro 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

Esse trabalho apresenta...

Palavras-chaves: ABC. DEF. GHI.

Abstract

This work presents ...

Key-words: ABC. DEF. GHF.

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*
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

Uma nova geração de aplicações tem crescido no campo de robótica (MOHAMED; AL-JAROODI; JAWHAR, 2008): grupos de robôs heterogêneos interconectados por uma rede de comunicação agindo cooperativamente de modo à completar uma missão comum. citar exemplos

Sempre que um grupo de robôs visa executar algum comportamento coletivo, aumenta-se em eficiência, confiabilidade, flexibilidade, escalabilidade e versatilidade para resolver tarefas complexas.

falar sobre o ROS

falar sobre o gap existente no ROS para aplicações MRS por causa da dificuldade

1.1 Visão Geral

Colocar texto aqui

1.2 Organização do trabalho

Colocar texto aqui

2 Revisão teórica

Colocar texto aqui

2.1 Sistemas Multi-robos

Colocar texto aqui

2.1.1 Taxonomias

Colocar texto aqui

2.2 Alocação de Tarefas

Colocar texto aqui

2.3 Recursos

Colocar texto aqui

2.4 Sistema Multi-Agente

2.4.1 Agentes

an **agent** is anything that can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **effectors**.

the **performance measure** determines how successful an agent is. The complete perceptual history is called **percept sequence**.

So, it is desired to measure its performance over the long run. a rational agent is not omniscient.

Saying that, we cannot blame an agent for failing to take into account something it could not perceive, or for failing to take an action that it is incapable of taking.

In other words, what is rational at any given time depends on:

- The performance measure that defines degree of success;
- Everything that the agent has perceived so far, that is, the percept sequence;

- What the agent knowsabout the environment;
- and, finally, the actions that the agent can perform.

(RUSSELL; NORVIG; INTELLIGENCE, 1995) defines an **ideal rational agent** as: *For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, an the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

ideal mapping: Specifying which action an agent ought to take in response to any given percept sequence provides a design for an ideal agent.

Talking about autonomy: If the agent's actions are based completely on built-in knownledge, such that it need pay no attention to its percepts, then we say that tthe agent lacks **autonomy**.

The job of AI is to design the **agent program**: a function that implements the agent mapping from percepts to actions. We assume this program will run on sort of computing device: the **architecture**.

There exists fours types of agent program:

- **Simple reflex agents**:
- **Agents that keep track of the world**:
- **Goal-based agents**:
- **Utility-based agents**:

This paper deals with utility-based agents!!!

To sum up, ...

2.4.2 Objetos

2.4.3 Ambiente

2.5 Sistema Multi-Robô

2.6 Alocação de Tarefas Multi-Robô

2.7 Arquiteturas de Alocação de Tarefas Multi-Robô

2.7.1 Arquiteturas baseadas em Mercado

- **Murdoch**:
- **M+**:

2.7.2 Arquiteturas baseadas em Comportamento

- **Alliance**:

Resource-constrained project scheduling problem The RCPSP problem is a generalization of the production-specific Job-Shop, Flow-Shop and Open-Shop scheduling problems. Given

a set of q resources with given capacities, a set of q resources with given capacities, a network of precedence constraints between the activities, and for each activity and each resource the amount of the resource required by the activity over its execution,

the goal of the RCPSP problem is to find a schedule meeting all the constraints whose makespan (i.e., the time at which all activities are finished) is minimal.

Resource

There are two main classes of resources: *reusable resources* and *consumable resources*.

A reusable resource is "borrowed"by an action (task) duting its execution. It is released, unchanged, when the action (task) is completed or is interrupted. A reusable resource r has a total capacity Q_r

2.8 Introduction

A new generation of applications in robotics field has been growing.

Whenever a group of robots aim to perform some collective behavior, it increases perfomance, realiability, flexibility, scalability, and versatility to resolve complex tasks. In this case, one of the most challenging problems is how to optimally assign a set of robots to a set of tasks in such a way that optimizes the overall system performance subject to a set of constraints. This problem is known as Multi-Robot Task Allocation (MRTA).

the new generation of robotics application (MOHAMED; AL-JAROODI; JAWHAR, 2008), where there exists multiple robots that are composed of heterogeneous interconnected hardware components

2.8.1 ROS

The Robot Operation System (ROS) (QUIGLEY et al., 2009) is a framework that has encouraged the robotics researchers community to work together since its release.

Firstly, ROS is flexible. An atomic ROS-based project, known as package, may be developed in multiple languages. Therefore, ROS developers can take advantage of each supported language, based on its runtime efficiency, reliability, resources, syntax, semantics, and existing documentation.

In addition, ROS is a tools-based software development framework. There are many tools that may be used during building and running time of ROS packages; such as, get and set configuration parameters, visualize the peer-to-peer connection topology, measure bandwidth utilization, graphically plot message data, and so on. The usage of its tools is highly recommended, as long as they may ensure the stability and reliability of the developed packages despite their complexity.

Moreover, it has filled the gap in middleware services once existed in the new generation of robotics applications. As a middleware services provider, ROS (1) simplifies the development process, (2) supports communications and interoperability, (3) offers and facilitates often-needed robot services, and also provides (4) efficient utilization of available resources, (5) heterogeneity abstractions and (6) automatic recourse discovery and configutarion. In order to enclose all middleware requirements, ROS 2.0 attempts to support embedded components and low-resource-devices, as well.

2.8.2 Comparing DAIs, MASs, and MRSs

Distributed Artificial Intelligence (DAI) has existed as a subfield of AI for less than two decades. DAI is concerned with systems that consist of multiple independent entities that interact in a domain. Traditionally, DAI has been divided into two sub-disciplines: Distributed Problem Solving (DPS) focuses on the information management aspects of systems with several components working together towards a common goal; Multiagent Systems (MAS) deals with behavior management in collections of several independent

entities, or agents. This survey of MAS is intended to serve as an introduction to the field and as an organizational framework. A series of general multiagent scenarios are presented. For each scenario, the issues that arise are described along with a sampling of the techniques that exist to deal with them. The presented techniques are not exhaustive, but they highlight how multiagent systems can be and have been used to build complex systems. When options exist, the techniques presented are biased towards machine learning approaches. Additional opportunities for applying machine learning to MAS are highlighted and robotic soccer is presented as an appropriate test bed for MAS. This survey does not focus exclusively on robotic systems. However, we believe that much of the prior research in non-robotic MAS is relevant to robotic MAS, and we explicitly discuss several robotic MAS, including all of those presented in this issue (STONE; VELOSO, 2000).

3 ROS

Acrônimo para *Robot Operting System* (QUIGLEY et al., 2009), o ROS é uma *framework* 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.

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 componentes

[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>

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

3.1 Conceitos Básicos

4 Alocação de Tarefas em Sistemas Multi-Robôs

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 deste capítulo

4.1 Definição Formal

Colocar texto aqui

4.2 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.

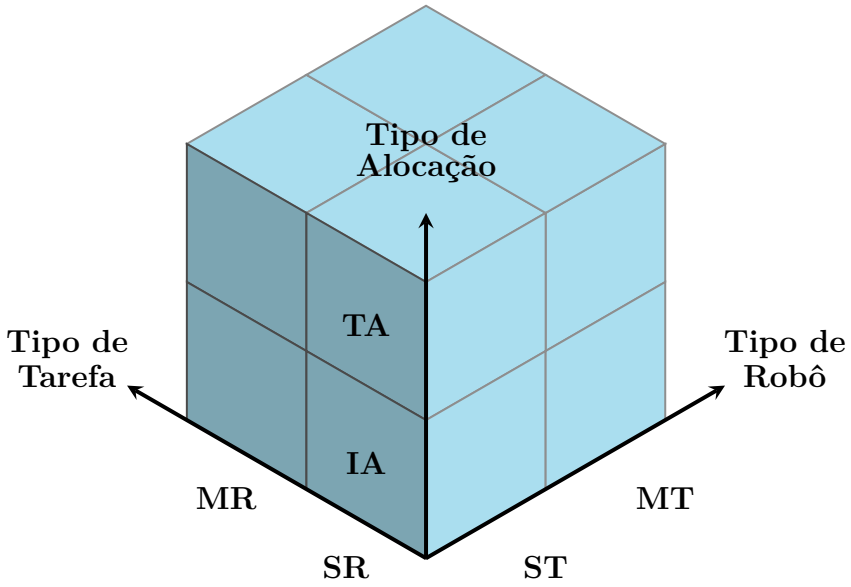


Figura 1 – Representação visual da taxonomia de três eixos sugerida por (GERKEY; MATARIĆ, 2004).

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

4.3 Arquiteturas MRTA

4.3.1 Arquiteturas baseadas em Comportamento

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

4.3.2 Arquiteturas baseadas em Mercado

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

4.4 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 4.1, 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.

$$aplicável_{ij}(t) = \begin{cases} 1, & \text{se o módulo de } \textit{feedback} \text{ sensorial da configuração de comportamento } c_{ij} \text{ do robô } r_i \text{ indicar que esta configuração é aplicável mediante ao estado atual do ambiente no instante } t; \\ 0, & \text{caso contrário.} \end{cases} \tag{4.1}$$

A Equação 4.2 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.

$$inibida_{ij}(t) = \begin{cases} 1, & \text{se outra configuração de comportamento } c_{ik} \text{ (com } k \neq j\text{) está ativa no robô } r_i \text{ no instante } t; \\ 0, & \text{caso contrário.} \end{cases} \tag{4.2}$$

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 4.3. 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.

$$recebida_{ij}(k, t_1, t_2) = \begin{cases} 1, & \text{se o robô } r_i \text{ recebeu mensagem do robô } r_k \text{ referente à tarefa } h_i(c_{ij}) \text{ dentro do intervalo de tempo } [t_1; t_2], \text{ em que } t_1 < t_2; \\ 0, & \text{caso contrário.} \end{cases} \tag{4.3}$$

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 4.3 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.

$$reiniciada_{ij}(t) = \exists x, (recebida_{ij}(x, t - dt, t) \wedge \neg recebida_{ij}(x, 0, t - dt)) \tag{4.4}$$

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

A Equação 4.5 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.

$$ativa_{ij}(\Delta t, t) = \begin{cases} 1, & \text{se a configuração de comportamento } c_{ij} \text{ do} \\ & \text{robô } r_i \text{ estiver ativa por mais de } \Delta t \text{ unidades} \\ & \text{de tempo no instante } t; \\ 0, & \text{caso contrário.} \end{cases} \tag{4.5}$$

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 4.5. 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 4.6 define as condições em que r_i está aquiescente à desativação de c_{ij}.

$$\begin{aligned} aquiescente_{ij}(t) = (&ativa_{ij}(\psi_{ij}(t), t) \wedge \exists x, recebida_{ij}(x, t - \tau_i, t)) \\ &\vee ativa_{ij}(\lambda_{ij}(t), t) \end{aligned} \tag{4.6}$$

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 4.7. 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 $h_i(c_{ij})$.

$$impaciência_{ij}(t) = \begin{cases} \min\limits_{x} \delta_{slow_{ij}}(x,t), & \text{se } recebida_{ij}(x, t-\tau_i, t) \wedge \neg recebida_{ij}(x, 0, t- \\ & \phi_{ij}(x,t); \\ \delta_{fast_{ij}}(t), & \text{caso contrário.} \end{cases} \tag{4.7}$$

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 4.8 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.

$$\begin{aligned} motivação_{ij}(0) &= 0 \\ motivação_{ij}(t) &= (motivação_{ij}(t-dt) + impaciência_{ij}(t)) \\ &\quad \times aplicável_{ij}(t) \times inibida_{ij}(t) \\ &\quad \times reiniciada_{ij}(t) \times aquiescente_{ij}(t). \end{aligned} \tag{4.8}$$

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 4.9:

$$ativa_{ij}(t) = motivação_{ij}(t) \geq \theta \tag{4.9}$$

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.

$$\overline{\Delta t}_{ativação_{ij}} = \frac{\theta - motivação_{ij}(t)}{impaciência_{ij}(t)\rho_i} \tag{4.10}$$

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 4.10 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 4.9 e 4.10 podem ser rescritas como as Equações 4.11 e 4.12, respectivamente.

$$ativa_{ij}(t) = motivação_{ij}(t) == 1 \tag{4.11}$$

$$\overline{\Delta t}_{ativação_{ij}} = \frac{1 - motivação_{ij}(t)}{impaciência_{ij}(t)\rho_i} \tag{4.12}$$

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

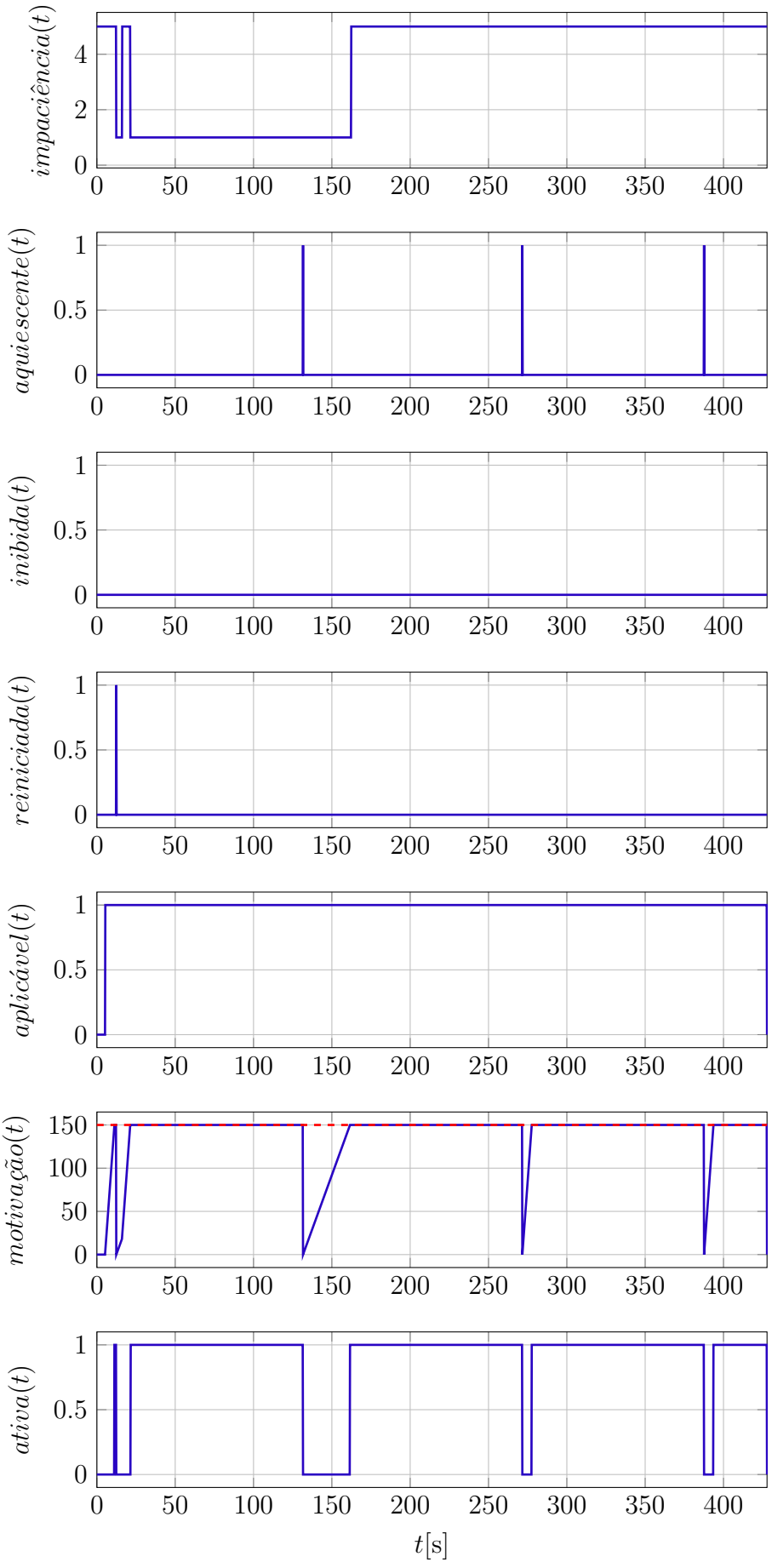


Figura 2 – Motivação da configuração de comportamento /robot1/wander.

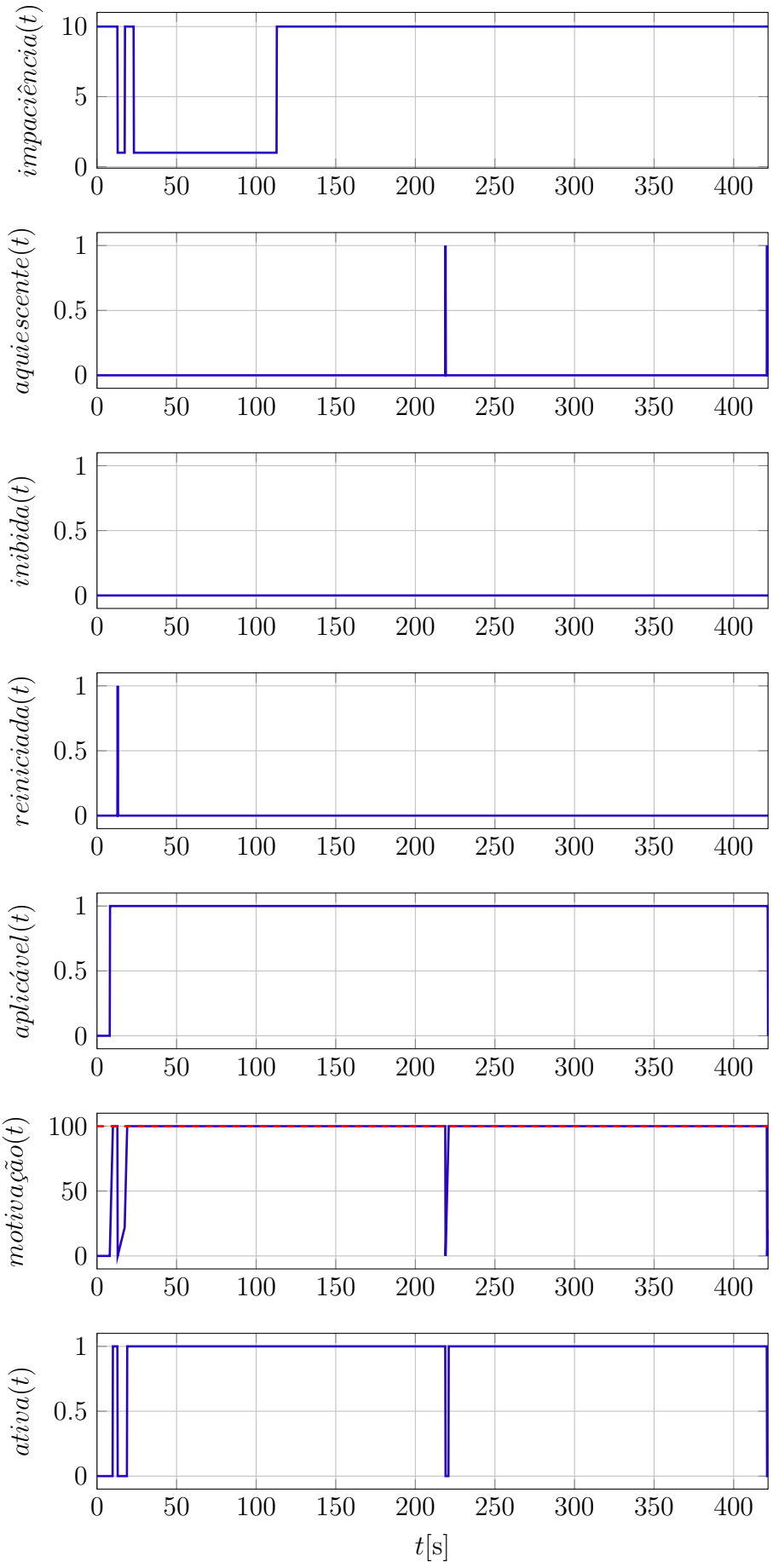


Figura 3 – Motivação da configuração de comportamento /robot2/wander.

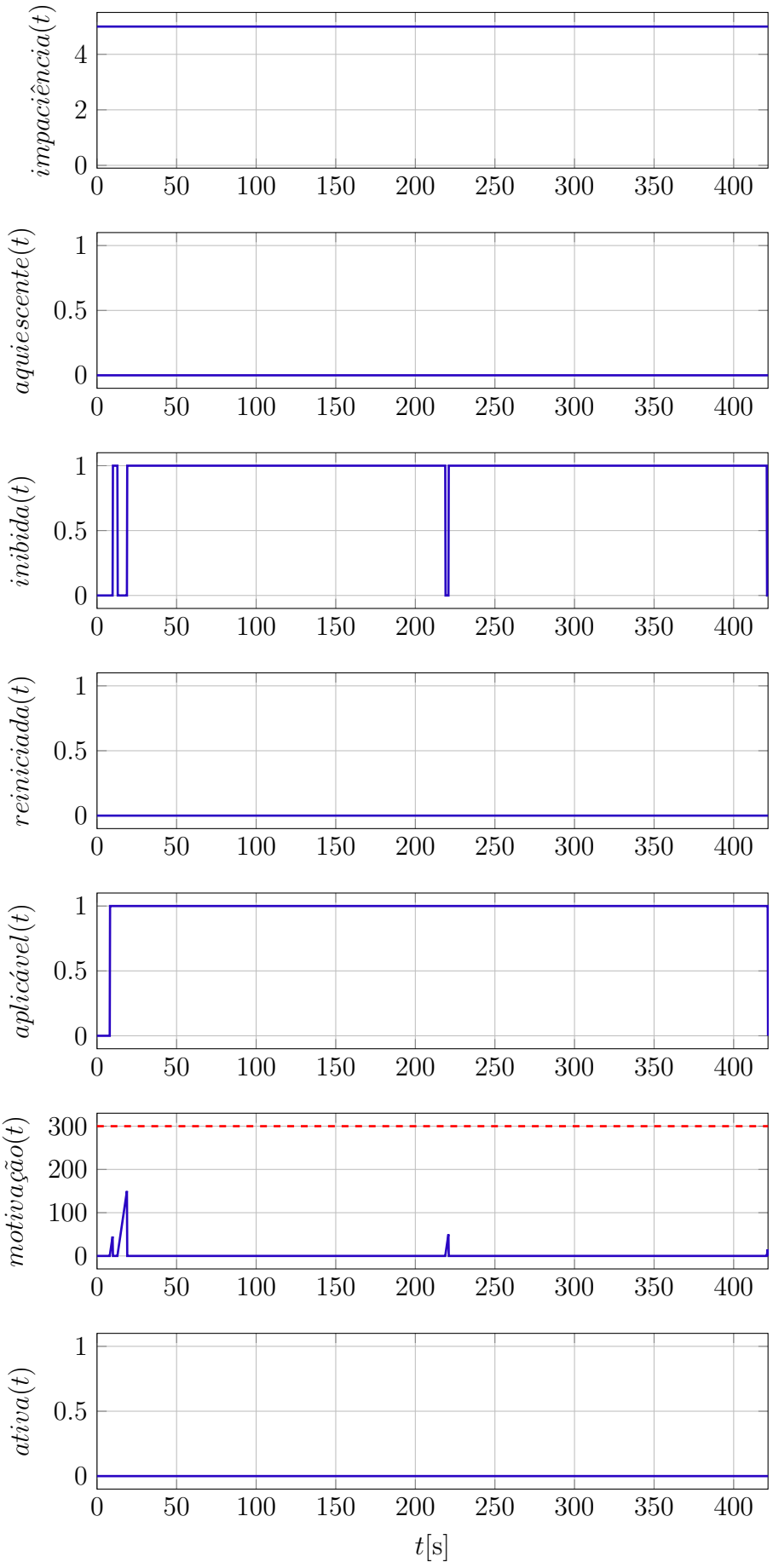


Figura 4 – Motivação da configuração de comportamento /robot2/border-protection.

5 Recursos

Recursos são altamente utilizados, compartilhados, produzidos e consumidos em aplicações industriais e de robótica. Para que os robôs sejam capazes de executar uma dada tarefa, normalmente é requerido que eles possuam um conjunto de recursos, sendo especificado a quantidade necessária de cada um deles. Por exemplo, para que um robô móvel possa adquirir e processar uma imagem digital, como parte de uma dada tarefa, é preciso que esse possua uma câmera digital e, dependendo da taxa de processamento desejada, um processador dedicado especialmente para a renderização de gráficos em tempo real, isto é, uma GPU (*Graphics Processing Unit*).

Seja uma linha de montagem para a produção em série de uma determinado produto, composta por grupo de robôs heterogêneos e, também, por máquinas automáticas. Os processos são realizados pelas máquinas de forma sequencial. E os robôs são responsáveis por transportar matéria-prima e produto semi-acabado de um processo a outro. Além disso, eles devem abastecer os suprimentos das máquinas quando notificados. *As máquinas possuem sinalizadores que identificam em qual estado elas se encontram: (1) aguardando, modo em que a máquina está ociosa, esperando um novo produto semi-acabado entrar para processar; (2) preparando, modo em que a máquina se configura para processar o produto semi-acabado recebido; (3)* ***processando peça****, modo em que a máquina está trabalhando o produto semi-acabado; (4)* ***peça processada****, produto já recebeu todo trabalho necessário nesta etapa; (5) danificada, ; (6) em manutenção, ; e (7) em abastecimento, .*

falar sobre o conteúdo deste capítulo

5.1 Tarefas

Uma tarefa a pode ser especificada pelos seus requerimentos de recurso, pelo seu tempo de início $s(a)$, pelo seu tempo de término $e(a)$, bem como, sua duração $d(a)$. Em aplicações reais, o tempo de início, de término e a duração de uma tarefa são estimações apenas. Assim, $s(a)$ e $e(a)$ são especificados como variáveis estocásticas que possuem uma alta probabilidade de ocorrer dentro dos intervalos: $s(a) \in [s_{min}(a); s_{max}(a)]$ e $e(a) \in [e_{min}(a); e_{max}(a)]$. Podemos ainda definir esses dois parâmetros como distribuições normais:

$$s(a) \sim \mathcal{N}(\mu_{s(a)}, \sigma^2_{s(a)})$$
$$e(a) \sim \mathcal{N}(\mu_{e(a)}, \sigma^2_{e(a)})$$

Neste caso, com uma probabilidade de $99,7\%$, teremos:

$$s_{min}(a) \approx \mu_{s(a)} - 3\sigma_{s(a)}$$
$$s_{max}(a) \approx \mu_{s(a)} + 3\sigma_{s(a)}$$
$$e_{min}(a) \approx \mu_{e(a)} - 3\sigma_{e(a)}$$
$$e_{max}(a) \approx \mu_{e(a)} + 3\sigma_{e(a)}$$

Tarefas podem ainda ser *preemptivas* ou *não-preemptivas*. Tarefas não-preemptivas devem ser executadas sem interrupções, de modo que, $d(a) = e(a) - s(a)$. Entretanto, podem ocorrer interrupções durante a execução de tarefas preemptivas. Neste caso, os recursos utilizados por a são liberados para que outras tarefas possam usá-los. O cálculo da duração de tarefas preemptivas é dado por:

$$d(a) = \sum_{i=1}^{k} d_i \leq e(a) - s(a)$$

em que $d_i(a)$ é a duração de cada interrupção. *Serão consideradas, no restante deste trabalho, apenas tarefas não-preemptivas.*

Contudo, o foco deste capítulo e deste trabalho está nos requerimentos de recurso que as tarefas da missão fazem aos robôs do sistema. Conforme dito anteriormente, ao especificar uma tarefa é necessário informar quais são os recursos necessários para sua execução e completude. Assim, nos próximos parágrafos deste capítulo, serão definidos as classes e os tipos de recursos existentes.

Existem duas classes de recursos, basicamente: aqueles que podem ser usados novamente, os quais são denominados *recursos reusáveis*; e aqueles que são descartáveis, denominados *recursos consumíveis* (GHALLAB; NAU; TRAVERSO, 2004). Cada classe de recurso é caracterizada por um *perfil*. O perfil de um recurso r é uma função do tempo, $z_r(t)$, que define sua quantidade instantânea. A seguir, cada classe de recurso será definida, bem como, será detalhado o perfil de cada uma delas.

5.2 Recursos Reusáveis

Um recurso reusável é "emprestado" por uma tarefa durante sua execução. E quando ela é interrompida ou finalizada, esse recurso é liberado sem alterações. Assim, um recurso reusável r tem uma capacidade total Q_r e uma quantidade corrente $z_r(t) \in [0; Q_r]$.

Seja uma tarefa a que requer durante a sua execução uma quantidade q do recurso r. Ao ser iniciada, no tempo $s(a)$, a quantidade corrente de r é diminuída em um montante q. Entretanto, quando a é finalizada, no tempo $e(a)$, a quantidade corrente de r é aumentada pelo mesmo montante, q. Portanto, um recurso reusável possui um perfil característico conforme mostra a Figura 5.

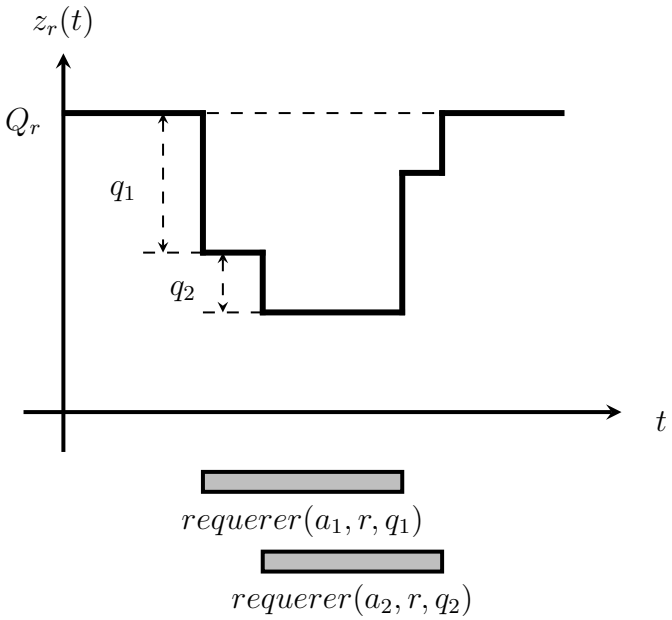


Figura 5 – Perfil característico de um recurso reusável.

Verifica-se, na Figura 5, que o recurso r possui inicialmente uma quantidade equivalente à sua capacidade total, isto é, $z_r(0) = Q_r$. Isso acontece pois nenhuma tarefa fez requisição do recurso r. Entretanto, quando a tarefa a_1 é iniciada, a quantidade do recurso r, em $s(a_1)$, passa a ser $z_r(s(a_1)) = Q_r - q_1$. Em $s(a_2)$, a tarefa a_2 é iniciada. Com isso, a quantidade de r passa a valer $z_r(s(a_2)) = Q_r - q_1 - q_2$, pois a tarefa a_1 ainda está em execução, utilizando uma quantia q_1 de r. Quando a_1 é finalizada, no instante $e(a_1)$, essa deixa de utilizar o recurso r, o qual passa a ter uma quantidade $z_r(e(a_1)) = Q_r - q_2$. Finalmente, ao término de a_2, em $e(a_2)$, todas as requisições de r se encerram, fazendo com que sua quantidade volte ao valor de sua capacidade, de modo que, $z_r(e(a_2)) = Q_r$. É importante lembrar que, após cada variação de quantidade, a quantidade do recurso reusável r se mantém constante até que um novo evento ocorra, ou seja, até que uma nova requisição seja iniciada ou encerrada.

5.3 Recursos Consumíveis

Um recurso consumível pode ser produzido ou consumido durante a execução de uma tarefa. Esta classe de recurso pode ser modelada como um reservatório de capacidade máxima limitada Q_r e nível (quantidade) corrente $z_r(t) \in [0; Q_r]$.

Seja, pois, uma tarefa a que produz uma quantidade q do recurso r durante a sua execução. Quando iniciada, em $s(a)$, aumenta um montante q do seu nível $z_r(t)$ ao longo do tempo. Essa produção é modelada por uma função dependente do tempo, crescente no intervalo temporal $[s(a); e(a)]$.

Seja, agora, uma tarefa a que consome uma quantidade q do recurso r durante

a sua execução. Quando iniciada, no instante $s(a)$, reduz um montante q do seu nível $z_r(t)$ ao longo do tempo. Essa redução/consumo é modelada como uma função do tempo, decrescente no intervalo de tempo $[s(a); e(a)]$. Portanto, um recurso consumível possui um perfil característico conforme mostrado na Figura 6.

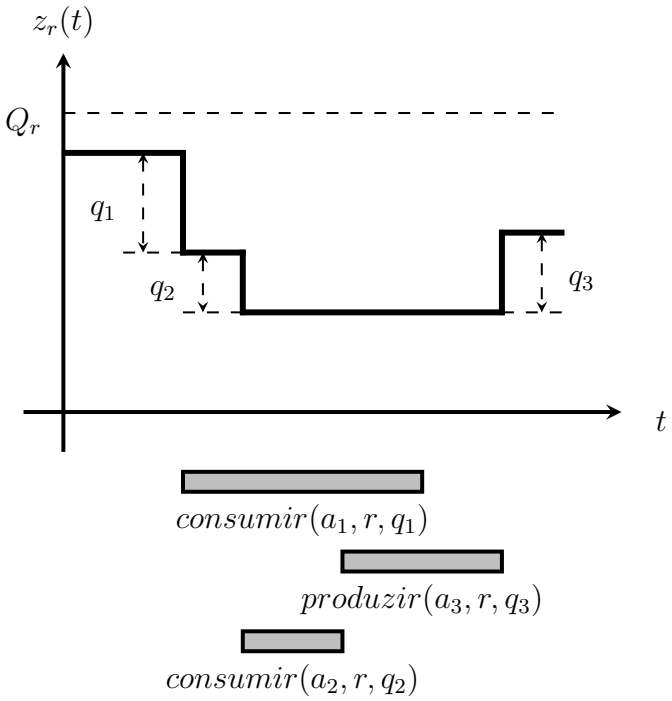


Figura 6 – Perfil característico de um recurso consumível.

A Figura 6 mostra que o recurso r possui inicialmente um valor menor que sua capacidade. Considerando $z_r(0) = z_0$ com $z_0 \in (0; Q_r]$, ao iniciar a_1, em $s(a_1)$, r começa a ser consumido por a_1, evoluindo como uma função degrau ao longo do tempo. Em $s(a_2)$, inicia-se o consumo de r pela tarefa a_2. A quantidade consumida por a_2 também varia no tempo como uma função degrau. Quando em $s(a_3)$, inicia-se a tarefa a_3 e, simultaneamente, se termina a execução da tarefa a_2. Neste instante, inicia-se a produção do recurso r, progredindo como uma função degrau ao final de a_3. Ao final da execução de cada tarefa, o efeito que cada uma tem sobre a quantidade do recurso consumível r é nulo. Note a diferença entre essa classe de recurso com a classe de recursos reusáveis. Ao lidar com recursos reusáveis, ao término de cada tarefa, é devolvida a quantidade utilizada durante sua execução. Enquanto, ao lidar com recursos consumíveis, não há devoluções para sua quantidade no instante em que a execução da tarefa termina.

São mostrados alguns modelos de funções temporais no Apêndice A. A seguir serão citados os tipos dos modelos sugeridos no Apêndice A para a função temporal de produção de um dado recurso consumível r: (1) a Figura 7a mostra um modelo do tipo degrau; (2) a Figura 9a mostra um modelo do tipo linear; e, enfim, (3) a Figura 10a mostra um modelo do tipo exponencial. Da mesma forma, serão agora citados os tipos dos modelos sugeridos no Apêndice A para a função temporal de consumo de um dado recurso consumível r:

(1) é mostrado na Figura 7b um modelo do tipo degrau; (2) na Figura 9b, é mostrado um modelo do tipo linear; e, por fim, (3) é mostrado na Figura 10b um modelo do tipo exponencial.

5.4 Tipos de Recurso

Recursos possuem um tipo, podendo ele ser: (1) contínuo, (2) discreto ou (3) unário.

Primeiramente, em *recursos contínuos*, a capacidade total do recurso é definida por um número pertencente ao conjunto dos números reais estritamente positivos, enquanto sua quantidade corrente é uma representação numérica que pertence ao conjunto dos números reais não-negativos. Assim temos,

$$z_r : t \in \mathbb{R}_+ \rightarrow z \in [0; Q_r] \subset \mathbb{R}_+ \mid Q_r \in \mathbb{R}_+^* \tag{5.1}$$

Exemplificando ... *(dar exemplo(s) de recursos reusáveis contínuos e recursos consumíveis contínuos. Falar do tipo de funções tbm: step, pulse, linear e exponential)*

Recursos discretos possuem capacidade total definida por um número inteiro estritamente positivo, isto é, um número natural, e quantidade corrente por um número inteiro não-negativo, obtendo:

$$z_r : t \in \mathbb{R}_+ \rightarrow z \in [0; Q_r] \subset \mathbb{Z}_+ \mid Q_r \in \mathbb{N} \tag{5.2}$$

Exemplificando ... *(dar exemplo(s) de recursos reusáveis discretos e recursos consumíveis discretos. Falar do tipo de funções tbm: step, pulse, linear e exponential)*

E, finalmente, um *recurso unário* sempre possui capacidade total igual à 1 e sua quantidade corrente pode assumir os valores 0 ou 1. Com isso, podemos concluir que a quantidade corrente do recurso informa sua disponibilidade ao longo do tempo: quando 0, o recurso se encontra indisponível; e, quando 1, o recurso está disponível. Em outras palavras,

$$z_r : t \in \mathbb{R}_+ \rightarrow z \in \{0; 1\} \quad \text{e} \quad Q_r = 1 \tag{5.3}$$

Exemplificando ... *(dar exemplo(s) de recursos reusáveis unários e recursos consumíveis unários. Falar do tipo de funções tbm: step, pulse, linear e exponential)*

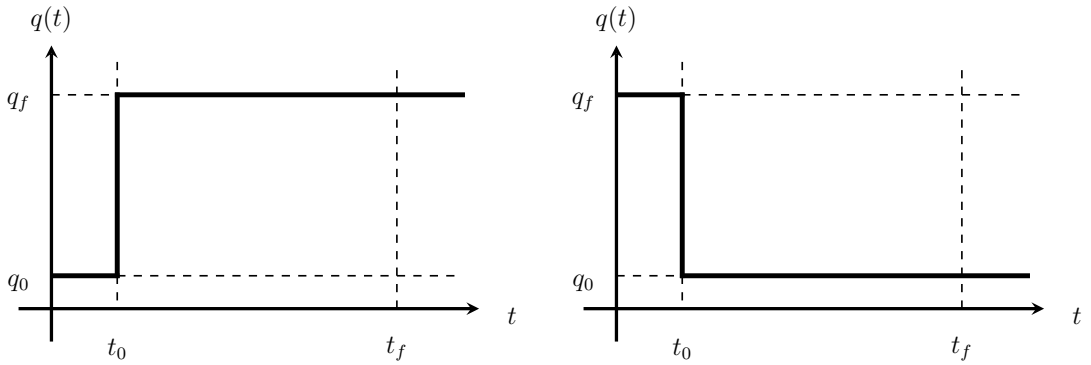
... *(dar de uma aplicação com várias tarefas que utilizam recursos de classes e tipos variados.)*

Apêndices

APÊNDICE A – Funções Temporais

A.1 Funções Degraus

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



(a) Função degrau ascendente.

(b) Função degrau descendente.

Figura 7 – 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

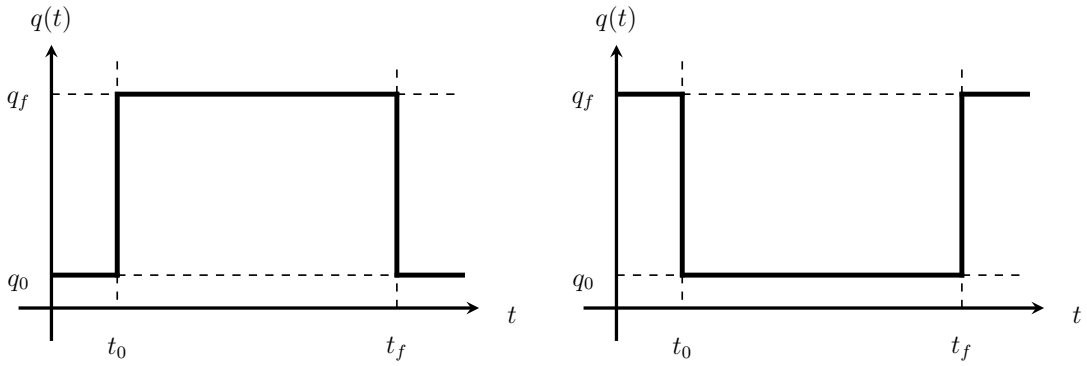
A Figura 8 mostra duas funções pulsos: ascendente 8a e descendente 8b.

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

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

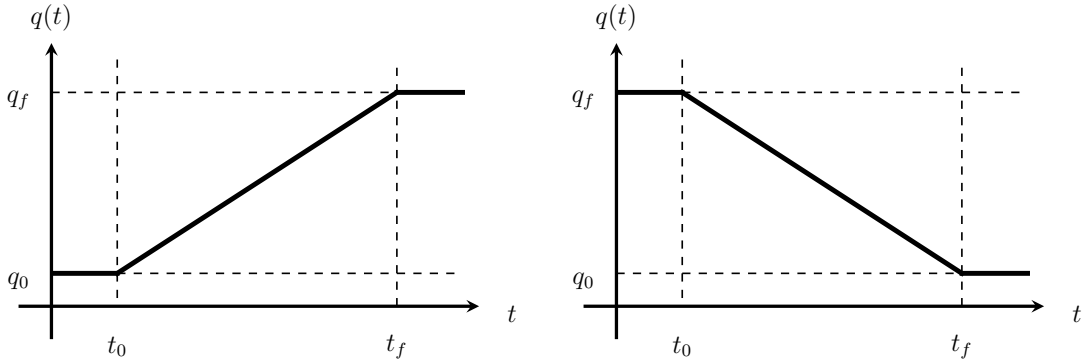
A.3 Funções Lineares

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



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

Figura 8 – Funções pulsos.



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

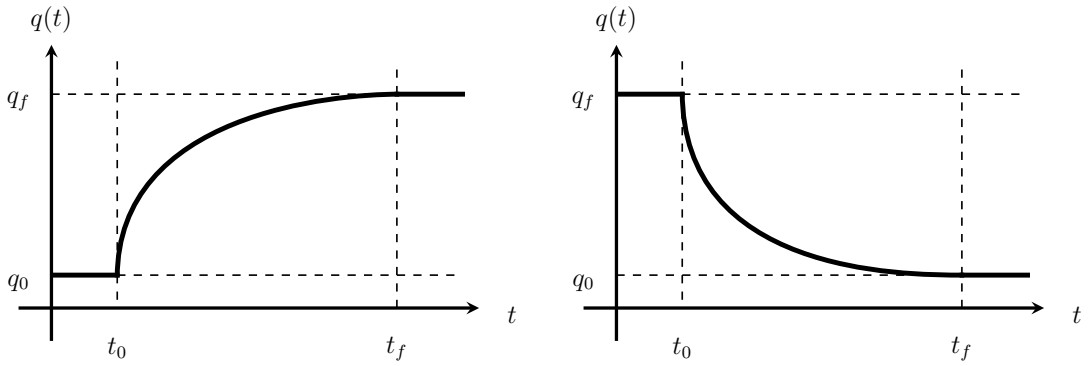
Figura 9 – Funções lineares.

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

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

A.4 Função Exponenciais

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



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

Figura 10 – 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. 25

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. 26

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. 26

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. 26

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. 10, 24, 25

GHALLAB, M.; NAU, D.; TRAVERSO, P. *Automated Planning: theory and practice*. [S.l.]: Elsevier, 2004. 35

MOHAMED, N.; AL-JAROODI, J.; JAWHAR, I. Middleware for robotics: A survey. In: IEEE. *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*. [S.l.], 2008. p. 736–742. 16, 20

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. 25, 30

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. 25, 26

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. 20, 22

RUSSELL, S.; NORVIG, P.; INTELLIGENCE, A. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, Citeseer, v. 25, 1995. 18

STONE, P.; VELOSO, M. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, v. 8, n. 3, p. 345–383, 2000. 21