

UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

INSTITUTO DE COMPUTAÇÃO - ICOMP

PROGRAMA PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

Uma Abordagem Baseada em Engenharia Dirigida por Modelos e Aprendizado de Máquina Aplicado a Robôs Móveis

Edson de Araújo Silva

Manaus - AM

2022

Edson de Araújo Silva

Uma Abordagem Baseada em Engenharia Dirigida por Modelos e Aprendizado de Máquina Aplicado a Robôs Móveis

Tese submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Amazonas como requisito parcial para obtenção do título de Doutor em Informática. Área de concentração: Engenharia de Software e Sistemas Embarcados

Orientador

Raimundo da Silva Barreto, Dr.

Universidade Federal do Amazonas - UFAM

Instituto de Computação - IComp

Manaus - AM

2022

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Silva, Edson de Araújo
S586a Uma Abordagem Baseada em Engenharia Dirigida por Modelos e Aprendizado de Máquina Aplicado a Robôs Móveis / Edson de Araújo Silva . 2022
80 f.: il. color; 31 cm.

Orientador: Raimundo da Silva Barreto
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Robótica. 2. Aprendizado de máquina. 3. Aprendizado por reforço. 4. Engenharia dirigida por modelo. I. Barreto, Raimundo da Silva. II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Uma Abordagem Baseada em Engenharia Dirigida por Modelos e Aprendizado de Máquina Aplicado a Robôs Móveis"

EDSON DE ARAÚJO SILVA

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Raimundo da Silva Barreto - PRESIDENTE

Prof. José Reginaldo Hughes Carvalho - MEMBRO INTERNO

Prof. Juan Gabriel Colonna - MEMBRO INTERNO

Prof. José Luiz de Souza Pio - MEMBRO EXTERNO

Prof. Fernando Santos Osório - MEMBRO EXTERNO

Manaus, 19 de dezembro de 2022

Dedico esta Tese a minha esposa Suellen e aos meus filhos Ícaro, Enzo e Evan.

Agradecimentos

Todo esforço, dedicação e perseverança não teriam valor se neste momento não olhasse para trás e pudesse demonstrar a gratidão aos que, de alguma forma, contribuíram para a realização deste, tão sonhado, objetivo em minha vida. Primeiramente, Àquele que nos concedeu o dom da vida e guia nosso espírito afim de seguirmos por um bom caminho, à Deus.

Agradeço a minha família, em especial a minha esposa Suellen Aline, por compreender esse momento em que a dedicação e o foco foram necessários para condução da pesquisa, mesmo que em detrimento a momentos junto aos familiares. Também por embarcar nessa jornada sem saber o que estaria por vir e mesmo assim sendo capaz de se desfazer do que havíamos construído juntos na certeza de que investimentos são necessários para progredir em busca de uma vida melhor.

Aos meus filhos Ícaro, Enzo e Evan, por fazerem meus dias mais felizes, me lembrando a cada dia que o carinho e afeto são a mais pura forma de amor que existe. Agradeço a Deus a oportunidade de poder crescer junto a vocês e poder acompanhar o caminho que estão trilhando em suas vidas.

Ao meu Orientador, Prof. Dr. Raimundo Barreto, por ser essa pessoa excepcional e que em todos os momentos esteve presente durante esse período de formação, não só como docente e pesquisador, mas como pessoa e amigo. Seus ensinamentos e palavras levarei sempre comigo e espero poder retribuir todo o aprendizado proporcionado, aos meus alunos e sociedade. Agradeço também ao Prof. Dr. José Reginaldo que sempre esteve disposto a contribuir com a pesquisa e foi de grande importância nos conhecimentos adquiridos na área da robótica. Sua capacidade de síntese e compreensão dos

mais diversos assuntos sempre me deixou impressionado. Meus agradecimentos ao Prof. Dr. Juan Colonna que fez parte de um ciclo de atividades executadas durante o desenvolvimento do projeto e contribuiu com seus valiosos conhecimentos em Aprendizado de Máquina.

Quero fazer um agradecimento especial aos meus colegas e amigos, Anderson Cruz, Gabriel Leitão e Márcio Alencar, que me receberam de braços abertos no Laboratório de Sistemas Embarcados onde desenvolvemos nossas atividades durante esse período. Por todo o apoio e troca de conhecimentos partilhado contribuindo para meu crescimento profissional.

Agradeço a todo corpo docente e técnico do ICOMP responsáveis por elevar o Programa a nível de excelência e proporcionar um ambiente propício ao desenvolvimento de pesquisas internacionais.

Agradeço a Fundação de Amparo à Pesquisa do Estado do Amazonas (FAPEAM) por conceder uma bolsa de estudos, sem a qual tornaria o trabalho mais desafiador. Isso permitiu que a pesquisa fosse desenvolvida com a dedicação e empenho necessários a sua conclusão. A todos, muito obrigado.

Tudo o que fizerem façam de coração, como quem obedece ao Senhor, e não aos homens.

Colossenses 3:23.

Uma Abordagem Baseada em Engenharia Dirigida por Modelos e Aprendizado de Máquina Aplicado a Robôs Móveis

Autor: Edson de Araújo Silva

Orientador: Raimundo da Silva Barreto, Dr.

Resumo

A aplicação de robôs móveis em ambientes complexos requer uma alta capacidade de autonomia em sua tomada de decisão. A literatura afirma que o próximo passo na evolução dos controladores robóticos autônomos é tornar os robôs autoadaptativos. Além disso, os avanços no campo da Aprendizagem de Máquina está aumentando, contribuindo para o surgimento de inúmeras oportunidades para o desenvolvimento de controles inteligentes aplicados aos robôs. No entanto, ainda existem vários desafios a serem enfrentados, por exemplo, a complexidade no desenvolvimento, a necessidade de reimplementação ao mudar aspectos do ambiente, a necessidade de grandes quantidades de dados, propagação de erros, mapeamento de estados complexos em uma única missão de robô, entre outros. A fim de mitigar esses problemas, propomos um Framework, denominado RLoRDE (Reinforcement Learning for Robotic model-Driven Engineering), que se baseia na utilização da Engenharia Dirigida por Modelos para simplificar o desenvolvimento de software robótico, onde o código é gerado de acordo com o modelo criado pelo desenvolvedor seguindo regras impostas pelos metamodelos desenvolvidos durante esta tese. Uma ferramenta gráfica auxilia na criação e transformação dos modelos para o código. Métodos de Aprendizado por Reforço estão disponíveis onde é possível gerar ambientes de treinamento para missões que requerem

flexibilidade para lidar com a variabilidade do ambiente e promover a autoadaptação. Nossos experimentos foram realizados aumentando o grau de complexidade do ambiente para a missão do robô. Os resultados experimentais mostram que o Framework RLoRDE é promissor no sentido de que obtivemos em média 69% de taxa de sucesso na missão em cenários onde o robô não foi treinado.

Palavras-chave: Robótica, Aprendizado de Máquina, Aprendizado por Reforço, Engenharia Dirigida por Modelos.

An Approach Based on Model Driven Engineering and Machine Learning Applied to Mobile Robots

Author: Edson de Araújo Silva

Advisor: Raimundo da Silva Barreto, Dr.

Abstract

The application of mobile robots in complex environments requires a high capacity for autonomy in their decision making. The literature states that the next step in the evolution of autonomous robotic controllers is to make robots self-adaptive. In addition, advances in the field of Machine Learning are increasing, contributing to the emergence of numerous opportunities for the development of intelligent controls applied to robots. However, there are still several challenges to be faced, for example, the complexity in development, the need for reprogramming when changing aspects of the environment, the need for large amounts of data, error propagation, mapping complex states into a single robot, among others. In order to mitigate these problems, we propose a Framework, called RLoRDE (Reinforcement Learning for Robotic model-Driven Engineering), which is based on the use of Model Driven Engineering to simplify the development of robotic software, where the code is generated according to the model created by the developer following rules imposed by the metamodels developed during this thesis. A graphical tool assists in creating and transforming models to code. Reinforcement Learning methods are available where it is possible to generate training environments for missions that require flexibility to deal with the variability of the environment and promote self-adaptation. Our experiments were carried out by increasing the degree of complexity of the environment for the robot's mission. The experimental results show

that the Framework RLoRDE is promising in the sense that we obtained an average 69% mission success rate in scenarios where the robot was not trained.

Keywords: Robotic, Machine Learning, Reinforcement Learning, Model Driven Engineering.

Lista de ilustrações

Figura 1 – Metodologia de pesquisa adotada (adaptado de Spínola, Dias-Neto e Travassos (2008)). 9
Figura 2 – Metodologia da pesquisa – Fase de Concepção (adaptado de Spínola, Dias-Neto e Travassos (2008)). 9
Figura 3 – Resumo dos estudos conduzidos ao longo da pesquisa. 11
Figura 4 – Visão Geral da Abordagem. 13
Figura 5 – Classificação da robótica (RIASCOS, 2010). 15
Figura 6 – Elementos básicos de um sistema de navegação autônoma para robôs móveis (SHALAL et al., 2013). 19
Figura 7 – Camadas da arquitetura de MDE. 22
Figura 8 – Conceitos gerais de MDE. 24
Figura 9 – Abordagem do Aprendizado Supervisionado (RASCHKA, 2015). . . 27
Figura 10 – Exemplo de agrupamento (VASILEV et al., 2019). 28
Figura 11 – Abordagem de Aprendizado por Reforço (VASILEV et al., 2019). . . 29
Figura 12 – Ilustração esquemática da rede neural convolucional (MNIH et al., 2015). 35
Figura 13 – Visão geral esquemática do algoritmo crítico de ator. Adaptado de Grondman et al. (2012). 37
Figura 14 – Metamodelo do Sistema Robótico. 50
Figura 15 – Metamodelo ROS. 53
Figura 16 – Trecho de Código ATL. 55
Figura 17 – Ferramenta gráfica e modelo robótico utilizado no experimento. . . . 57

Figura 18 – Transformação do Modelo do Robô para o Modelo ROS. 58
Figura 19 – Modelo do Robô ROS. 58
Figura 20 – Transformação do Modelo ROS para o Código. 58
Figura 21 – Cenários no Simulador Gazebo. 60
Figura 22 – Cenários Reais. 61
Figura 23 – Total de Recompensas por Episódio. 64
Figura 24 – Taxa de Sucesso da Missão. 64
Figura 25 – Velocidade Média. 65
Figura 26 – Distância Média Percorrida. 66
Figura 27 – Duração Média da Missão. 67
Figura 28 – Trajetória Percorrida nos Testes no Simulador. 68
Figura 29 – Trajetória Percorrida nos Testes no Ambiente Real. 69

Lista de tabelas

Tabela 1 – Comparação de Trabalhos Relacionados 47

Lista de abreviaturas e siglas

ATL	Atlas Transformation Language
DQN	Deep Q-Network
DSL	Domain Specific Language
GPML	General Purpose Modeling Languages
M2M	Model to Model
M2T	Model to Text
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MDP	Markov Decision Process
MOF	Meta-Object Facility
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
RL	Reinforcement Learning
RLoRDE	Reinforcement Learning for Robotic model-Driven Engineering
ROS	Robot Operating System
RSL	Revisão Sistemática da Literatura
UML	Unified Modeling Language

URDF	Unified Robot Description Format
XML	Extensible Markup Language

Lista de algoritmos

1 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$ 33
2 Deep Q-learning com Replay da Experiência 35
3 Advantage Actor-Critic 38
4 Deep Deterministic Policy Gradient Algorithm 39

Sumário

1 INTRODUÇÃO 1

1.1 Contextualização 1

1.2 Motivação 4

1.3 Problema 5

1.4 Objetivos da Pesquisa 7

1.5 Método de Pesquisa 8

1.6 Método Proposto 11

1.7 Organização do Trabalho 13

2 FUNDAMENTAÇÃO TEÓRICA 14

2.1 Robótica 14

2.1.1 Robôs Móveis 16

2.1.1.1 Cinemática 17

2.1.1.2 Navegação 18

2.1.1.3 Arquitetura de Controle para Veículos Autônomos 20

2.2 Model Driven Engineering 21

2.3 Aprendizado de Máquina 26

2.3.1 Aprendizado Supervisionado 26

2.3.2 Aprendizado Não Supervisionado 28

2.3.3 Aprendizado Por Reforço 28

2.3.3.1 Algoritmo Q-Learning 32

2.3.3.2 Algoritmo Deep Q-Network DQN 34

2.3.3.3 Algoritmo Advantage Actor Critic (A2C) 36

2.3.3.4 Algoritmo Deep Deterministic Policy Gradient (DDPG) 38

3 TRABALHOS RELACIONADOS 40
3.1 Abordagens direcionadas as fases de projeto 40
3.2 Abordagens com adaptação em tempo de execução 44

4 MÉTODO PROPOSTO 48
4.1 Aprendizado por Reforço para Robótica com Engenharia Dirigida por Modelos 48
4.1.1 Metamodelo Robótico 49
4.1.2 Metamodelo ROS 52
4.1.3 Transformações 54

5 RESULTADOS EXPERIMENTAIS 56
5.1 Experimento 56
5.1.1 Ambiente de Aprendizagem 59
5.1.2 Representação de Estados e Ações 62
5.1.3 Função de Recompensa 62
5.2 Resultados 63

6 CONCLUSÃO 70
6.1 Considerações Finais 70
6.2 Contribuições da Pesquisa 71
6.3 Limitações 72
6.4 Trabalho Futuro 73

Referências 74

1 Introdução

Este capítulo é destinado a esclarecer o enfoque da pesquisa dando subsídios necessários a compreensão do escopo do trabalho. Desta forma, é realizado uma contextualização do cenário atual na Seção 1.1 em relação a robótica e tecnologias aplicadas neste campo. Logo em seguida são relatadas as motivações para realização da pesquisa na Seção 1.2 e os problemas a serem tratados na Seção 1.3. Na Seção 1.4 são detalhados os objetivos. Também são expostos a metodologia na Seção 1.5 e uma breve apresentação da proposta na Seção 1.6.

1.1 Contextualização

Um robô pode ser definido, de acordo com Oxford (2019), como uma máquina programável por um computador e capaz de executar uma série complexa de ações automaticamente. Os avanços tecnológicos, tanto em hardware, com a miniaturização de componentes e aumento do poder de processamento, bem como em software, em áreas como aprendizagem de máquina, vem proporcionando aos robôs um aumento de suas capacidades, embora limitadas, que vão além de apenas seguir linhas de código programadas.

Diante destes avanços, há uma expectativa do surgimento de novas gerações de dispositivos robóticos que interagem e cooperam com pessoas em ambientes comuns (companheiros robôs, cuidadores de idosos, robôs de serviço, robôs colaboradores), que se integram em ambientes complexos (doméstico, externo, espaços públicos), que se encaixam em diferentes níveis de hierarquias de sistema, que podem cumprir diferentes

tarefas e que são capazes de se adaptar a situações diferentes e condições variáveis (SCHLEGEL; STECK; LOTZ, 2012).

Expandir a autonomia de robôs móveis traz diversos benefícios para sua aplicação em ambientes reais. O poder de decisão e adaptação em tempo de execução diante de situações inesperadas são alguns exemplos desses benefícios. Essas características são importantes porque a maioria dos ambientes de aplicação são dinâmicos e não estruturados.

Mudanças ao longo do tempo, como substituição ou degradação de sensores, atualizações de software ou operação em novos ambientes, afetam como um robô toma decisões, se move, raciocina e interage com seu ambiente. Normalmente, essas mudanças exigem adaptação manual demorada de sistemas robóticos móveis. A autoadaptação de robôs tem sido objetivo de diversos estudos com foco em arquitetura de software (EDWARDS et al., 2009; SYKES et al., 2008; GEORGAS; TAYLOR, 2008; GARCIA et al., 2018), outros utilizando métodos formais (ALDRICH et al., 2019; MUSIL et al., 2017; WEYNS; MALEK; ANDERSSON, 2010) e com o paradigma de Engenharia Dirigida por Modelo (*Model Driven-Engineering - MDE*) (DRAGULE; MEYERS; PELLICCIONE, 2017; DJUKIĆ; POPOVIĆ; TOLVANEN, 2016; NORDMANN; WREDE; STEIL, 2015; STECK; LOTZ; SCHLEGEL, 2011). Há também trabalhos que aproveitam o avanço do aprendizado de máquina, principalmente no uso do aprendizado por reforço (NAGABANDI et al., 2018; JAMSHIDI et al., 2019; YANG et al., 2020; HRABIA; LEHMANN; ALBAYRAK, 2019; MARTINEZ-TENOR et al., 2018). Tornar o software robótico adaptável a essas mudanças pode realmente levar a flexibilidade e a inteligência dos robôs para o próximo nível de aplicação.

Apesar de seus inegáveis benefícios, os sistemas robóticos consistem em diferentes componentes de hardware e software que resultam em uma arquitetura de sistema altamente complexa e variável. Como resultado, usualmente somente especialistas são capazes de montar, configurar e programar esses robôs (ADAM et al., 2016; BUCHMANN et al., 2015; BAUMGARTL et al., 2013). Além disso, os sistemas robóticos são desenvolvidos, na maioria das vezes de forma artesanal, em vez de seguir processos de engenharia bem estabelecidos (DRAGULE; MEYERS; PELLICCIONE, 2017; ESTEVEZ

et al., 2017; STECK; LOTZ; SCHLEGEL, 2011).

Um sistema de engenharia robótico é um esforço complexo que requer soluções de múltiplos domínios, bem como sua integração bem sucedida (ADAM et al., 2016). Sua complexidade aumenta significativamente ao ter que lidar com hardware heterogêneo, principalmente quando se considera a integração de diversos robôs e com diversas abordagens computacionais para resolver problemas funcionais (por exemplo planejamento, percepção e controle) (HOCHGESCHWENDER et al., 2016).

Soma-se a isso o fato de que os robôs que operam em ambientes complexos têm que lidar com muitas situações e contingências diferentes que podem prejudicar seu funcionamento durante a realização da tarefa. Isso pode acontecer pois, em tempo de execução, o robô tem que gerenciar uma enorme quantidade de diferentes variantes de execução, que podem não ter sido previstas ou completamente pré-programadas e, portanto, não podem ser analisadas e verificadas inteiramente nas fases de desenvolvimento (STECK; LOTZ; SCHLEGEL, 2011).

A maioria das missões que um robô enfrenta terá situações imprevisíveis e emergentes durante a execução da missão e, consequentemente, os robôs devem ser resilientes a essas situações imprevisíveis e emergentes (DRAGULE; MEYERS; PELLICCIONE, 2017). Desse modo, há uma necessidade de adaptação em tempo de execução que é claramente evidenciada em nossa Revisão Sistemática da Literatura (RSL) publicada anteriormente (SILVA et al., 2021).

A RSL, realizada durante o andamento do trabalho, evidenciou uma adoção crescente de métodos de MDE aplicados as abordagens de desenvolvimento de software robótico. Com o estudo acerca dessas abordagens foi possível criar uma classificação considerando os conceitos de MDE. Essa classificação conta com 63 abordagens encontradas na literatura. Também demonstramos o avanço da modelagem dentro das fases de desenvolvimento do software robótico além de fazer uma comparação entre as expectativas passadas descritas no roteiro plurianual para robótica 2020 (ROBOTICS, 2016) e os resultados alcançados pelas abordagens analisadas.

Ademais, um processo de engenharia sistemático é essencial para substituir sistemas monolíticos, feitos manualmente, por sistemas compostos de componentes

amadurecidos e reutilizáveis, a fim de diminuir os custos, tempo de colocação no mercado e aumentar a qualidade e robustez (STECK; LOTZ; SCHLEGEL, 2011). Há uma demanda crescente por aplicações cada vez mais complexas e com requisitos como reutilização, flexibilidade e adaptabilidade (ESTEVEZ et al., 2017).

O paradigma de Engenharia Dirigida por Modelos (MDE - *Model Driven Engineering*) ganhou popularidade entre os desenvolvedores de software e várias metodologias, modelos e ferramentas foram desenvolvidos com intuito de facilitar a decomposição de tarefas, permitir a reutilização de software, minimizar erros de codificação e permitir a manutenção de software mais barata (PARASCHOS; SPANOUDAKIS; LAGOUDAKIS, 2012). Os métodos de desenvolvimento de domínio específico orientado por modelo são conhecidos por lidar com os desafios da construção de sistemas heterogêneos complexos em domínios como: aeroespacial, telecomunicações e automotivo, onde enfrentam desafios semelhantes aos da robótica avançada (NORDMANN; WREDE; STEIL, 2015).

1.2 Motivação

MDE é uma tecnologia que atingiu um nível maduro em outros campos, mas ainda não na robótica (CICCOZZI et al., 2016). O conceito de MDE é particularmente adequado para o gerenciamento da variabilidade, que é uma característica típica dos sistemas robóticos. Os modelos são definidos como artefatos de desenvolvimento primários. Os modelos podem ser mais compreensíveis, independentes de plataforma, e podem ser traduzidos em implementações diferentes (RINGERT; RUMPE; WORTMANN, 2015).

Com MDE, o conceito de metamodelos e linguagens específicas de domínio são usados para suportar a geração de código (ou outros artefatos) a partir de modelos abstratos que descrevem um domínio em particular. Com isso e através da separação sistemática de preocupações, a qualidade do software é aprimorada (HOCHGESCHWENDER et al., 2016).

A transição da engenharia orientada por código para a engenharia baseada em modelos na robótica é uma das necessidades básicas para alcançar separação de funções

e gerenciar decisões em tempo de execução (STECK; LOTZ; SCHLEGEL, 2011). Deste modo, dispor de um conjunto de metamodelos capazes de representar características peculiares do aprendizado por reforço na robótica agregará maior qualidade e robustez ao software desenvolvido, além de diminuir tempo e custos.

Trabalhos que empregam o paradigma MDE tiveram inúmeras contribuições para a robótica conforme apresentado em nossa pesquisa (SILVA et al., 2021). Através da abstração em diferentes níveis, engenheiros de software robóticos podem trabalhar sistematicamente, melhorando a qualidade dos sistemas em termos de segurança, confiabilidade, reusabilidade, reduzindo variabilidade e complexidade, promovendo o reuso de componentes de software e hardware. A MDE enfatiza fortemente o desenvolvimento de modelos de software de alto nível em vez da simples geração de código-fonte. Os modelos são considerados fundamentais neste paradigma por possuírem sintaxe e semântica bem definidas, sendo não apenas aplicados à documentação, mas também contrários às diretrizes informais de programação do sistema. Além disso, usando geradores de código, o MDE desenvolve modelos executáveis transformando modelos em modelos ou modelos em código-fonte (BAUMGARTL et al., 2013).

1.3 Problema

Nos últimos anos diversos softwares robóticos foram desenvolvidos, porém dificilmente podem interoperar uns com os outros porque suas dependências em hardware, ou plataforma de software específicos, estão inseridas no código o que, em caso de alguma mudança, implica várias alterações demoradas no código. Além disso, o software robótico é difícil de se adaptar às mudanças na plataforma de destino, ou seja, ao ambiente inserido (STECK; LOTZ; SCHLEGEL, 2011). Consequentemente, isso torna o software robótico difícil e caro de se desenvolver, porque há pouca oportunidade de reutilização.

Algumas características podem afetar de forma negativa a confiabilidade no sistema, como por exemplo, as falhas introduzidas no sistema durante o desenvolvimento, além de problemas inesperados ou não observados na interação com o ambiente

durante a operação (MÜHLBACHER et al., 2016).

O Roteiro Plurianual para a Robótica na Europa 2020 afirma claramente que: "geralmente não há processos de desenvolvimento de sistema, destacados pela falta de modelos e métodos gerais de arquitetura, o que resulta na necessidade de habilidade na construção de sistemas robóticos em vez de seguir processos de engenharia estabelecidos" (ROBOTICS, 2016).

A comunidade de roboticistas que utilizam o paradigma orientado por modelos ainda não possui uma reutilização significativa de modelos, linguagens e ferramentas específicas de domínio, o que leva a uma enorme quantidade de reimplementação para os mesmos aspectos e alta fragmentação (WIGAND et al., 2017). Deste modo, engenheiros de software exigem métodos e ferramentas significativamente aprimorados para desenvolver e manter aplicativos robóticos de maior qualidade (DJUKIĆ; POPOVIĆ; TOLVANEN, 2016).

Embora o avanço da MDE tenha feito contribuições significativas para a robótica, os *Frameworks* atuais não atendem (SILVA et al., 2021) ou quase não atendem (DRAGULE; MEYERS; PELLICCIONE, 2017) as necessidades de autoadaptação para seu uso em ambientes que possuem alto grau de variabilidade. Os principais desafios da utilização dessa abordagem são: (i) a dificuldade em extrair conceitos heterogêneos, abstrair a tecnologia e propor conceitos mais próximos do domínio de aplicação; (ii) usabilidade dos modelos; (iii) definição de padrões no desenvolvimento de software robótico; e (iv) escalabilidade e modularidade.

Por outro lado, no campo do Aprendizado por Reforço (*Reinforcement Learning - RL*), os sistemas robóticos aprendem a partir de suas interações com o ambiente, formando uma percepção e agindo sobre essa percepção. O uso de aprendizado por reforço, em vez de um controlador convencional, permite que o sistema descubra possibilidades que de outra forma não seriam antecipadas (DOORAKI; LEE, 2021). Isso garante ao sistema robótico uma seleção dinâmica das ações mais adequadas e permite que o sistema se adapte às mudanças no ambiente e ao seu próprio estado (HRABIA; LEHMANN; ALBAYRAK, 2019). Além disso, avanços recentes em aprendizado profundo demonstraram fortes capacidades de generalização, bem como a capacidade

de aprender recursos relevantes para seu ambiente de aplicação (JEONG et al., 2020). Isso pode ser de grande contribuição para diversas áreas dentro da robótica como navegação visual (SHANTIA et al., 2021), direção autônoma (LIKMETA et al., 2020), UAVs (Unmanned Aerial Vehicles) (YAN; XIANG; WANG, 2020) e robôs de serviço (JAMSHIDI et al., 2019).

Embora o uso de RL traga diversos benefícios, ainda há grandes desafios a serem enfrentados, dentre os quais podemos citar: (i) a necessidade de uma quantidade significativa de dados, o que pode ser um desafio crítico para a robótica (KALASHNIKOV et al., 2018); (ii) a abordagem é propensa a se propagar e até aumentar os riscos, como gerar atrasos e sinais ruidosos do sensor (YANG et al., 2020); (iii) a capacidade do robô de mapear estados complexos ainda está em estudo (SHANTIA et al., 2021), por exemplo, entradas de imagens de câmeras, em estimativas de valor significativas para ações; e (iv) o treinamento do robô é destinado a tarefas específicas e únicas. Portanto, acreditamos que fornecer aos robôs a capacidade de auto-adaptação às mudanças do ambiente em que o robô se encontra pode trazer ganhos tanto em termos de autonomia do robô quanto na diversificação de aplicações robóticas.

Desta forma, o problema abordado nesta tese pode ser expresso através da seguinte questão: *agregar aspectos de aprendizado por reforço em uma família de metamodelos de desenvolvimento de software robótico para que o robô possa utilizar o conhecimento adquirido proporcionará uma melhor adaptação ao ambiente inserido?*

1.4 Objetivos da Pesquisa

O objetivo principal desta pesquisa é *propor uma família de metamodelos capaz de agregar conceitos de aprendizado por reforço em modelos de desenvolvimento de software robótico, de tal forma que as ações do sistema robótico sejam, em tempo de execução, ajustadas às necessidades de interação do robô com o ambiente.*

Os objetivos específicos são:

1. Simplificar o desenvolvimento do software robótico através da geração e uso de modelos de alto-nível, e que tais modelos possam ser utilizados para a geração de

código automático;

2. Demonstrar que o uso de aprendizagem por reforço é uma estratégia útil para resolver o problema da autoadaptação em sistemas robóticos;

3. Investigar a integração de técnicas de aprendizagem por reforço e métodos baseados em modelos;

4. Desenvolver um *Framework* para geração de ambientes de treinamento para robôs móveis com aprendizado por reforço;

5. Avaliar experimentalmente em ambiente simulado a execução de tarefas pré-determinadas com adição de novas características de ambientes dinâmicos; e

6. Avaliar experimentalmente em ambiente real a execução de tarefas pré-determinadas afim de validar os resultados obtidos por meio da avaliação em ambiente simulado.

1.5 Método de Pesquisa

O método de pesquisa adotado neste trabalho está fundamentado nos princípios da Engenharia de Software Experimental que se baseia na condução de estudos primários e secundários em fases diferentes da investigação (SPÍNOLA; DIAS-NETO; TRAVASSOS, 2008). Ela se divide em duas fases: concepção e avaliação do modelo proposto, conforme descrito na Figura 1. A pesquisa será conduzida até a etapa do Estudo de Caso.

A **Fase de Concepção do Modelo** envolve alguns passos e a execução de estudos secundários e/ou primários com o objetivo de se obter uma proposta inicial do modelo como mostra a Figura 2.

- **Revisão Informal (RI)**: trata-se do meio para estudar e descrever uma pesquisa que servirá de apoio para outros estudos, através de alguns resultados relevantes. A estratégia tem como objetivo fornecer subsídios teóricos acerca dos assuntos en-

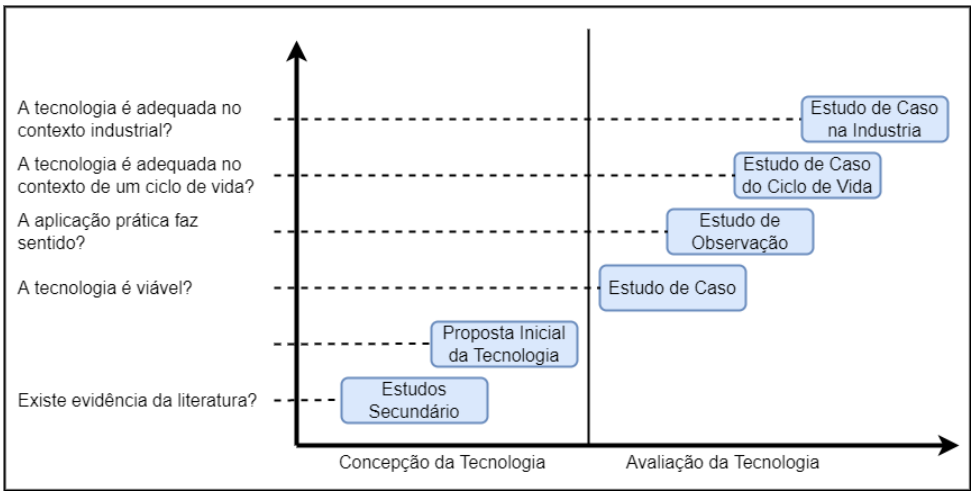


Figura 1 – Metodologia de pesquisa adotada (adaptado de Spínola, Dias-Neto e Travassos (2008)).

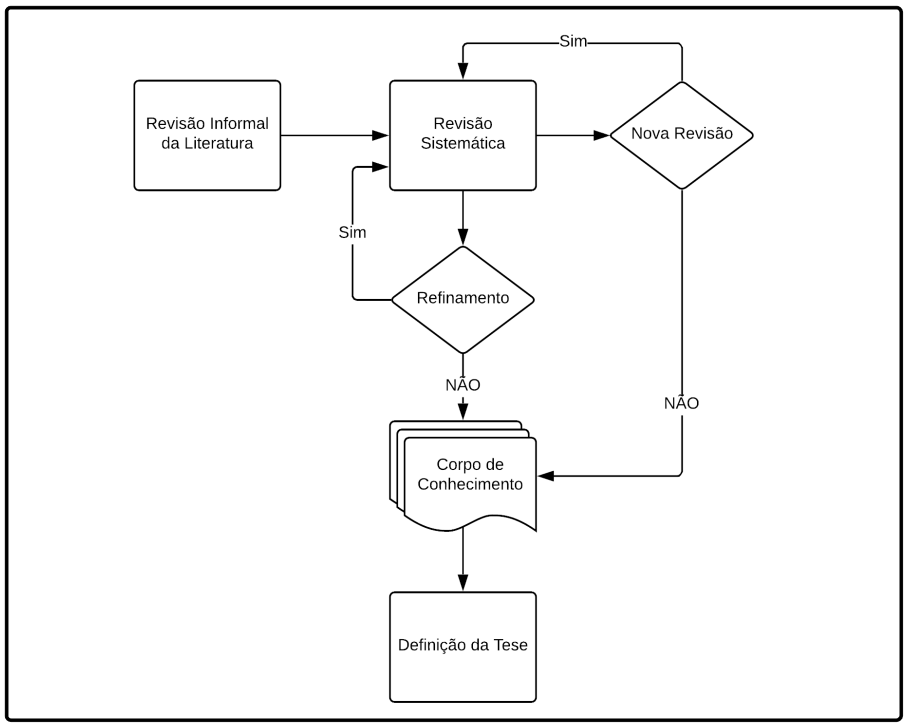


Figura 2 – Metodologia da pesquisa – Fase de Concepção (adaptado de Spínola, Dias-Neto e Travassos (2008)).

volvidos e identificar os conceitos básicos para apoiar a definição de um protocolo de revisão sistemática mais preciso e abrangente.

- **Revisão Sistemática da Literatura (RSL)**: processo formal definido para alcançar resultados com valor científico e apresentar um resultado justo de um tópico de pesquisa, usando uma metodologia confiável e rigorosa. A RSL foi baseada no *guidelines* desenvolvido por (KITCHENHAM; CHARTERS, 2007) definido em três etapas: (i) Planejamento do Mapeamento, (ii) Condução do Mapeamento e (iii) Resultados da Revisão.

- **Corpo de Conhecimento**: caracterizado pelo resultado obtido através da revisão sistemática sobre Engenharia Dirigida por Modelo aplicado a Robótica, além dos conceitos de Aprendizado por Reforço.

- **Definição da Tese**: com base no corpo de conhecimento adquirido na etapa anterior, o modelo é definido por meio da junção das técnicas de MDE, Robótica e Aprendizado por Reforço em Metamodelo formal.

Concluída a fase de concepção, foi realizada a etapa de **avaliação da abordagem proposta (*Framework*)**. Para esta fase, foram realizados dois experimentos de pesquisa:

- **Estudo de Caso**: nesta etapa o *Framework* proposto foi aplicado a um caso de uso para avaliação do comportamento do robô em um ambiente simulado utilizando as métricas propostas por (LAMPE; CHATILA, 2006) para avaliar o desempenho do robô. As métricas para a missão de navegação foram a velocidade média percorrida, distância, duração da missão e taxa de sucesso da missão.

- **Estudo de Caso Real**: nesta etapa o *Framework* proposto foi aplicado a um caso de uso para avaliar o comportamento do robô em um ambiente real para validar os resultados alcançados no ambiente simulado e realizar um estudo comparativo.

A Figura 3 apresenta um resumo das etapas que compõem a metodologia de pesquisa adotada neste trabalho, assim como os tipos de estudos conduzidos e os resultados alcançados.

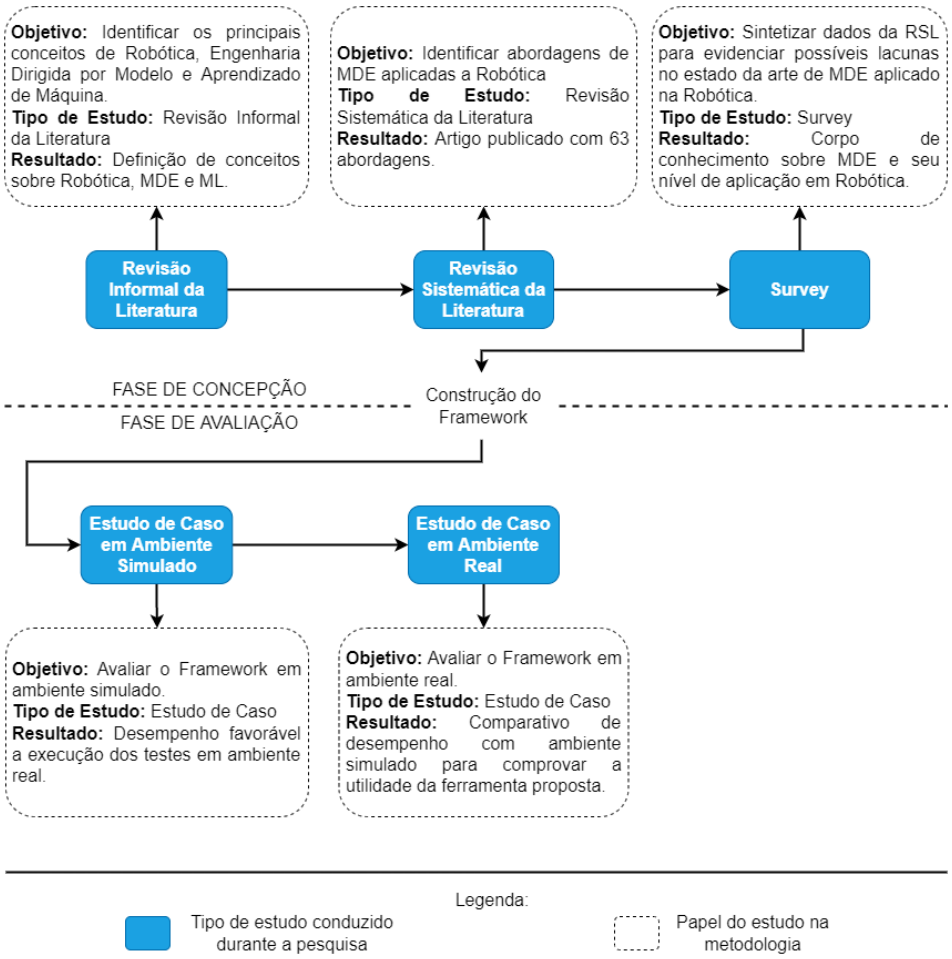


Figura 3 – Resumo dos estudos conduzidos ao longo da pesquisa.

1.6 Método Proposto

Diante dos problemas descritos na Seção 1.3 dos quais vale lembrar: ambientes complexos com alta variabilidade; falta de confiabilidade; falta de reusabilidade de componentes de software; adaptação em tempo de execução; e falta de métodos e ferramentas que proporcionem maior qualidade, nesta tese desenvolvemos um *Framework* com a capacidade de geração de código e utilização de aprendizado por reforço para inserir nos modelos gerados capacidades de autoadaptação em tempo de execução.

A complexidade de ambiente dinâmicos é caracterizada pela variabilidade do ambiente e aumenta significativamente ao ter que lidar com hardware heterogêneo e ao integrar diversas abordagens computacionais para resolução de problemas como: planejamento, percepção e controle. Essas abordagens possuem características únicas que as diferenciam umas das outras e proporcionam um melhor resultado em determinadas

situações.

Para lidar com essa complexidade de ambientes dinâmicos, apresentamos o *Framework* denominado RLoRDE (*Reinforcement Learning for Robotic model-Driven Engineering*), desenvolvido para atender as necessidades de implementação de ambientes de treinamento para robôs guiados por aprendizado de máquina. Este *Framework* suporta diferentes arquiteturas robóticas e possui três tipos de controladores baseados em aprendizado por reforço. Além disso, uma das principais características do *Framework* é a possibilidade de criar pontos de variação no controle robótico, permitindo a utilização de vários controladores ao mesmo tempo. A utilização desses pontos de variação permite o uso de controladores tradicionais em conjunto com controladores baseados em aprendizado de máquina para mitigar alguns dos problemas citados anteriormente, alguns dos quais incluem a necessidade de auto-adaptação, a necessidade de uma enorme quantidade de dados para treinar e mapear estados complexos.

A Figura 4 mostra uma visão geral da abordagem proposta. Para ilustrar o funcionamento do *Framework*, imagine a necessidade de construir um controlador para um carro autônomo com a tarefa de estacionar. Utilizando o RLoRDE o roboticista irá utilizar a ferramenta de modelagem para criar o modelo com a arquitetura e os sensores que serão utilizados no cenário. É necessário escolher qual algoritmo RL será utilizado no treinamento e se será necessário mais de um, caso o problema possua pontos de variação. Neste caso podemos definir os tipos de estacionamento, fila indiana, 45 graus e perpendicular a guia da calçada, como pontos de variação no modelo. Após o modelo criado é possível realizar a primeira etapa para geração do código, que consiste na transformação do modelo robótico para o modelo exigido no *middleware* ROS (*Robot Operating System*). A segunda etapa é a geração do código que implica na geração do robô simulado no simulador Gazebo, o ambiente de treinamento GYM da OPENAI e os pacotes do ROS. Antes de executar o treinamento também será necessário a criação do ambiente tridimensional (mundo 3D) onde o robô realizará o aprendizado. A criação do cenário 3D não é suportada pela ferramenta.

Ao treinar o dispositivo robótico em um ambiente simulado, a rede neural tende a maximizar as ações para atender as necessidades de resolução do problema. A

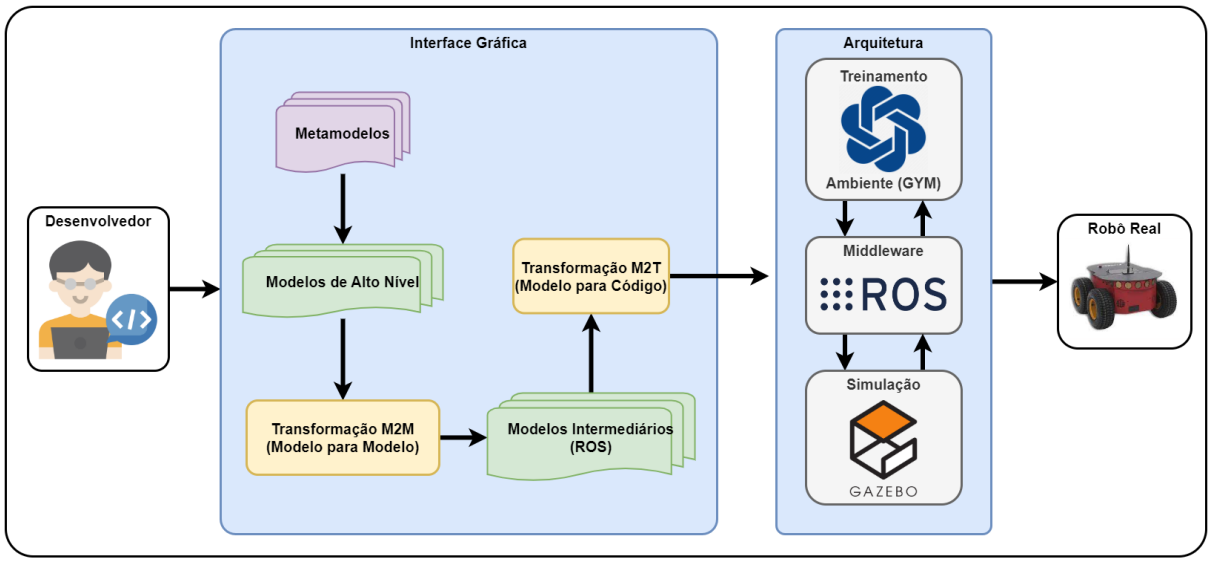


Figura 4 – Visão Geral da Abordagem.

percepção do ambiente e sua execução baseada em tentativa e erro, característico do uso de RL, poderá compor um controlador capaz de lidar com algumas peculiaridades do ambiente que não estavam previstas. Ao complementar isto com regras para utilização dos pontos de variação, é possível dar mais segurança e flexibilidade para o robô. Desta forma busca-se alcançar a autoadaptação em tempo de execução treinando o agente em ambiente simulado para aperfeiçoamento de suas habilidades.

1.7 Organização do Trabalho

O restante desta Tese está organizada da seguinte forma. O Capítulo 2 apresenta os principais conceitos para o entendimento do texto como fundamentos de robótica, engenharia dirigida por modelo e aprendizado de máquina. O Capítulo 3 apresenta os trabalhos que mais se aprofundaram na área de engenharia dirigida por modelos aplicado a robótica e com aprendizado de máquina. O Capítulo 4 detalha o método proposto em busca de uma possível solução. Os experimentos e resultados alcançados e a análise a respeito do que foi observado são apresentados no Capítulo 5. Por fim, o Capítulo 6 apresenta as considerações finais, limitações da proposta e trabalhos futuros.

2

Fundamentação Teórica

Neste Capítulo, abordamos os principais termos, conceitos e definições utilizados no texto. A Seção 2.1 abrange conceitos fundamentais de robótica e trata da subárea de robôs móveis que foi alvo da investigação nesta pesquisa. A Seção 2.2 detalha conceitos de MDE que se consolidaram como arcabouço de abordagens de desenvolvimento por meio de modelos. A Seção 2.3 apresenta conceitos relevantes para área de aprendizado por reforço e os algoritmos utilizados na abordagem proposta.

2.1 Robótica

O progresso da sociedade se deve em grande parte pela evolução da tecnologia desde a revolução industrial. O estilo de vida moderno foi proporcionado pelo avanço em áreas como Tecnologia de Informação (TI). O termo TI, cunhado pela primeira vez em 1958 vem crescendo desde o início dos anos 90, e tem por definição a aplicação de computadores e equipamentos de telecomunicações para armazenar, recuperar, transmitir e manipular dados (LI et al., 2014).

Agora, estamos diante de um novo desafio tecnológico, que é como armazenar, recuperar, manipular e gerenciar um grande volume de dados, para gerar novos conhecimentos de forma autônoma e inteligente. A robótica ganhou força nos últimos anos e já conta com inúmeros casos de inclusão nos mais variados nichos de mercado e na sociedade como um todo. Só para se ter uma ideia, hoje já temos robôs, ainda em fase de testes, que fazem entregas para Amazon (AGRELA, 2019) e para a fabricante de

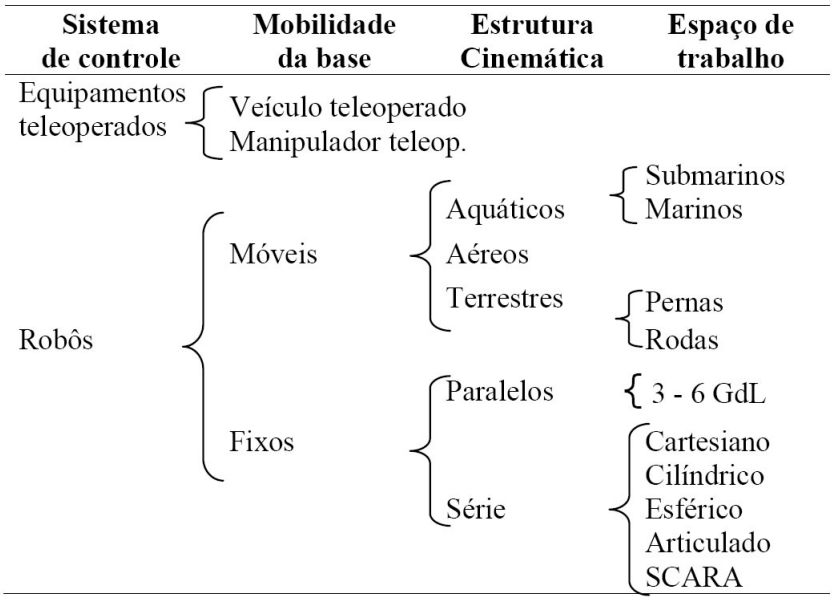


Figura 5 – Classificação da robótica (RIASCOS, 2010).

peças alemã Continental (GUSMãO, 2019), robôs domésticos produzidos pela Toyota motors (BUCKLAND, 2019), robôs garçons idealizados pela maior rede de restaurantes de capital aberto da Ásia, a Haidilao International Holding (DU, 2019). Além dos carros autônomos, fabricados por empresas de ponta nessa área de pesquisa como Uber, Google e Tesla.

Robótica é um ramo da engenharia que envolve a concepção, projeto, fabricação e operação de robôs. Este campo se sobrepõe à eletrônica, ciência da computação, inteligência artificial, mecatrônica, nanotecnologia e bioengenharia (ARREGUIN, 2008).

Os robôs podem ser classificados (veja a Figura 5) utilizando vários critérios tais como: autonomia do sistema de controle, mobilidade da base, estrutura cinemática, forma de acionamento, graus de liberdade, geometria do espaço de trabalho.

Este trabalho concentra-se na área de robôs móveis e os experimentos foram realizados usando um robô terrestre com a cinemática do tipo esteira, embora a ferramenta, proveniente deste trabalho, dê suporte a outras classes de robôs.

2.1.1 Robôs Móveis

Uma das características fundamentais para os Robôs Móveis Autônomos (RMA) é sua capacidade de locomoção e operação de modo semi ou completamente autônomo. Sua capacidade de autonomia pode ser elevada a medida que aspectos relevantes para sua interação com o ambiente sejam levados em consideração como: capacidade de percepção (sensores), capacidade de agir (atuadores para produção das ações) e a robustez e inteligência (capacidade de lidar com as mais diversas situações, de modo a resolver e executar tarefas por mais complexas que sejam) (ROMERO et al., 2014).

Dentre as características citadas anteriormente, a robustez e inteligência dos robôs depende dos controladores envolvidos na execução de suas tarefas diante da missão a ser realizada. Conforme Medeiros (1998), um sistema de controle de um robô com certo grau de autonomia e complexidade é necessário atender a algumas especificações de comportamento e requisitos de projeto:

- Reatividade ao meio ambiente - O veículo deve ser reativo a mudanças bruscas no ambiente e capaz de levar em consideração eventos externos com prazos compatíveis com a execução correta, eficiente e segura de suas tarefas.

- Comportamento inteligente - Isso requer que diferentes compromissos sejam feitos com base em regras de bom senso para exibir um comportamento inteligente. As reações do robô a estímulos externos devem ser guiadas pelos objetivos de sua tarefa principal.

- Integração de vários sensores - A precisão, confiabilidade e aplicabilidade limitadas de sensores individuais devem ser compensadas pela integração de vários sensores complementares.

- Resolução de objetivos múltiplos - No caso de robôs móveis, situações que exigem ações concorrentes conflitantes são inevitáveis. O sistema de controle deve fornecer meios para atender a esses múltiplos objetivos.

- Robustez - O robô deve lidar com entradas imperfeitas, eventos inesperados e mau funcionamento repentino.

- Confiabilidade - A capacidade de operar sem falhas ou degradação de desempenho durante um determinado período.
- Programabilidade - Um robô útil deve ser capaz de realizar várias tarefas descritas em algum nível de abstração, em vez de apenas uma tarefa precisa.
- Modularidade - O sistema de controle de veículos autônomos deve ser dividido em subsistemas (ou módulos) menores que podem ser projetados, implementados, depurados e mantidos de forma separada e incremental.
- Flexibilidade - A robótica experimental requer mudanças contínuas no projeto durante a fase de implementação. Portanto, estruturas de controle flexíveis são necessárias para permitir que o projeto seja guiado pelo sucesso ou falha dos elementos individuais.
- Expansibilidade - É necessário muito tempo para projetar, construir e testar os componentes individuais de um robô. Portanto, uma arquitetura expansível é desejável para poder construir o sistema de forma incremental.
- Adaptabilidade - Como o estado do mundo muda muito rapidamente e de forma imprevisível, o sistema de controle deve ser adaptável para alternar suave e rapidamente entre diferentes estratégias de controle.
- Raciocínio global - Um agente de decisão global de alto nível, responsável pela compreensão da situação global, é obrigado a dar conta dos erros introduzidos pela má interpretação dos dados sensoriais e a fundir as informações parciais disponíveis.

2.1.1.1 Cinemática

A cinemática de robôs móveis é o estudo da posição, velocidade e aceleração de robôs móveis no espaço tridimensional. Ela é importante para a compreensão e o projeto de robôs móveis, pois permite que sejam desenvolvidos algoritmos de controle para mover o robô de um ponto a outro de forma segura e precisa (ROMERO et al., 2014).

Existem vários modelos de cinemática para robôs móveis, incluindo modelos de rodas diferenciais, modelos de esferas e modelos de patas. Cada modelo tem suas próprias vantagens e desvantagens, e é importante escolher o modelo mais adequado para a aplicação específica.

Os modelos de rodas diferenciais são os mais comuns em robôs móveis, eles consistem em duas ou mais rodas que são controladas de forma independente para permitir que o robô gire sobre seu próprio eixo. Já os modelos de esferas são usados em robôs com uma esfera como base de suporte, eles são úteis para robôs que precisam se mover em qualquer direção sem a necessidade de mudar de orientação. Por fim, os modelos de patas são usados em robôs que precisam subir escadas ou transpor obstáculos (SANZ, 2009).

A cinemática também é importante para a geração de trajetórias, onde se usa algoritmos para planejar o movimento do robô de forma segura e precisa.

2.1.1.2 Navegação

A navegação é um dos maiores desafios nos robôs móveis autônomos. Um sistema de navegação é o método para guiar um veículo. Há vários recursos necessários para a navegação autônoma, como apresentado por ALI e MAHMOUD (2003):

- Capacidade de executar ações elementares, como alcançar um determinado local ou seguir um líder;
- Capacidade de reagir a eventos inesperados em tempo real, como evitar um obstáculo que apareça de repente;
- Capacidade de formular um mapa do ambiente;
- Capacidade de aprender, que pode incluir a observação da localização de um obstáculo, da natureza do terreno e adaptar o torque de acionamento à inclinação (GOLNAZARIAN; HALL, 2000).

Sistemas autônomos inteligentes que detenham a habilidade de navegação são capazes de conduzir um robô móvel até seu alvo, através de uma trajetória isenta de

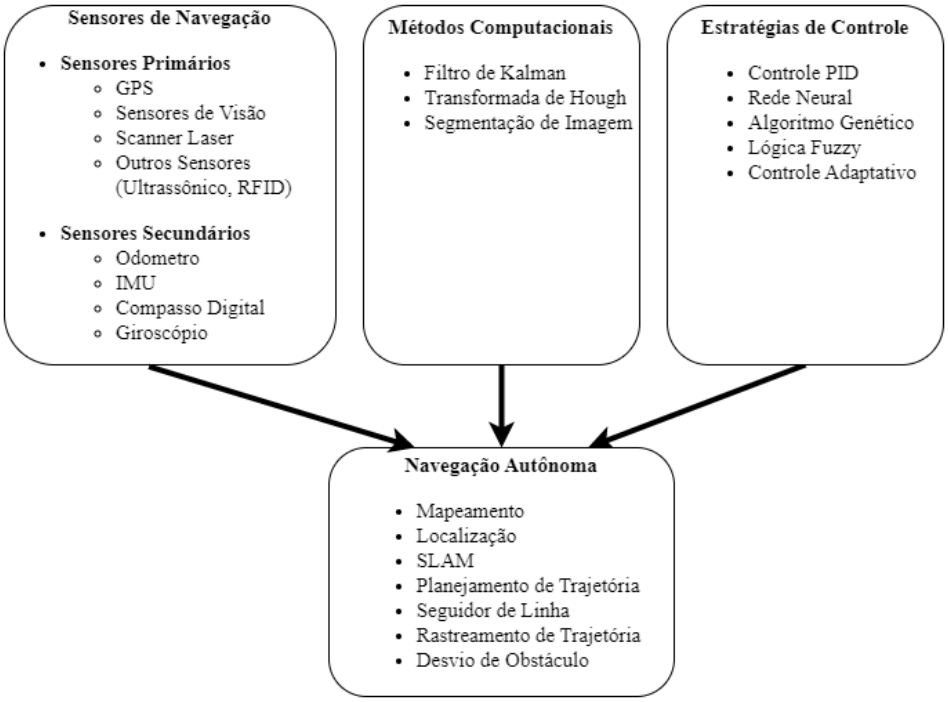


Figura 6 – Elementos básicos de um sistema de navegação autônoma para robôs móveis (SHALAL et al., 2013).

colisões. Tais sistemas de navegação representam internamente estratégias de navegação que definem adequadamente as ações do robô para cada uma das inúmeras situações distintas que se apresentam ao longo da trajetória. Devido à variabilidade de situações, sendo algumas inéditas, o desempenho de um sistema de navegação depende de sua capacidade de aprendizagem. Ou seja, somente aperfeiçoando sua estratégia de navegação, através da aprendizagem, um sistema de navegação torna-se apto para guiar eficientemente o robô em ambientes diversos (associados a diferentes graus de complexidade) e sujeitos a perturbações variadas (CAZANGI; FIGUEIREDO, 2001).

A arquitetura para robôs móveis consiste em sensores para navegação do robô, métodos computacionais e estratégias de controle de navegação. A Figura 6 resume os elementos básicos do sistema de navegação autônoma de robôs móveis. Nesta pesquisa buscou-se dar ênfase nas estratégias de controle para adaptação do agente robótico em tempo de execução. Para isso foi utilizada as redes neurais artificias para alcançar a adaptação aos cenários.

2.1.1.3 Arquitetura de Controle para Veículos Autônomos

Existem diversas arquiteturas de controle de veículos autônomos com diferentes enfoques e abordagens (MEDEIROS, 1998; DUDEK; JENKIN, 2010; BEKEY, 2005). No entanto, algumas tornaram-se mais conhecidas pelas suas características e potencialidades. Dentre elas estão o controle reativo, deliberativo, hierárquico, híbrido e robusto e inteligente (ROMERO et al., 2014). A seguir é detalhado o uso do controle reativo que foi a abordagem adotada na realização dos experimentos.

As estratégias de navegação de robôs móveis podem ser geralmente classificadas em duas categorias, planejamento de caminho global e navegação reativa local. A navegação reativa local emprega algumas estratégias para perceber o ambiente com base na informação sensorial de forma online. O robô deve adquirir um conjunto de mecanismos de estímulo-ação por meio de suas entradas sensoriais, como informações de distância de sensores infravermelhos, informações visuais de câmeras ou dados processados derivados após a fusão apropriada de várias saídas de sensores (LUH; LIU; LIN, 2008).

O controle reativo é baseado em regras simples, chamadas "regras de comportamento", que são usadas para controlar o comportamento do robô. Essas regras são definidas com base nas informações sensoriais do robô e na interação com o ambiente, e são aplicadas de forma independente ao planejamento de trajetórias (ARKIN; ARKIN et al., 1998).

Uma das principais vantagens do controle reativo é sua capacidade de lidar com incertezas no ambiente, como obstáculos inesperados ou mudanças na posição do robô. Além disso, ele é capaz de responder rapidamente a mudanças no ambiente, o que é importante em aplicações como robótica de segurança ou robótica de resgate (FONG; NOURBAKHSH; DAUTENHAHN, 2003).

Existem vários algoritmos de controle reativo diferentes, incluindo o controle reativo baseado em estados, o controle reativo baseado em eventos e o controle reativo baseado em híbrido. Cada um desses algoritmos tem suas próprias vantagens e desvantagens, e é importante escolher o algoritmo mais adequado para a aplicação específica (ALAMI; CHATILA, 1994).

Com o avanço do aprendizado de máquina, as redes neurais tem ganhado espaço de destaque no controle de agentes robóticos. O controle robótico reativo com redes neurais é uma abordagem que combina os princípios do controle robótico reativo com as técnicas de aprendizado de máquina baseadas em redes neurais (ZOU et al., 2006). A ideia é utilizar redes neurais para modelar a dinâmica do robô e do ambiente, e usar esses modelos para gerar ações de controle.

Existem várias maneiras de se implementar o controle robótico reativo com redes neurais, incluindo o uso de redes neurais feedforward, redes neurais recorrentes e redes neurais de reforço. Cada uma dessas abordagens tem suas próprias vantagens e desvantagens, e é importante escolher a abordagem mais adequada para a aplicação específica. Este trabalho adotou a abordagem de aprendizado por reforço como núcleo dos controladores gerados com o uso da ferramenta. Uma vantagem do controle robótico reativo é a capacidade de aprender e adaptar-se ao ambiente, o que é importante em aplicações onde o ambiente é imprevisível ou dinâmico (LEWIS, 1996).

2.2 Model Driven Engineering

A MDE surgiu como uma mudança de paradigma do desenvolvimento de software baseado em código para o desenvolvimento baseado em modelo (STAHL; VOELTER; CZARNECKI, 2006). Tal abordagem promove a sistematização e automação da construção de artefatos de software. Os modelos representam parte da funcionalidade, estrutura e comportamento de um sistema (DEELSTRA et al., 2003). Um metamodelo inclui o conjunto de conceitos necessários para descrever um domínio em um determinado nível de abstração, juntamente com os relacionamentos entre eles. O modelo é definido em termos de metamodelos formais. As transformações de modelo, comumente descritas como mapeamentos de metamodelo, permitem a transformação e evolução automáticas de modelos em outros modelos, definidos em qualquer nível de abstração ou em qualquer formato textual, como código (MENS; GORP, 2006).

Essa abordagem ajuda os especialistas a mudar seu foco do espaço de implementação para o espaço do problema (RAMASWAMY; MONSUEZ; TAPUS, 2014). Com o

aumento da camada de abstração, fornecido pelo paradigma baseado em modelo, é possível descrever aplicativos independentemente da plataforma de software e hardware. Modelos diferentes podem representar elementos do sistema em diferentes visualizações de domínio e seus relacionamentos (ESTÉVEZ et al., 2014). A Figura 7 mostra que existem pelo menos quatro níveis de abstração para integrar a modelagem na evolução do software (DÍAZ et al., 2014).

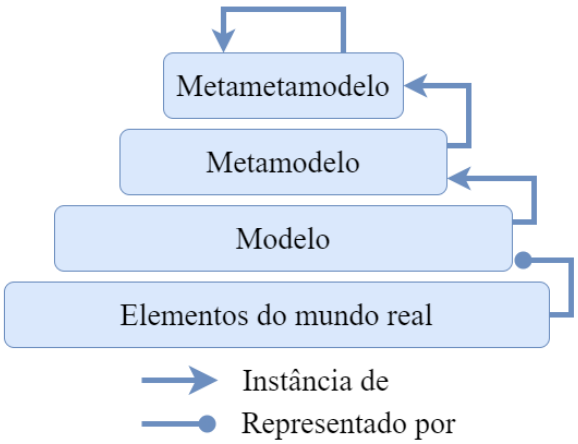


Figura 7 – Camadas da arquitetura de MDE.

- Nível M3 (meta-metamodelo). Para os metamodelos (nível M2) serem reutilizáveis, interoperáveis e portáteis, deve haver outro metamodelo em um nível mais alto de abstração, descrevendo exclusivamente os conceitos usados para representar qualquer metamodelo em qualquer domínio. Portanto, o meta-metamodelo (nível M3) define os conceitos, atributos e relacionamentos para os elementos no nível M2, enquanto os elementos colocados no nível M2 são as instâncias dos elementos no nível M3. Nesse nível, o OMG (Object Management Group) define o Meta-Object Facility (MOF) como a linguagem usada para descrever todos os elementos no nível M2 (OMG, 2011). O MOF é a raiz padrão na maioria dos desenvolvimentos baseados em modelos (DÍAZ et al., 2014).

- Nível M2 (Metamodelo). Nesse nível, os elementos do modelo vêm do nível M1. No caso de um metamodelo como UML (*Unified Modeling Language*) (UML; MOF, 2010), é possível especificar conceitos, como "Classe", "Atributo", "Método" ou "Associação" entre classes. O nível M2 define os elementos válidos em um modelo

específico no nível M1, enquanto os elementos colocados no nível M1 são as instâncias dos elementos no nível M2. O trabalho de Brugali e Gherardi (2016) propôs a abordagem HyperFlex, que é outro exemplo de metamodelo, onde foi aplicado no contexto do desenvolvimento de sistemas robóticos baseados em ROS. Este metamodelo impõe elementos como "*node*" e "*topic*" da arquitetura ROS.

- Nível M1 (modelo). Este nível é uma instância do nível M2 e define as instâncias do respectivo metamodelo. Por exemplo, se o metamodelo é UML, este nível tem que definir conceitos de classe como "Cliente", "Compra" e "Livro" e seus atributos como "Direção", "Nome", "Número", "Título", entre outros. O mesmo ocorre para modelos de caso de uso, ou modelos de sequência, dependendo das características do metamodelo. No caso do HyperFlex, ele deve definir conceitos como Nós e Tópicos que são característicos da plataforma ROS.

- Nível M0 (real). Este nível é responsável por organizar os elementos que representam o mundo real de acordo com o modelo (nível M1), como o objeto "carro" é instanciado usando uma linguagem de programação.

De acordo com o nível de abstração, o MDE requer um domínio fixo, delimitado por um campo específico de conhecimento. Esse domínio pode ser, por exemplo, o campo da robótica que, por sua vez, pode ser especializado em subdomínios, como a robótica aérea ou terrestre. A Figura 8 mostra os conceitos mais importantes relacionados ao MDE.

A Figura 8 estende a Figura 7 no sentido de que detalha os principais conceitos do MDE. Os metamodelos consistem em uma sintaxe abstrata e uma semântica estática. A sintaxe abstrata especifica a estrutura de um metamodelo, ou seja, construtores, propriedades e conectores (DÍAZ et al., 2014). Em outras palavras, é a abstração de conhecimento do domínio. A semântica estática enfoca a definição de regras de relacionamento entre seus elementos.

A sintaxe concreta de uma linguagem é usada para definir a notação específica usada pelos usuários. Ele define como os usuários aprenderão e usarão, seja lendo ou escrevendo e desenhando os modelos (SILVA, 2015). Idealmente, cada conceito de

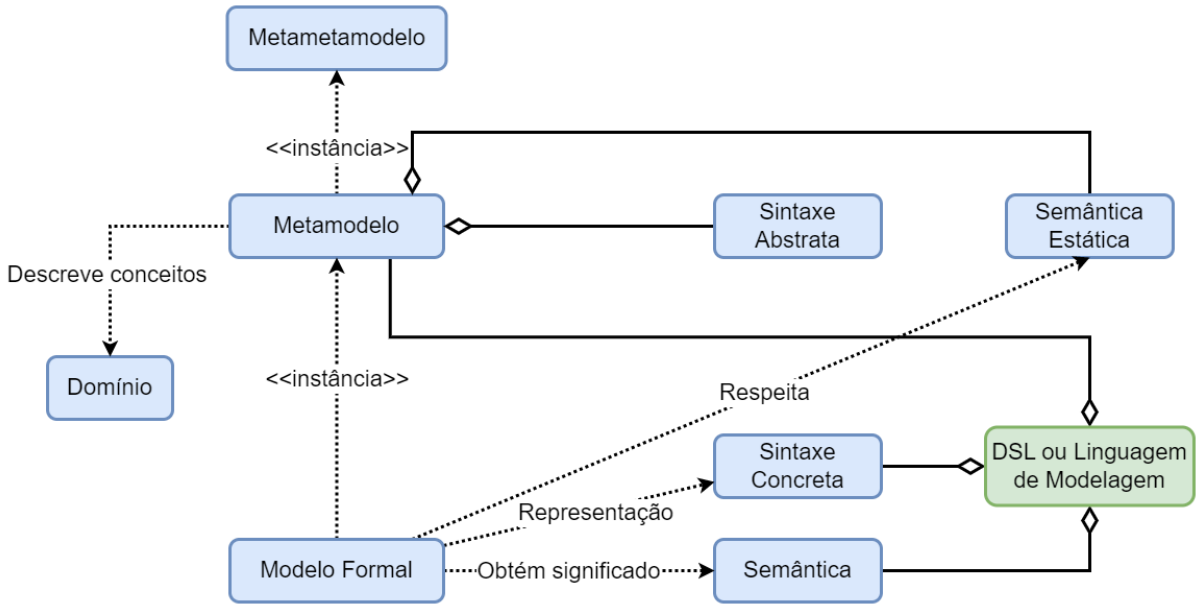


Figura 8 – Conceitos gerais de MDE.

domínio e idioma é mapeado para uma notação específica. Por exemplo, pode ser possível usar notação gráfica, como UML, ou uma notação textual, como Java.

Modelos formais são exemplos de metamodelos representados por uma sintaxe concreta. Além disso, eles também devem respeitar a semântica estática que o metamodelo possui para executar estruturas coerentes dentro de um domínio de conhecimento.

Uma Linguagem Específica de Domínio (DSL - *Domain Specific Language*) visa fazer duas coisas: (i) aumentar o nível de abstração além da programação por domínio de problema específico; e (ii) gerar código em uma linguagem de programação escolhida a partir dessas especificações de alto nível (KELLY; TOLVANEN, 2008).

Ao contrário das Linguagens de Modelagem de Propósito Geral (GPML - *General Purpose Modeling Languages*), as DSL são normalmente compostas por um conjunto de conceitos e notações gráficas que estão próximos do campo do respectivo domínio (ARNE et al., 2016). Isso fornece ao aplicativo o princípio de modelagem multiponto ou o princípio da separação de interesses. Este princípio afirma que um sistema complexo é melhor definido por múltiplas visões, considerando aspectos estáticos e dinâmicos (BOOCH; RUMBAUGH; JACOBSON, 1999; BRAMBILLA; CABOT; WIMMER, 2012).

Portanto, uma linguagem de modelagem fornece um ou mais pontos de vista classificados de acordo com as propriedades de abstração e perspectiva. Na dimensão de abstração, pode-se classificar um ponto de vista usando, por exemplo, a terminologia

de Arquitetura Dirigida por Modelo (MDA - *Model Driven Architecture*) (OMG, 2003), como o Modelo Independente de Plataforma (PIM - *Platform Independent Model*) ou o Modelo Específico de Plataforma (PSM - *Platform Specific Model*).

Gerard, Babau e Champeau (2010) consideram dois tipos de transformações no MDE para aumentar o nível de abstração do desenvolvimento de software usando modelos construídos em termos próximos aos conceitos do domínio do problema em questão. As transformações de modelo para texto (M2T) geram ou produzem artefatos de software: código-fonte, XML (*Extensible Markup Language*) e outros arquivos de texto. As transformações de modelo para modelo (M2M) convertem os modelos em outro conjunto de modelos, geralmente mais próximos do domínio da solução ou atendendo às necessidades de diferentes partes interessadas (SILVA, 2015).

O modelo conceitual proposto por Silva (2015) apresenta vários conceitos, tais como:

- O aspecto “Modelos” define os níveis de abstração (PIM ou PSM) e as linguagens de modelagem implementadas por cada abordagem.
- O aspecto “Transformações” define os tipos (M2M ou M2T) e os idiomas usados por cada abordagem para suportar o modelo de transformação.
- O aspecto “Linguagens de metamodelagens” define as tecnologias usadas nas linguagens de modelagem.
- O aspecto do “Domínio” define os domínios de aplicação de cada abordagem, que neste caso lida apenas com abordagens concretas que são específicas do domínio. Como o domínio “robótico"é diverso, especializamos esse aspecto para os “subdomínios"dentro do campo da robótica.
- Finalmente, o aspecto “suporte da ferramenta” define o tipo de ferramenta suportada por cada abordagem.

2.3 Aprendizado de Máquina

Na segunda metade do século XX, a aprendizagem de máquina (ML - *Machine Learning*) evoluiu como um subcampo de Inteligência Artificial (IA) que envolveu o desenvolvimento de algoritmos de autoaprendizagem para obter conhecimento a partir desses dados, a fim de fazer previsões (RASCHKA, 2015).

Avanços contínuos no poder computacional (de acordo com a Lei de Moore) começaram a tornar o aprendizado de máquina, antes uma disciplina de pesquisa, mais viável em contextos comerciais. Isso causou uma explosão de novas aplicações e técnicas novas ou redescobertas, catapultando os conceitos obscuros de ciência de dados, IA e aprendizado de máquina para a consciência pública e planejamento estratégico das empresas internacionalmente (HEARTY, 2016).

O aprendizado de máquina é a ferramenta usada para processamento de dados em larga escala, além de ser bem adequado para conjuntos de dados complexos que possuem um grande número de variáveis e características (VASILEV et al., 2019).

O aprendizado de máquina é comumente divido em três grandes classes:

- Aprendizado Supervisionado;
- Aprendizado Não Supervisionado; e
- Aprendizado por Reforço

2.3.1 Aprendizado Supervisionado

Algoritmos de aprendizado supervisionado são uma classe de algoritmos de aprendizado de máquina que usam dados previamente rotulados para aprender suas características, para que eles possam classificar dados semelhantes, mas não rotulados (VASILEV et al., 2019).

O principal objetivo da aprendizagem supervisionada, como mostra a Figura 9, é aprender um modelo a partir de dados de treinamento rotulados que nos permitam fazer previsões sobre dados não vistos ou futuros. Aqui, o termo supervisionado refere-se a

um conjunto de amostras onde os sinais de saída desejados (rótulos) já são conhecidos (RASCHKA; MIRJALILI, 2017).

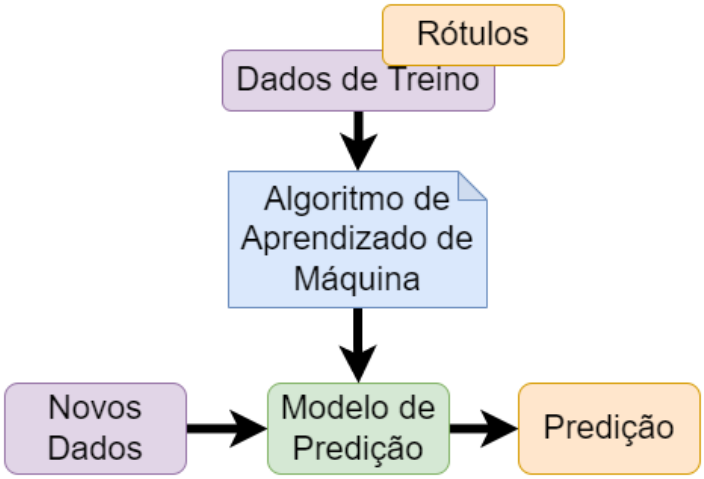


Figura 9 – Abordagem do Aprendizado Supervisionado (RASCHKA, 2015).

No aprendizado supervisionado, o objetivo é inferir uma função ou mapeamento a partir dos dados de treinamento rotulados. Os dados de treinamento consistem no vetor de entrada X e no vetor de saída Y de rótulos. Um rótulo do vetor Y é a explicação de seu respectivo exemplo de entrada do vetor de entrada X (MOHAMMED; KHAN; BASHIER, 2016).

Dois grupos ou categorias de algoritmos se enquadram no campo da aprendizagem supervisionada. Eles são: Regressão e Classificação.

Os algoritmos de regressão são um tipo de algoritmo supervisionado que usa características dos dados de entrada para prever um valor. A análise de regressão tenta encontrar o valor dos parâmetros para a função que melhor se ajusta a um conjunto de dados de entrada. Em um algoritmo de regressão linear, o objetivo é minimizar uma função de custo, encontrando parâmetros apropriados para a função, sobre os dados de entrada que melhor se aproximam dos valores-alvo (VASILEV et al., 2019).

O objetivo dos algoritmos de classificação são prever os rótulos de classe categóricos de novas instâncias, com base em observações anteriores. Esses rótulos de classe são valores discretos e não ordenados, que podem ser entendidos como membros do grupo das instâncias (RASCHKA; MIRJALILI, 2017).

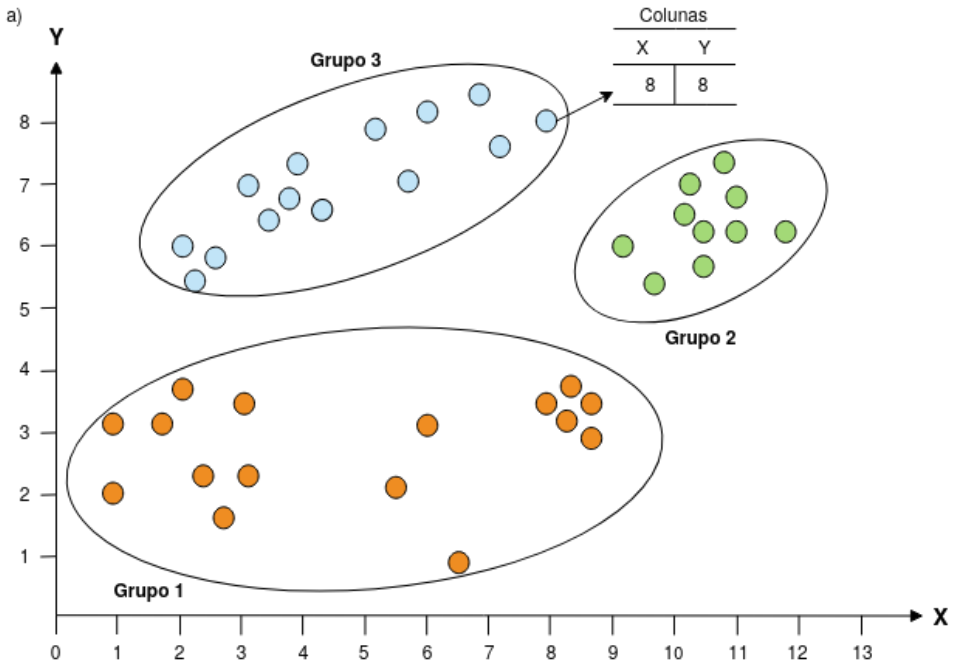


Figura 10 – Exemplo de agrupamento (VASILEV et al., 2019).

2.3.2 Aprendizado Não Supervisionado

Técnicas de aprendizado não supervisionadas são um valioso conjunto de ferramentas para análise exploratória. Eles trazem padrões e estruturas dentro dos conjuntos de dados, que produzem informações que podem ser informativas em si ou servem como um guia para uma análise mais aprofundada (HEARTY, 2016).

Os dados não são rotulados, mas o algoritmo deve chegar a uma conclusão baseado nas características comuns entre eles. Um dos exemplos mais comuns e talvez mais simples de aprendizado não supervisionado é o agrupamento (VASILEV et al., 2019). Essa é uma técnica que tenta separar os dados em subconjuntos, como pode ser visto na Figura 10.

2.3.3 Aprendizado Por Reforço

Aprendizagem por Reforço (*Reinforcement Learning* - RL) é uma abordagem através da qual os programas, conhecidos como agentes, trabalham em um ambiente conhecido ou desconhecido para se adaptar e aprender constantemente com base em pontos recebidos. Esses pontos podem ser positivos, também conhecidos como

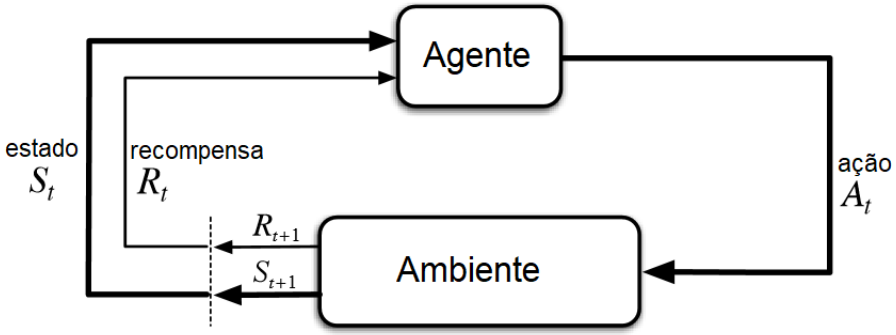


Figura 11 – Abordagem de Aprendizado por Reforço (VASILEV et al., 2019).

recompensas, ou negativos, também chamados de punições. Considerando os agentes e a interação com o ambiente, determina-se qual ação tomar (NANDY; BISWAS, 2017). A Figura 11 apresenta o funcionamento da abordagem de RL.

Nesse tipo de aprendizado, o objetivo é desenvolver um sistema (agente) que melhore seu desempenho com base nas interações com o ambiente (RASCHKA, 2015).

Alguns pontos importantes sobre RL descritos por Nandy e Biswas (2017) são:

- Difere do Aprendizado de Máquina normal, pois não são analisados os conjuntos de dados de treinamento.
- A interação não acontece com dados, mas com ambientes, através dos quais são representados por cenários do mundo real.
- Como RL é baseado em ambientes, muitos parâmetros entram em jogo. É preciso muita informação para aprender e agir de acordo.
- Ambientes na RL são cenários do mundo real que podem ser mundos simulados 2D ou 3D ou cenários baseados em jogos.
- RL é mais amplo porque os ambientes podem ser de grande escala e pode haver muitos fatores associados a eles.
- O objetivo do RL é alcançar uma meta.
- Recompensas em RL são obtidas do meio ambiente.

RL se concentra em aprender como mapear situações para ações de modo a maximizar uma recompensa, que é um valor associado a ação. O agente não é infor-

mado sobre quais ações executar, em vez disso, deve descobrir quais ações geram mais recompensas, tentando executá-las. As ações podem afetar não apenas a recompensa imediata, mas também a próxima situação e, com isso, todas as recompensas subsequentes (SUTTON; BARTO, 2018).

Os elementos básicos de um sistema de RL são apresentados por (VASILEV et al., 2019):

- Agente: a entidade para a qual estamos tentando aprender ações.
- Ambiente: o mundo em que o agente opera.
- Estado: todas as informações disponíveis para o agente sobre seu ambiente atual.
- Ação: Uma possível resposta ou conjunto de respostas que um agente pode executar. Após cada ação, o ambiente muda de estado e, em seguida, fornece feedback ao agente.
- Recompensa: o feedback que o agente recebe do ambiente após cada ação. O principal objetivo do agente é maximizar o retorno total (recompensas acumuladas) a longo prazo.
- Política: determina quais ações o agente executará, dado o estado atual. No contexto do aprendizado profundo, podemos treinar uma rede neural para tomar essas decisões. No decorrer do treinamento, o agente tentará modificar sua política para tomar melhores decisões. A tarefa de encontrar a política ideal é chamada de aprimoramento (ou controle) de políticas e é uma das principais tarefas de RL.
- Função Valor: determina o que é bom para o agente a longo prazo (diferente da recompensa imediata). Ou seja, quando aplicada a função de valor em um determinado estado, ele nos informa o retorno total que podemos esperar no futuro, se começarmos a partir desse estado. A política do agente levará em consideração o valor - e, em menor grau, a recompensa - ao decidir que ação tomar. A tarefa de encontrar a função de valor é chamada de previsão (também conhecida como avaliação de políticas).

Os ambientes no aprendizado por reforço são representados pelo Processo de Decisão de Markov (MDP - *Markov decision process*) (NANDY; BISWAS, 2017). Um MDP é uma estrutura matemática para modelar decisões. Podemos usá-lo para descrever o problema da RL. Um MDP fornece uma definição formal das propriedades e é formado por uma tupla (S, A, P, R, γ), onde (VASILEV et al., 2019):

- S é o conjunto finito de todos os estados possíveis do ambiente e s_t é o estado no tempo t;
- A é o conjunto de todas as ações possíveis, e a_t é a ação no momento t.
- P é a dinâmica do ambiente (também conhecida como matriz de probabilidades de transição). Ele define a probabilidade condicional de fazer a transição para um novo estado, s', considerando o estado existente, s e uma ação, a (para todos os estados e ações):

$$P^a_{ss'} = Pr(s_t + 1 = s'|s_t = s, a_t = a) \tag{2.1}$$

  Temos probabilidades de transição entre os estados, porque o MDP é estocástico (inclui aleatoriedade). Essas probabilidades representam o modelo do ambiente, isto é, como ele provavelmente mudará, dado seu estado atual e uma ação, a.

- R é a função de recompensa. Ela descreve a recompensa que o agente receberia quando executa a ação a e faz a transição de $s \rightarrow s'$:

$$R^a_{ss'} = \mathbb{E}[r_{t+1} = s', s_t = s, a_t = a] \tag{2.2}$$

  A expectativa $\mathbb{E}$ é necessária porque o ambiente descrito pelo MDP é estocástico e as transições entre estados diferentes são descritas pelas probabilidades de transição. Em outras palavras, não podemos dizer com certeza em que novo estado (e, portanto, qual recompensa) terminaremos quando o agente tomar uma ação.

- γ é o fator de desconto. É um valor no intervalo $[0 : 1]$ e determina quanto o algoritmo valoriza as recompensas imediatas, em oposição às recompensas futuras.

A tarefa de aprendizagem em um MDP é encontrar uma política $\pi : S \rightarrow A$ para selecionar ações de maior retorno futuro. A qualidade de uma política é indicada por uma função valor V^{π}. O valor de $V^{\pi}(s)$ específica a quantidade total de recompensa que um agente pode acumular no futuro, iniciando no estado s e então seguindo a política π. Em um MDP com horizonte infinito descontado, a expectativa de recompensa acumulada é denotada por:

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) | s_0 = s\right] \tag{2.3}$$

O valor para uma dada política π, expressa pela Eq. 2.3, pode ser iterativamente computada pela equação de Bellman (BELLMAN; KALABA, 1965). Esta função, tipicamente, é inicializada com valores de função escolhidos arbitrariamente e, a cada iteração, para todo $s \in S$, o valor das funções são atualizados com base na recompensa imediata e na estimativa atual de V_{π}, definida por:

$$V_{t+1}^{\pi} = R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_{t}^{\pi}(s') \tag{2.4}$$

O objetivo do MDP é encontrar uma política de estados e ações ótima que maximize a recompensa recebida. A política ótima $\pi^{*}(s)$ é tal que $V^{\pi^{*}}(s) \geq V^{\pi}(s)$ para todo $s \in S$ e todas as políticas π.

2.3.3.1 Algoritmo Q-Learning

Q-learning é um algoritmo de aprendizado por reforço sem modelo com o objetivo de aprender uma política, que diga a um agente que ação tomar em que circunstâncias. Ele não requer um modelo do ambiente e pode lidar com problemas com transições e recompensas estocásticas, sem exigir adaptações.

Para qualquer processo finito de decisão de Markov (FMDP), o Q-learning encontra uma política ideal no sentido de maximizar o valor esperado da recompensa total em todas e quaisquer etapas sucessivas, a partir do estado atual (MELO, 2001).

Q-learning é um algoritmo de controle de Diferença Temporal (TD) fora da política criado por (WATKINS, 1989), definido como:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{2.5}$$

Nesse caso, a função de valor-ação aprendida, Q, aproxima-se diretamente de $q*$, a função ideal de valor-ação, independentemente da política que está sendo seguida (SUTTON; BARTO, 2018). Antes do início do aprendizado, Q é inicializado com um valor possivelmente arbitrário. Então, a cada tempo t o agente seleciona uma ação a_t, observa uma recompensa r_t, entra em um novo estado s_{t+1} (que pode depender do estado anterior s_t e da ação selecionada) e Q é atualizada. O núcleo do algoritmo é uma atualização simples da iteração de valor, usando a média ponderada do valor antigo e as novas informações. O algoritmo Q-learning é mostrado no Algoritimo 1 na forma procedural.

Algorithm 1 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $epsilon > 0$ Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

foreach *episode* **do**
  Initialize S
  foreach *step of episode* **do**
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ $S \leftarrow S'$
  end
end

No Q-learning, uma tabela é construída na memória $Q[s, a]$ para armazenar valores Q para todas as combinações possíveis de s e a. Com essa tabela é determinado a próxima ação a ser executada com valores máximos de $Q(s', a')$. No entanto, se as combinações de estados e ações forem muito grandes, a memória e o requisito de computação para Q serão muito altos. Isso torno seu uso inviável em problemas robóticos do mundo real mais complexos. Com intuito de resolver esse problema surgiu

uma nova abordagem, chamada de Deep Q-Learning (MNIH et al., 2013) que generaliza a aproximação da função de valor Q em vez de armazenar as soluções para cada estado possível.

2.3.3.2 Algoritmo Deep Q-Network DQN

Os recentes avanços nas redes neurais profundas, nas quais várias camadas de nós são usadas para criar progressivamente mais representações abstratas dos dados, tornaram possível que as redes neurais artificiais aprendessem conceitos como categorias de objetos diretamente a partir de dados sensoriais brutos (MNIH et al., 2015).

As redes neurais profundas são aproximadores de função de uso geral adequados que foram mais amplamente utilizados em tarefas de aprendizado supervisionado. Recentemente, no entanto, eles têm sido aplicados a problemas de aprendizagem por reforço, dando origem ao campo da Aprendizagem por Reforço Profundo (DRL - *Deep Reinforcement Learning*). Este campo procura combinar os avanços em redes neurais profundas com algoritmos de aprendizado por reforço para criar agentes capazes de agir de maneira inteligente em ambientes complexos (HAUSKNECHT; STONE, 2015).

O método Deep Q-Network introduzido por (MNIH et al., 2013) emprega uma rede profunda para estimar a função de valor de cada ação discreta e, ao agir, seleciona a saída com valor máximo para um determinado estado observado. Foi utilizada uma rede neural profunda convolucional (CNN - *Convolutional Neural Network*), representada na Figura 12 para aproximar a função de valor de ação ideal

$$Q^*(s,a) = max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + r_t + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi\right], \tag{2.6}$$

que é a soma máxima de recompensas r_t descontadas por γ em cada etapa t, alcançável por uma política de comportamento $\pi = P(a|s)$, depois de fazer uma observação (s) e executar uma ação (a).

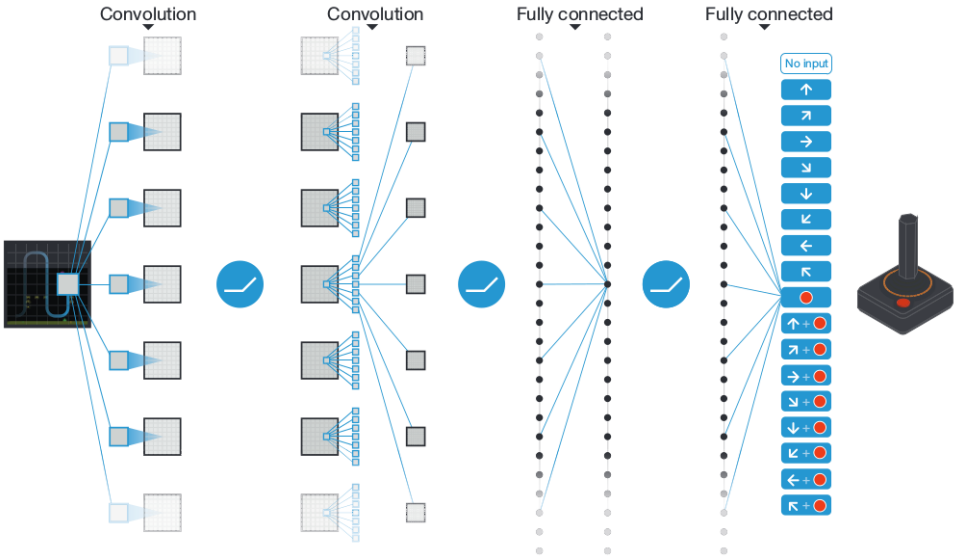


Figura 12 – Ilustração esquemática da rede neural convolucional (MNIH et al., 2015).

O algoritmo DQN criado por (MNIH et al., 2015) foi capaz de desempenho no nível humano em muitos jogos Atari usando pixels brutos de entrada. O algoritmo pode ser observado no Algoritmo 2.

Algorithm 2 Deep Q-learning com Replay da Experiência

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights θ^{-}
for *episode* = *1, M* **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi = \phi(s_1)$
  for t = *1, T* **do**
    With probability ϵ select a random action a_t otherside select $a_t = argmax_a Q(\phi(s_t), a; z\theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transition $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } \phi_{j+1} \\ r_j + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta) & \text{otherside} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end
end

2.3.3.3 Algoritmo Advantage Actor Critic (A2C)

Os algoritmos de ator-crítico baseados em gradiente de política estão entre os algoritmos mais populares na estrutura de aprendizado por reforço. Sua vantagem de poder procurar políticas ótimas usando estimativas de gradiente de baixa variância as tornou úteis em várias aplicações da vida real, como robótica, controle de energia e finanças.

Métodos somente críticos, como Q-learning (WATKINS, 1989) e SARSA (RUMMERY; NIRANJAN, 1994), usam uma função de valor de ação de estado e nenhuma função explícita para a política. Para espaços de estado e ação contínuos, esta será uma função de valor de ação de estado aproximada. Esses métodos aprendem a função de valor ótimo encontrando, online, uma solução aproximada para as equações de Bellman (Eq. 2.4).

Os métodos, como os algoritmos RSV (GULLAPALLI, 1990) e REINFORCE (WILLIAMS, 1992), são principalmente apenas atores e não usam nenhuma forma de função de valor armazenado. Em vez disso, a maioria dos algoritmos somente de atores trabalha com uma família parametrizada de políticas e otimiza o custo diretamente sobre o espaço de parâmetros da política. Uma grande vantagem dos métodos somente de atores sobre os métodos somente críticos é que eles permitem que a política gere ações no espaço de ação contínuo completo.

Os métodos de ator-crítico (WITTEN, 1977) visam combinar as vantagens dos métodos somente do ator e somente do crítico. Assim como os métodos somente de ator, os métodos críticos de ator são capazes de produzir ações contínuas, enquanto a grande variação nos gradientes de política dos métodos somente de ator é combatida pela adição de um crítico. O papel do crítico é avaliar a atual política prescrita pelo ator. O crítico aproxima e atualiza a função de valor usando amostras. A função de valor é então usada para atualizar os parâmetros de política do ator na direção da melhoria do desempenho. Esses métodos geralmente preservam as propriedades de convergência desejáveis dos métodos de gradiente de política, em contraste com os métodos somente críticos. Em métodos de ator-crítico, a política não é inferida diretamente da função de valor usando a Eq. 2.3. Em vez disso, a política é atualizada na direção do gradiente da

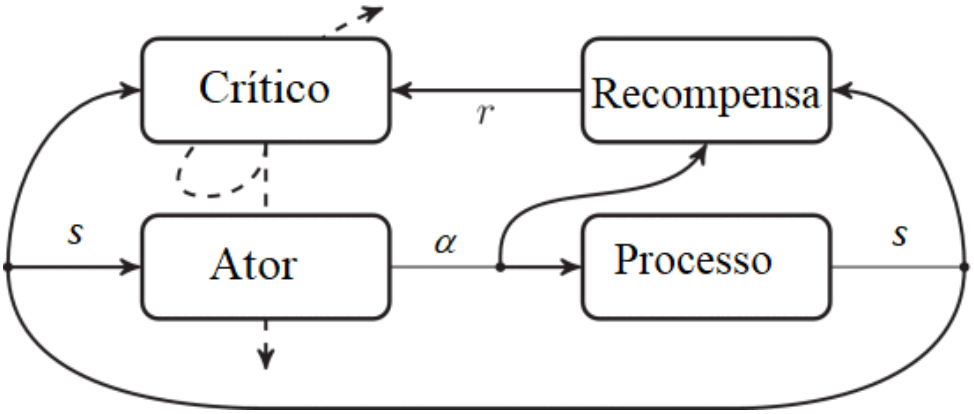


Figura 13 – Visão geral esquemática do algoritmo crítico de ator. Adaptado de Grondman et al. (2012).

política usando apenas um pequeno tamanho de etapa, significando que uma mudança na função valor resultará apenas em uma pequena mudança na política, levando a um comportamento menos ou nenhuma oscilação na política (GRONDMAN et al., 2012).

A Figura 13 mostra a estrutura esquemática de um algoritmo ator-crítico. O agente de aprendizagem foi dividido em duas entidades separadas: o ator (política) e o crítico (função de valor). O ator é responsável apenas por gerar uma entrada de controle (a), dado o estado atual (s). O crítico é responsável por processar as recompensas que recebe, ou seja, avaliar a qualidade da política atual adaptando a estimativa da função valor. Após várias etapas de avaliação de política pelo crítico, o ator é atualizado usando as informações do crítico. O Algoritmo 3 apresenta sua forma procedural. A linha tracejada indica que o crítico é responsável por atualizar o ator e a si mesmo.

Algorithm 3 Advantage Actor-Critic

Require: Initialize Actor-Critic network G with parameter θ.

for each episode **do**
  Get initial state s
  Initialize a storage buffer S, A, R, V, S'
  for $i = 1, 2, 3.., N$ **do**
    Sample an action $a \sim G(s)_\pi$ and get the associeted value $v \leftarrow G(s)_v$
    Run the action a through the environment, obtain the reward and next state $r, s' \leftarrow ENV(s, a)$
    Collect and store $S, A, R, V, S' \leftarrow s, a, r, v, s'$
    $s \leftarrow s'$
  end for
  Compute the discounted returns $\hat{V} = \sum_{l=0}^{N-1} \gamma^l r_{t+l}$
  Compute an advantage function $\psi(V, R, S')$
  Optimize θ to minimize $-\log(G(A \mid S)_\pi)\psi(V, R, S') + \lambda\|V - \hat{V}\|$
  Empty S, A, R, V, S'

2.3.3.4 Algoritmo Deep Deterministic Policy Gradient (DDPG)

Como o Q-learning não pode ser aplicado diretamente ao espaço de ação contínua, Lillicrap et al. (2015) combinou a estrutura ator-crítico com gradiente de política determinístico para obter um novo algoritmo com a capacidade para o controle contínuo chamado DDPG. O DDPG é uma versão aprimorada do algoritmo ator-crítico, que usa as redes neurais profundas para estimar a função de política ideal em vez de escolher a ação com base em uma distribuição específica (QIU et al., 2019).

O algoritmo DDPG pode evitar a maldição da dimensionalidade em comparação com MDP e Q-learning, que exigem a discretização do estado. Consiste em um método que usa funções de aproximação que podem aprender políticas de espaço de ação contínua. O algoritmo faz uso de uma rede neural para a rede do ator e outra para a rede crítica. Essas duas redes calculam a predição da ação para o estado atual e geram um sinal de erro de diferença temporal para cada intervalo de tempo. A entrada da rede de atores é o estado atual e a saída é um valor real que representa uma ação escolhida para um espaço de ação contínuo. A saída do crítico é simplesmente o valor Q estimado do estado atual e a ação dada pelo ator (JESUS et al., 2019). O algoritmo DDPG é apresentado, em sua forma procedural, em Algoritmo **??**.

Algorithm 4 Deep Deterministic Policy Gradient Algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

3

Trabalhos Relacionados

Este capítulo está dividido em duas partes. Na Seção 3.1 são apresentados as principais abordagens relacionadas com as fases de projeto e, em seguida, na Seção 3.2 são apresentados as abordagens que buscam a adaptação em tempo de execução como forma de agregar mais autonomia aos sistemas robóticos.

3.1 Abordagens direcionadas as fases de projeto

A Revisão Sistemática da Literatura (RSL) tem o intuito de verificar o estado da arte no desenvolvimento de software robótico por meio de abordagens dirigidas por modelos. A revisão evidenciou uma evolução na aplicação dos modelos dentro das etapas de desenvolvimento de software definidas por Sommerville *et. al.* (SOMMERVILLE; ARAKAKI; MELNIKOFF, 2008). Essas fases de desenvolvimento são divididas em: (1) Análise de Requisitos, (2) Projeto, (3) Implementação e Teste, (4) Integração, (5) Execução e (6) Manutenção. Desta forma, foi possível verificar em que medida as abordagens analisadas se estendem às fases de desenvolvimento de software. A maioria das abordagens aborda apenas a fase de projeto 1 - Análise de Requisitos até a fase 4 - Integração. O resultado da RSL publicado em um survey pode ser visto na integra em Silva et al. (2021).

O roteiro plurianual de robótica da Europa 2020 (ROBOTICS, 2016) assume que os modelos podem ser usados em robótica em três passos. O passo 1 assume que os modelos são usados para definir missões por pessoas na fase de projeto. O passo 2 exige que os robôs usem modelos de tempo de execução para interagir com o ambiente

e explicar o que está acontecendo. O passo 3 significa que os robôs se adaptam e melhoram os modelos para redefinir o que estão fazendo com base em seu aprendizado. A RSL evidenciou poucos trabalhos que lidam com o uso de modelos de tempo de execução, por exemplo, ao tomar decisões sobre quais dispositivos usar ou algoritmos para executar com base nas condições de seu ambiente atual.

As abordagens que contemplam o passo 1 do roteiro são abordagens que avançaram nas etapas de projeto do software robótico. Dentre elas o trabalho de Figat e Zieliński (2022), no qual foi proposto um metamodelo que, segundo o autor, é capaz de gerar um controlador robótico composto por qualquer hardware e capaz de realizar qualquer tarefa. Para isso, a descrição da atividade baseia-se em uma representação matemática de uma rede de Petri denominada Robotic System Hierarchical Petri Net (RSHPN) que tem a finalidade de promover a separação de interesses. O metamodelo proposto visa facilitar o processo de desenvolvimento do sistema robótico através da parametrização, orientando o projetista durante a fase de projeto. Além disso, a abordagem possui alguns recursos de verificação e validação do sistema, inerentes ao uso da rede de Petri, como segurança e vivacidade. Também é possível gerar código para middleware ROS em C++ e Python. Apesar da proposta de facilitar o processo de desenvolvimento de software robótico, a necessidade de criar uma rede de Petri de seis camadas torna seu uso complexo e, portanto, dificulta sua adoção. Para minimizar este problema, foi desenvolvida uma ferramenta chamada Robotic System Specification Language (RSSL). No entanto, não há muita explicação no artigo sobre essa ferramenta. Este trabalho trata as tarefas como atividades pontuais e não antecipa situações inesperadas. Nossa proposta possui uma ferramenta gráfica que facilita seu uso, dando ao usuário a possibilidade de montar um treinamento apenas arrastando componentes visuais.

Wigand et al. (2017) introduz um esquema de composição de linguagem e gerador especificamente adaptado aos sistemas robóticos baseados em componentes. Tem suporte a reutilização, extensibilidade, refinamento de linguagem específica de domínio e geradores de código. Sua proposta além de modelar capacidades independentes da plataforma, considera a modelagem da plataforma de software e hardware como dimen-

sões separadas. A abordagem está fundamentada nessas três dimensões: capacidade, software e hardware. A dimensão de capacidade trata das partes funcionais de um sistema robótico que são independentes de qualquer estrutura de software e hardware. Esta dimensão baseia-se na terminologia MDA, onde se aplica os Modelo Independente de Plataforma (PIM - *Independent Platform Model*) e Modelo Especifico de Plataforma (PSM - *Platform Specific Model*). Na dimensão de hardware os diferentes aspectos da plataforma são modelados, incluindo cinemática, dinâmica, restrições de segurança, interface de controle. A dimensão de software se concentra nas estruturas de software direcionadas à geração de código. Também possui um gerador de código modular e flexível para suportar vários destinos de geração heterogêneos.

A abordagem de Estevez et al. (2017) apresenta uma plataforma para suportar o ciclo de desenvolvimento de manipuladores robóticos. Foi desenvolvido um perfil UML denominado TARMAN que pode ser usado por qualquer ferramenta de modelagem UML que suporte o padrão XMI (*XML Metadata Interchange*). A modelagem da lógica e da arquitetura dos softwares é feita com notação UML, que torna a metodologia proposta genérica e válida para qualquer ferramenta de modelagem UML que suporte o padrão XMI. Foi definida uma transformação M2M, cujo modelo de entrada é o arquivo XMI, que será processado para gerar o modelo TARMAN.xml. A geração do código é realizada por meio de uma transformação M2T que recebe como entrada o modelo anterior. A plataforma desenvolvida foi testada nos midlewares ROS e OROCOS.

A abordagem HyperFlex (BRUGALI; GHERARDI, 2016) se baseia nos conceitos de arquitetura de software estável e modelagem de variabilidade. A abordagem é suportada por um conjunto de ferramentas MDE que permitem aos desenvolvedores de software robótico modelar a arquitetura do software e a variabilidade funcional de uma família de sistemas de controle similares (SPL) desenvolvidos para um determinado domínio. O HyperFlex usa o formalismo dos Modelos de Recursos (KANG et al., 1990) para representar simbolicamente a variabilidade de um sistema funcional e as restrições entre pontos de variação e variantes que limitam o conjunto de configurações válidas. Permite modelar explicitamente a arquitetura de sistemas funcionais em termos de componentes, interfaces, conectores de componentes. Modelos de sistemas funcionais

podem ser reutilizados como blocos de construção e compostos hierarquicamente para criar sistemas e aplicativos mais complexos.

No trabalho de Ciccozzi et al. (2016) os autores propuseram uma família de linguagem de modelagem específica de domínio para a especificação de missões de sistemas móveis com vários robôs. A abordagem é composta de três DSLs diferentes compreendendo MML (*Monitoring Mission Language*), RL (*Robot Language*) e BL (*Behaviour Language*). A camada MML traz os conceitos que são usados para especificar missões sem se referir a aspectos específicos, como as características técnicas dos robôs que realizarão as missões. Na prática, essa camada permite especificar missões como sequências de tarefas, ligadas entre si através de dependências, bifurcações e junções de tarefas. A especificação de uma missão é acompanhada pela definição do contexto espacial em que a missão deve ser executada fisicamente. A camada RL especifica o tipo e a configuração dos robôs individuais que devem cumprir a missão especificada. Por fim a camada BL fornece os meios para definir uma especificação dos movimentos e ações atômicos dos robôs individuais contendo todos os movimentos e ações de baixo nível dos robôs.

Nordmann, Wrede e Steil (2015) apresenta uma estrutura conceitual e um metamodelo para arquiteturas de controle de movimento baseadas em primitivas de movimento. O metamodelo é composto por duas DSLs que modelam dois aspectos: i) aspectos estruturais de arquiteturas de movimento que focam nos chamados Módulos Adaptativos que modelam primitivas de movimento como sistemas dinâmicos com capacidades de aprendizado de máquina; e ii) aspectos comportamentais para a coordenação de sistemas de controle de movimento. A flexibilidade para adaptar primitivas de movimento a diferentes situações e ambientes é obtida por meio de técnicas de aprendizado de máquina. Foi construída uma validação qualitativa de n Controladores de Rastreamento, incluindo redes neurais recorrentes como Módulos Adaptativos e uma Transformação de escala. A superposição de saídas do Controlador de Rastreamento resultou na mistura pretendida de controladores. Para demonstrar sua aplicação, foi utilizado um braço robótico. A validação da proposta foi apenas de natureza qualitativa, necessitando de outros experimentos para validar características não funcionais, como

tempo de resposta, taxa de sucesso e outros parâmetros. Outro diferencial é que nossa proposta busca favorecer o aprendizado a partir de dados brutos de sensores, o que pode garantir uma melhor resposta a interações inesperadas com o ambiente.

3.2 Abordagens com adaptação em tempo de execução

O trabalho de (NAGABANDI et al., 2018) propõe uma abordagem de meta-aprendizagem para adaptação online usando um modelo de aprendizagem por reforço. Nesta abordagem, os autores consideram cada segmento de uma trajetória como uma "tarefa" diferente, e as observações dos passos de tempo anteriores são consideradas como informações sobre a configuração atual das tarefas. Essa visão induz uma configuração de problema meta-RL mais geral, permitindo que a noção de uma tarefa represente qualquer coisa em uma parte diferente do espaço de estados, experimente perturbações ou tente alcançar um novo objetivo. Eles contornam o problema de adquirir um modelo global que seja preciso em todo o espaço de estados treinando especificamente um modelo de rede neural para exigir apenas uma pequena quantidade de experiência para se adaptar. Apesar de o treinamento ser mais rápido e não exigir muitos dados, os autores não deixam claro o tempo de resposta de possíveis mudanças no ambiente. Isso é um problema quando se considera a aplicação em ambientes reais que possuem requisitos de tempo como carros autônomos ou de resgate. Nossa abordagem, em contraste, visa o emprego mútuo de algoritmos de aprendizado de máquina com técnicas de controle clássicas.

FLYAQ é uma plataforma projetada para permitir que usuários de domínio não especialistas programem missões para uma equipe de multicópteros. O trabalho de Dragule et al. Dragule, Meyers e Pelliccione (2017) estendeu a plataforma FLYAQ no sentido de apoiar robôs adaptativos no nível da missão, o que significa que os robôs podem mudar seu comportamento para realizar missões em circunstâncias variadas. Isso foi alcançado com a inclusão de uma linguagem declarativa chamada MSL (Mission Specification Language). Portanto, basta especificar os objetivos que devem ser alcançados e quais restrições não devem ser violadas. O MSL destina-se a especificar

propriedades de uma missão que permitem aos usuários definir restrições de tempo de forma declarativa. Essas restrições são baseadas em padrões temporais que podem ser ausência (algo nunca deve ocorrer), universalidade (algo deve sempre ocorrer), existência (algo deve eventualmente ocorrer), existência limitada (algo deve ocorrer no máximo n vezes), precedência (ocorrência de P deve ser precedido por uma ocorrência de Q), ou resposta (uma ocorrência de P deve ser seguida por uma ocorrência de Q). A utilização dessa abordagem requer conhecimento da linguagem MSL, o que dificulta seu uso. Em vez disso, nossa abordagem garante um aprendizado rápido usando a ferramenta de geração de código baseada em GUI.

Uma plataforma não nomeada foi desenvolvida em (DJUKIĆ; POPOVIĆ; TOLVANEN, 2016), que é capaz de descrever sua atividade em tempo de execução. A plataforma também tem como objetivo gerar cenários de teste e documentação durante a depuração em nível de modelo. O cenário de aplicação adota o braço robótico industrial IGUS Robolink D. Está dividida em três DSLs. A primeira é usada para especificar características topológicas de uma mão ou braço. A segunda DSL é criada para especificar o ambiente no qual um robô executa suas ações (paleta, tela, tigela para limpeza e um pincel de pintura). A terceira DSL é usada para especificação da lógica de controle, sensores com sinais de entrada e motores/atuadores realizando movimentos. Para esta tarefa, eles usaram a linguagem IEC 61131-3 visando a especificação de diagramas de blocos de funções. A propriedade mais importante da tela é um bitmap contendo um retrato que deve ser pintado. As trajetórias são calculadas em tempo de execução quando a paleta é movida ou girada, e também quando dois braços com juntas de raiz móveis são usados.

SMARTMARS (STECK; LOTZ; SCHLEGEL, 2011) é um Perfil UML baseado em Papyrus UML. Vários modelos são criados em tempo de design pelo desenvolvedor e usados em tempo de execução pelo robô. Assim, tanto o engenheiro do sistema quanto o robô raciocinam usando os parâmetros, restrições, propriedades e outras informações explicadas nos mesmos modelos. Tal acesso se dá por meio de diferentes representações. O conhecimento sobre como transformar modelos entre as diferentes representações com informações parciais deve ser considerado, além de qual representação é necessária.

Os resultados que podem ser reincorporados são codificados nas ferramentas (tempo de design) e nos gráficos de ação (tempo de execução). Os modelos compreendem pontos de variação que são vinculados durante a fase de desenvolvimento. Esses pontos de variação estão vinculados aos componentes pré-programados que são executados de acordo com fatores previamente configurados. Este trabalho faz uso dos pontos de variação que são adicionados nas etapas de projeto da solução, porém, não utilizam o aprendizado de máquina como possíveis fontes de controle. Nossa abordagem aproveita as vantagens que o aprendizado de máquina proporciona no campo da robótica e busca aliar isso ao uso de técnicas clássicas de controle, favorecendo um controle robusto e resiliente às mudanças no ambiente.

A tabela 1 apresenta um resumo dos trabalhos analisados, classificando-os de acordo com alguns atributos relevantes para pesquisas sobre o tema. Até o momento, nenhum dos trabalhos selecionados na RSL aborda a Etapa 3 proposta pelo Robotics Multi-Annual Roadmap 2020 (ROBOTICS, 2016), onde afirma que os robôs serão capazes de se adaptar e melhorar modelos a partir de sua própria aprendizagem. Isso, portanto, demonstra uma área de pesquisa inexplorada até o momento e tornando-se o próximo passo na evolução do desenvolvimento de software robótico por meio do uso de modelos.

Artigo	Principal Área de Estudo	Níveis do Modelo	Transformação	Linguagem de Metamodelagem	Domínio	Suporte a autoadaptação	Suporte a ML
(FIGAT; ZIELIŃSKI, 2022)	MDE	PSM	M2M e M2T	Petri Nets	Robos Móveis	Não	Não
(WIGAND et al., 2017)	MDE	PIM e PSM	M2M e M2T	MPS Base Language	Robôs Humanoides	Não	Não
(ESTEVEZ et al., 2017)	MDE	PIM e PSM	M2M e M2T	UML	Braço Robótico	Não	Não
(BRUGALI; GHERARDI, 2016)	MDE	PIM e PSM	M2M e M2T	ECORE	Robôs Terrestres	Não	Não
(CICCOZZI et al., 2016)	MDE	PIM e PSM	M2M e M2T	ECORE	Robôs Aéreos e Aquáticos	Não	Não
(NORDMANN; WREDE; STEIL, 2015)	ML e MDE	PIM e PSM	M2M e M2T	MPS Base Language	Braço Robótico	Não	Sim
(NAGABANDI et al., 2018)	ML	-	-	-	Robôs Móveis	Sim	Sim
(DRAGULE; MEYERS; PELLICCIONE, 2017)	MDE	PIM	M2T	ECORE	Robôs Aéreos	Não	Não
(DJUKIĆ; POPOVIĆ; TOLVANEN, 2016)	MDE	PSM	M2M e M2T	ECORE	Braço Robótico	Não	Não
(STECK; LOTZ; SCHLEGEL, 2011)	MDE	PIM e PSM	M2M e M2T	ECORE	Robôs Móveis	Não	Sim
Desta Tese	ML e MDE	PIM e PSM	M2M e M2T	ECORE	Robôs Móveis	Sim	Sim

Tabela 1 – Comparação de Trabalhos Relacionados

4

Método Proposto

Este capítulo tem como objetivo apresentar o *Framework*, denominado RLoRDE (*Reinforcement Learning for Robotic model-Driven Engineering*), desenvolvido para auxiliar na construção de software robótico com adição de técnicas de aprendizado por reforço profundo. Na Seção 4.1 são apresentados as características do *Framework*. A Subseção 4.1.1 detalha o metamodelo proposto composto por conceitos da área de robótica com suporte a pontos de variação e aprendizado de máquina. Na Subseção 4.1.2 é demonstrado o metamodelo com os conceitos do middleware ROS.

4.1 Aprendizado por Reforço para Robótica com Engenharia Dirigida por Modelos

No campo da robótica a exploração automática é um problema importante dentro da navegação autônoma que significa que os robôs se movem incessantemente para construir todo o mapa do ambiente sem nenhum conhecimento prévio (LI; ZHANG; ZHAO, 2020). Diante disto, este *Framework* tem a capacidade de gerar boa parte da pilha de navegação para o treinamento em cenários criados para promover o aprendizado de tarefas utilizando o aprendizado por reforço em ambientes desconhecidos.

Utilizando-se do avanço da área de aprendizado de máquina, o *Framework* suporta três tipos de controladores robóticos baseados em RL além de poder ser aplicado em diversas arquiteturas. O *Framework* possibilita criar pontos de variação no controle, permitindo a utilização de controladores variados. Os pontos de variação permitem o

uso de controladores tradicionais em conjunto com controladores baseados em aprendizado de máquina para resolver alguns dos problemas apresentados no Capítulo 1, alguns dos quais incluem a necessidade de auto-adaptação, a necessidade de uma enorme quantidade de dados para treinar e mapear estados complexos.

O ambiente de treinamento é composto pelo ambiente no simulador Gazebo (GAZEBO, 2022) que possui o modelo do robô e o ambiente tridimensional vazio, sendo necessário a modelagem do cenário. Também é gerado o ambiente de treinamento OPENAI GYM, responsável pela execução do aprendizado por reforço. A interface de comunicação com os pacotes gerados é feita através do middleware ROS. A Figura 4 mostra uma visão geral da abordagem proposta.

O *Framework* desenvolvido é composto por um metamodelo robótico independente de plataforma, um metamodelo constituído para operação com ROS e um conjunto de transformadores capazes de converter de modelo em modelo (modelo robótico para modelo ROS) e transformar modelo em texto (modelo ROS para código).

Todo o código gerado para o treinamento dos controladores robóticos dentro do cenário disponibilizado pelo desenvolvedor é gerado de forma automática por meio dos transformadores que utilizam o modelos desenvolvidos a partir da ferramenta gráfica disponibilizada. Isso garante uma grande redução da carga de trabalho necessária para implementação de todo o código necessário para o treinamento do robô. Após o treinamento é possível utilizar os controladores para montar estratégias de solução de tarefas alocando os controladores treinados e definindo pontos de variação no modelo.

4.1.1 Metamodelo Robótico

O metamodelo robótico, apresentado na Figura 14, foi desenvolvido para ser independente de tecnologia, ou seja, o modelo desenvolvido a partir deste metamodelo pode ser utilizado para desenvolvimento em qualquer plataforma. No entanto, isso só é possível com a utilização de um transformador que recebe o modelo gerado e realiza as transformações necessárias para o modelo da tecnologia utilizada ou o código-fonte diretamente.

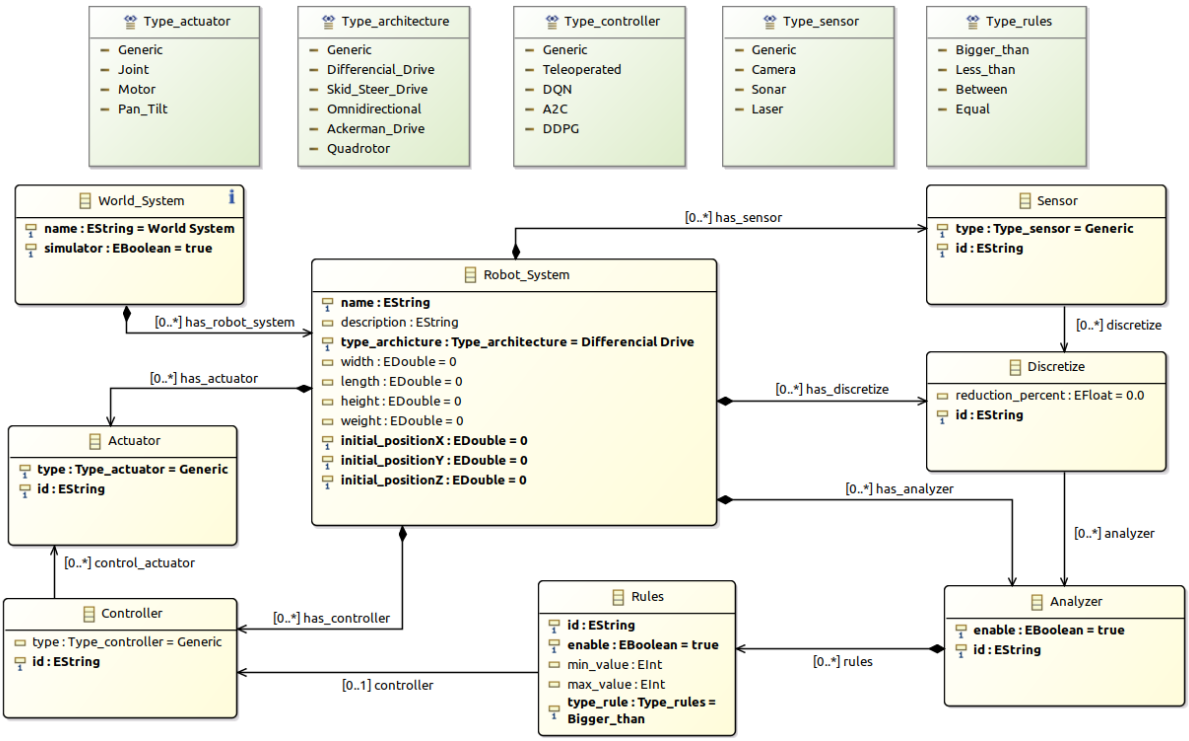


Figura 14 – Metamodelo do Sistema Robótico.

Este metamodelo possui cinco enumeradores para definição de constantes utilizadas na geração de código para o ambiente de treinamento pela ferramenta proposta. O enumerador *"Type_actuator"* refere-se aos tipos de atuadores suportados pela ferramenta que são *"Joint"*, *"Motor"* e *"Pan_Tilt"* que é aplicada à câmera. O tipo *"Genérico"*, também presente nos demais enumeradores, pode ser utilizado para adicionar tipos não suportados pela ferramenta.

O enumerador *"Type_architeture"* refere-se aos tipos de arquiteturas robóticas suportadas pela ferramenta. Estes consistem em *"Differential_Drive"*, *"Skid_Steer_Drive"*, *"Omnidirecional"* e os tipos *"Ackerman_Drive"* para carro autônomo e *"Quadrotor"* destinado a atender treinamentos com drones.

O enumerador *"Type_controller"* refere-se aos controladores disponíveis para uso no treinamento. Eles são compostos por alguns dos algoritmos mais utilizados no aprendizado por reforço profundo, como DQN, A2C e DDPG. O controlador *"Teleoperated"* e *"Generic"* também foi adicionado para adicionar controladores tradicionais ou personalizados pelo usuário.

O enumerador *"Type_sensor"* refere-se aos sensores disponíveis para uso em

sistemas robóticos. Estes são compostos por sensores de câmera, sonar e laser. O enumerador *"Type_rules"* foi criado para ser utilizado como regra de ativação do controlador e é composto por *"Bigger_than"* (maior que), *"Less_than"* (menor que), *"Between"* (entre) e *"Equal"* (igual). Essas regras serão utilizadas como gatilhos para ativação dos pontos de variação inseridos no modelo desenvolvido, incluindo a adição de camadas de controle com base nos dados coletados pelos sensores.

O metamodelo possui classes que representam os componentes dos sistemas robóticos desenvolvidos. Para possibilitar a geração de código para grupos de robôs, foi adicionada uma classe chamada *"World_System"*, onde todos os sistemas robóticos são acoplados. Esta classe possui os atributos *"name"* do tipo string que se refere ao identificador global do sistema dos robôs utilizados no treinamento e um atributo booleano para indicar o uso do simulador. A indicação do simulador é necessária pois após o treinamento é possível gerar o código para a aplicação real modificando o modelo ROS gerado e adicionando os pacotes referentes aos componentes reais do sistema como o robô utilizado, sensores e atuadores.

A classe *"Robot_System"* representa sistemas robóticos criados para simulação e treinamento. Este é composto pelo *"name"*, *"description"* e pelo enumerador *"type_architecture"*. Também fazem parte dos atributos os dados de dimensão dos robôs com *"width"*, *"length"*, *"heigth"* e *"weight"*. As dimensões juntamente com o *"tipo_architecture"* são definidas para criar uma representação visual 3D genérica do robô na simulação, composta por um cubo e as rodas, de acordo com as propriedades definidas nos atributos. Esta representação 3D pode ser substituída pela modelagem 3D do modelo do robô que será utilizada em ambiente real no modelo ROS gerado a partir desta definição. Os atributos *"initial_positionX"*, *"initial_positionY"* e *"initial_positionZ"* consiste nas coordenadas de inicialização do robô no ambiente de simulação Gazebo.

Cada sistema robótico pode conter vários sensores, atuadores e controladores com seu tipo e identificação única. A classe Discretize foi adicionada para servir como filtro para os dados do sensor. É composta pelo atributo *"reduction_percent"* que define o percentual de redução dos dados do sensor e *"id"* que é seu identificador único. Essa redução faz o papel de discretizador dos sensores utilizados no *Framework*.

Uma das principais contribuições desta pesquisa é a possibilidade de definições de pontos de variação implantados no software robótico. Para isso foi adicionado, como recurso, a classe *"Analyzer"* que é responsável por receber os dados discretizados dos sensores e possui os atributos *"id"* para identificação única e *"enable"* para ativação e desativação durante a simulação. Cada analisador definido no modelo pode conter regras que estão diretamente relacionadas aos controladores. Cada regra tem um atributo único *"id"*, *"enable"* para ativação, *"type_rule"* para definir o tipo de regra aplicada e os valores *"min_value"* e *"max_value"* para o tipo de regra.

A utilização do *"Analyzer"* e as regras definidas são aplicadas aos controladores disponíveis e customizados de acordo com a necessidade do usuário. Dessa forma, é possível criar pontos de variação no software do robô para interação com ambientes onde apenas o uso do aprendizado por reforço não é suficiente. Outra possibilidade de uso dos analisadores é mesclar modelos de aprendizado de máquina com tarefas de distintas a serem executadas, como no exemplo dado na Seção 1.6, onde para cada tipo de estacionamento seria aplicado um modelo de rede neural treinado previamente. Outra vantagem dessa abordagem é poder inserir camadas de segurança para reduzir os riscos envolvidos nas tarefas realizadas em ambientes reais.

4.1.2 Metamodelo ROS

O metamodelo ROS, mostrado na Figura 15, foi desenvolvido para integração com o Middleware ROS. O *Framework* ROS privilegia uma abordagem de desenvolvimento de software que consiste em projetar componentes que implementem recursos robóticos comuns e reutilizáveis (BRUGALI; GHERARDI, 2016).

Assim como o metamodelo robótico, ele possui uma classe chamada *"World_ROS"* que representa o mundo onde os sistemas robóticos são organizados. Essa classe possui os atributos *"world_name"* e *"simulator"* para indicar a geração de código para o simulador Gazebo. A classe *"RobotSystemROS"* representa os sistemas robóticos incluídos no treinamento e possui os mesmos atributos que a classe *"Robot_System"* do metamodelo do sistema robótico.

class diagram.png

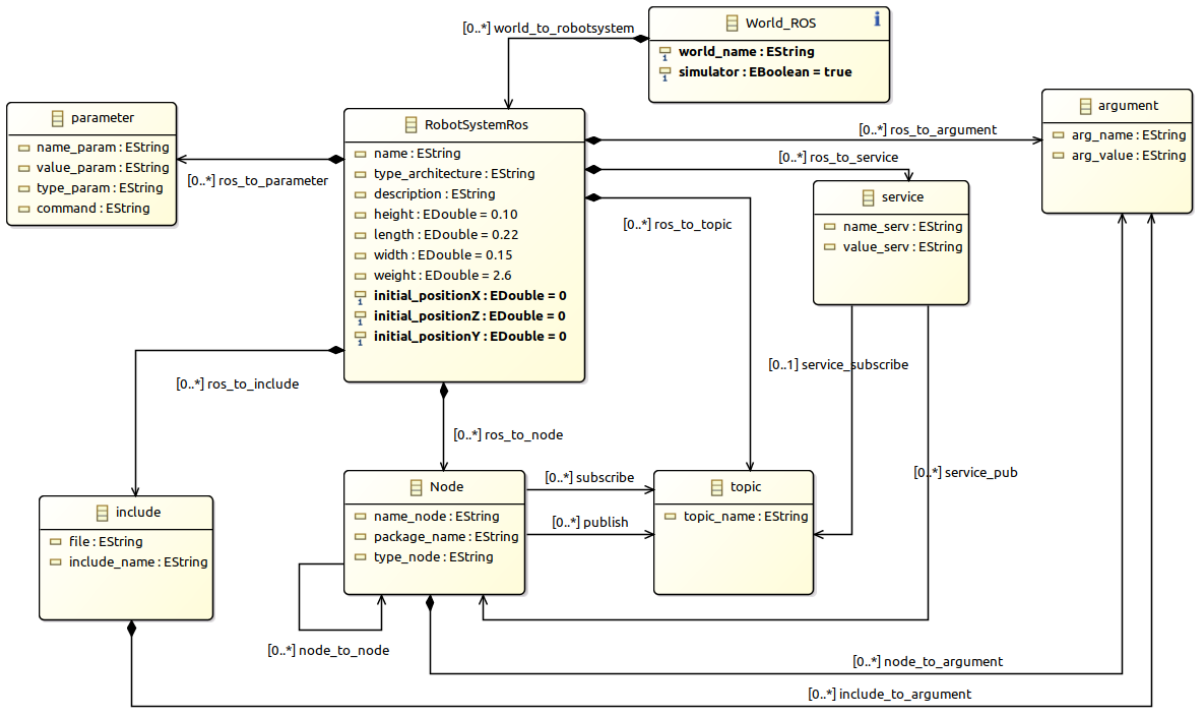


Figura 15 – Metamodelo ROS.

As demais classes pertencentes ao metamodelo representam a infraestrutura necessária para implementação do código no *Framework* ROS. A classe *"Node"* representa processos que realizam computação. Como o ROS foi projetado para ser modular, o nó compreende o software responsável por realizar alguma ação no sistema, como ler dados de sensores ou enviar comandos de controle. Em geral, a arquitetura de um robô é formada por vários nós. Esta classe possui os atributos *"name_node"*, *"package_name"* e *"type_node"* que são necessários durante a chamada em tempo de execução para o sistema ROS.

Os nós se comunicam passando mensagens. Uma mensagem é simplesmente uma estrutura de dados. Tipos primitivos padrão (inteiro, ponto flutuante, booleano, etc.) são suportados, assim como arrays de tipos primitivos. As mensagens são roteadas por meio de um sistema de transporte com semântica de publicação/assinatura. Um nó envia uma mensagem publicando-a em um determinado *"tópico"*. O tópico é um nome usado para identificar o conteúdo da mensagem. Um nó interessado em um determinado tipo de dados se inscreverá no tópico apropriado. Pode haver vários editores e assinantes simultâneos para um único tópico e um único nó pode publicar

e/ou assinar vários tópicos.

O modelo de publicação/assinatura é um paradigma de comunicação muito flexível, mas seu transporte unidirecional para muitos não é apropriado para interações de solicitação/resposta, que geralmente são necessárias em um sistema distribuído. A requisição/resposta é feita via *"serviço"*, definido por um par de estruturas de mensagens: uma para a requisição e outra para a resposta. Um nó de provisão oferece um serviço com um nome, e um cliente utiliza o serviço enviando a requisição mensagem e aguardando a resposta.

O servidor *"parameter"* permite que os dados sejam armazenados por chave em um local central. A classe *"include"* faz parte do *Framework* ROS e permite a composição de softwares já desenvolvidos através de pacotes. Esses pacotes podem ser incluídos para formar o sistema de software completo do robô. Os argumentos, da classe *"argument"*, são parâmetros que podem ser passados para nós ou inclusões quando são chamados.

4.1.3 Transformações

Um dos benefícios promovidos pelo uso do MDE é a possibilidade de geração de código fonte. Isso é possível usando os métodos de transformação de modelo para modelo (M2M) e modelo para texto (M2T). No RLoRDE usamos a transformação M2M para transformar o modelo do sistema robótico para o modelo ROS e a transformação M2T para transformar o modelo ROS no código-fonte.

A transformação M2M foi realizada utilizando a Linguagem de Transformação Atlas (ATL - *Atlas Transformation Language*) (JOUAULT et al., 2006) que é uma linguagem de transformação de modelo e toolkit presente no Eclipse. ATL fornece maneiras de produzir um conjunto de modelos de destino a partir de um conjunto de modelos de origem. A Figura 16 apresenta um trecho de código na linguagem ATL.

A transformação M2T foi realizada utilizando a ferramenta Acceleo (ACCELEO, 2022) que também está disponível no Eclipse. Acceleo é uma tecnologia baseada em modelo que inclui ferramentas para criar geradores de código personalizados. Permite

```
rule worldsystem2worldsystemROS{
    from
        rs: Robot_System!World_System(thisModule.getSimulated)
    to
        wsr: Robot_SystemROS!World_ROS (
            world_name <- rs.name,
            simulator <- rs.simulator,
            world_to_robotsystem <- Robot_SystemROS!RobotSystemRos.allInstances()
        ),
        gazebo: Robot_SystemROS!include (
            include_name <- 'gazebo',
            file <- '$(find gazebo_ros)/launch/empty_world.launch',
            include_to_argument <- arg_world,
            include_to_argument <- arg_gui,
            include_to_argument <- arg_paused,
            include_to_argument <- arg_use_time,
            include_to_argument <- arg_headless,
            include_to_argument <- arg_debug
        ),
        arg_world: Robot_SystemROS!argument (
            arg_name <- 'world_name',
            arg_value <- '$(find '+thisModule.getNameSystem+'_gazebo)/worlds/'+thisModule.getNameSystem+'.world'
        ),
        arg_gui: Robot_SystemROS!argument (
            arg_name <- 'gui',
            arg_value <- 'true'
        ),
```

Figura 16 – Trecho de Código ATL.

a produção automática de qualquer tipo de código fonte a partir de qualquer fonte de dados disponível no formato EMF.

Quando a transformação para o código é realizada, cinco pacotes são criados no sistema ROS. Um pacote ROS é simplesmente um diretório descendente de ROS_PACKAGE_PATH (*ROS Environment Variables*). Esses pacotes são *"gym"*, *"agents"*, *"control"*, *"description"* e *"gazebo"*. O pacote *"gym"* contém o ambiente de treinamento de aprendizado por reforço. Gym é um kit de ferramentas para desenvolver e comparar algoritmos de aprendizado por reforço.

O pacote *"agents"* contém a implementação de controladores baseados em aprendizado por reforço. Essas implementações são baseadas nos algoritmos DQN, A2C e DDPG. O pacote *"control"* contém os controladores personalizados pelo usuário ou ações de controle que podem ser acionadas pelo dados de detecção do robô.

O pacote *"description"* possui o arquivo no Formato Unificado de Descrição do Robô (URDF - Unified Robot Description Format), que é um formato XML para representação de um modelo de robô. O arquivo gerado inicialmente é genérico e representa apenas as medidas do robô a ser utilizado. O usuário é capaz de alterar este arquivo com o modelo real do robô a ser usado. O pacote "gazebo"contém o arquivo de inicialização para o ambiente ROS e o simulador Gazebo.

5

Resultados Experimentais

Neste Capítulo são apresentados os detalhes e resultados dos experimentos realizados. O experimento para avaliação do *Framework* desenvolvido é relatado na Seção 5.1. Na Seção 5.2 são demonstrados os resultados alcançados com o uso da ferramenta proposta.

5.1 Experimento

Para demonstrar a aplicabilidade do *Framework*, desenvolvemos um experimento para validar a utilização da ferramenta em um caso real e avaliar a capacidade de autoadaptação de uma rede neural no cenário simulado e real. O robô utilizado no treinamento e no ambiente real foi um Pioneer 3AT da Mobile Robots. Um sensor laser 2D Lidar modelo LMS100 da Sick foi usado para perceber o ambiente pelo robô.

A missão do robô consistiu na exploração do cenário utilizando arquitetura reativa com objetivo de localização da saída. Um modelo robótico foi criado seguindo as definições do metamodelo e utilizando a ferramenta gráfica proposta. A Figura 17 apresenta a ferramenta disponível no *Framework* onde consta o modelo desenvolvido no lado esquerdo e os componentes da ferramenta disponíveis no lado direito. O modelo é composto pelos componentes do laser para leitura do ambiente, discretização para o filtro de dados do laser e o analisador que contém duas regras para escolha dos controladores. Os controladores utilizados para este experimento foram DQN para navegação de cenários e um controlador simples desenvolvido para evitar obstáculos. As duas regras de ativação do controlador foram baseadas na distância disponível

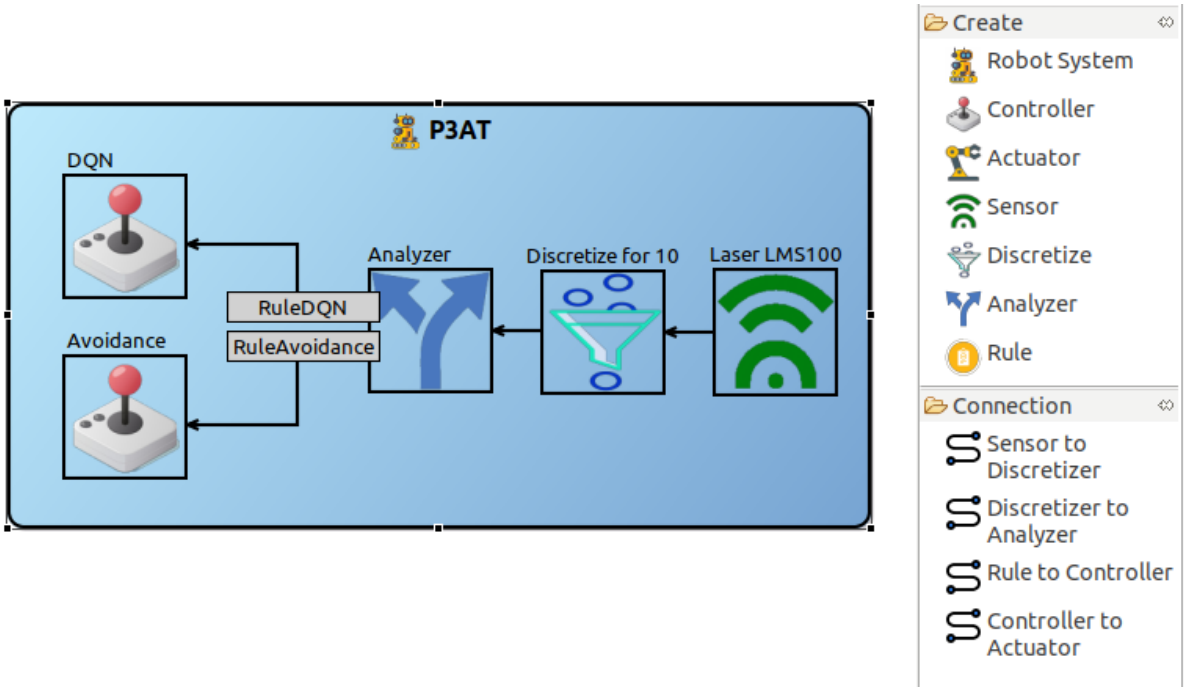


Figura 17 – Ferramenta gráfica e modelo robótico utilizado no experimento.

nos dados do laser. Quando a distância do laser era superior a 30 cm, os comandos eram enviados pelo controlador DQN, caso contrário, os comandos eram enviados pelo controlador Avoidance que enviava comandos ao robô para se afastar do obstáculo.

Após a criação do modelo do sistema robótico, a transformação para o modelo robótico ROS foi realizada conforme apresentado na Figura 18 de forma automática pela ferramenta. O modelo ROS gerado (Figura 19) é composto pelos componentes necessários para sua execução com o simulador Gazebo. Este simulador é muito utilizado na robótica. O próximo passo foi realizar a transformação para o código, conforme Figura 20, onde foram criados os diretórios e o código fonte do ambiente GYM para execução do treinamento, os diretórios ROS que compõem a representação do robô em URDF (Unified Robot Description Format), que compreende um XML de especificação para descrever os robôs, controladores e arquivos de execução do Gazebo. Para este experimento usamos a representação 3D real do robô Pioneer 3AT em formato URDF. A maior parte do código gerado está na linguagem Python.

Neste experimento, o agente foi treinado apenas no Cenário 1 (ver Figura 21-a). O modelo de rede neural treinado foi então submetido a uma série de tentativas de concluir a missão onde foram feitas alterações, aumentando a complexidade dos cenários. O modelo treinado no Cenário 1 foi obrigado a se adaptar às mudanças do

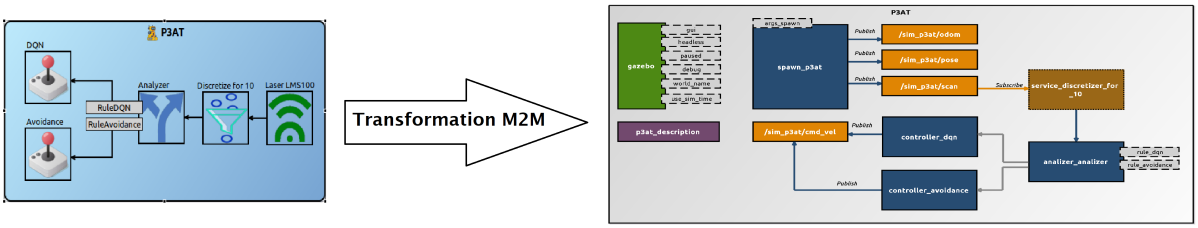


Figura 18 – Transformação do Modelo do Robô para o Modelo ROS.

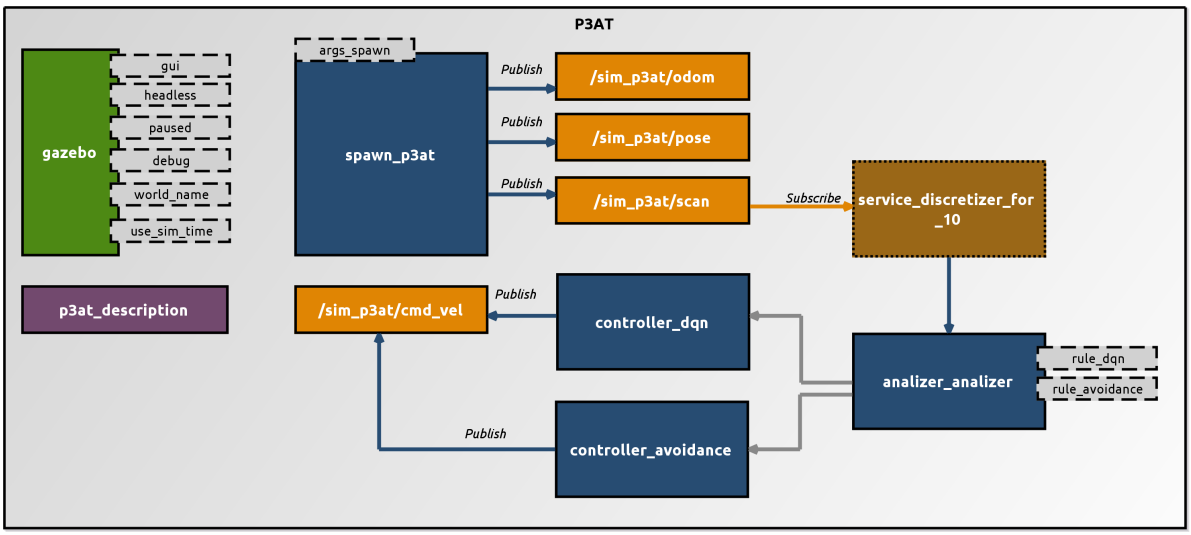


Figura 19 – Modelo do Robô ROS.

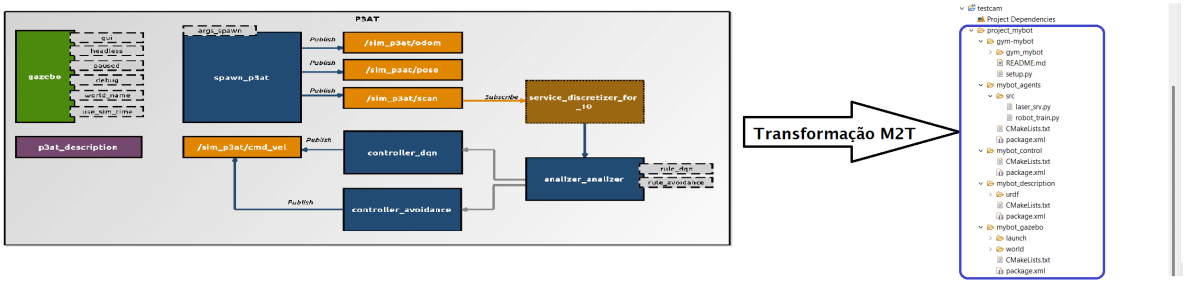


Figura 20 – Transformação do Modelo ROS para o Código.

ambiente para cumprir seu objetivo nos demais cenários. O cenário de treinamento consistia em um circuito onde o robô é estimulado a aprender movimentos básicos para se locomover e sair do ambiente.

A ferramenta proposta nesta pesquisa é capaz de gerar ambientes de treinamento para situações mais complexas como carros autônomos e até drones, demonstrando sua aplicação em diversas áreas da robótica móvel. No entanto, para a avaliação experimental, foi escolhido o robô de tração nas quatro rodas modelo Pioneer 3AT. O motivo desta escolha é que era o modelo de robô disponível e, além disso, era desejável comparar

ambientes simulados com ambientes reais.

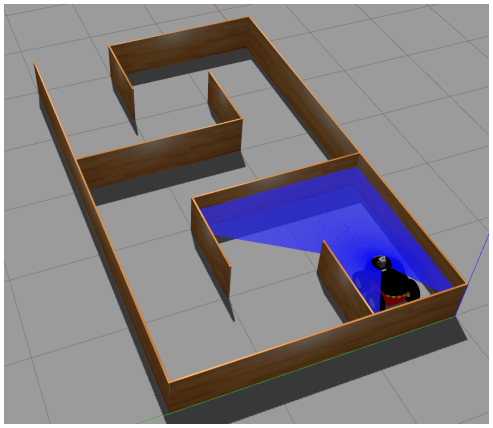
Os movimentos, compostos por velocidade linear no eixo X e velocidade angular no eixo Z, foram realizados diretamente a partir dos dados filtrados dos sensores, neste caso um laser. A Figura 21 apresenta os cenários 3D no simulador Gazebo onde o agente foi testado. O treinamento ocorreu apenas no cenário 1 (Figura 21-a). A Figura 22 apresenta os cenários reais utilizados nos testes. O tamanho do cenário compreendeu um retângulo de 5x3m ($15m^2$).

Para avaliar o desempenho do robô utilizamos métricas propostas por (LAMPE; CHATILA, 2006). As métricas para a missão de navegação foram a velocidade média percorrida, distância, duração da missão e taxa de sucesso da missão. As mudanças no cenário seguiram critérios de complexidade gradual com base no cálculo da complexidade global variando de 0 a 1 e são mostradas na Figura 21. Essa complexidade é calculada usando uma grade de ocupação binária conforme apresentada em (LAMPE; CHATILA, 2006). Nos cenários 1, 2 e 3 não houve alteração na complexidade, mas houve alteração na disposição dos cenários 2 e 3. Além da alteração na disposição, também foi adicionado uma bifurcação. No cenário 5, além do aumento da complexidade, foram adicionados 2 obstáculos estáticos. Foram realizadas 10 voltas em cada cenário para calcular a média de acordo com as métricas utilizadas.

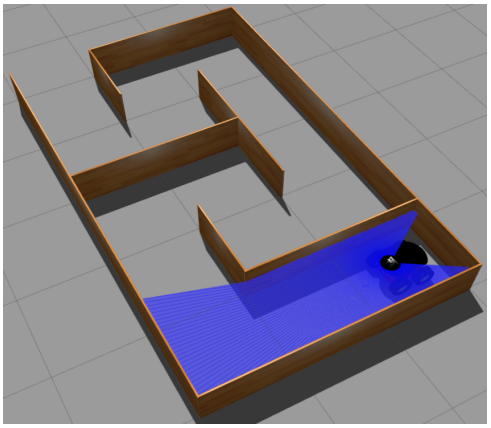
5.1.1 Ambiente de Aprendizagem

Gym é um kit de ferramentas para desenvolver e comparar algoritmos de aprendizado por reforço (BROCKMAN et al., 2016). A biblioteca do Gym é uma coleção de problemas de teste, conhecidos como ambientes, que podem ser usados para trabalhar seus algoritmos de aprendizado por reforço. Por meio do *Framework* RLoRDE, um ambiente customizado é criado automaticamente usando o modelo do sistema robótico modelado anteriormente. Este ambiente atende a todos os requisitos exigidos pelo kit de ferramentas Gym para sua execução na biblioteca numérica TensorFlow (TensorFlow, 2022).

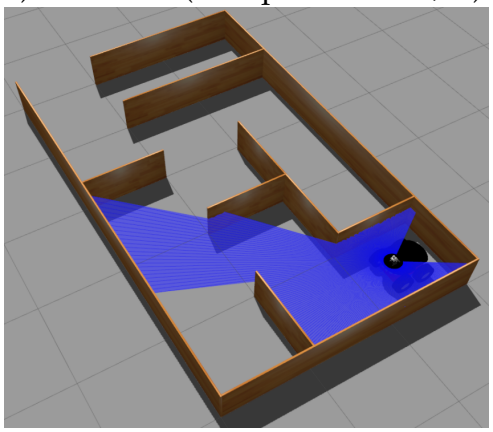
A ferramenta Gym foi utilizada como um dos núcleos centrais da proposta. Por



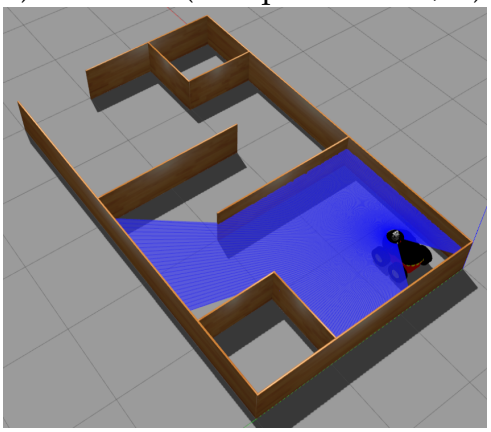
a) Cenário 1 (Complexidade: 0,34).



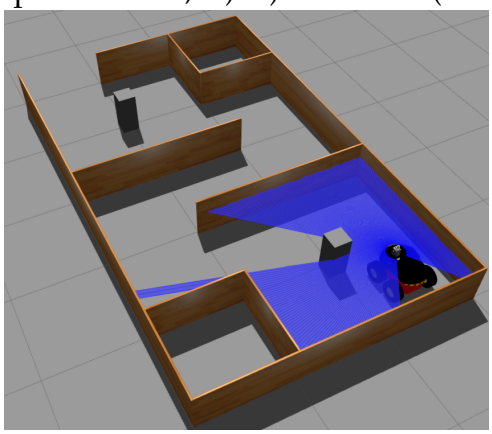
b) Cenário 2 (Complexidade: 0,34).



c) Cenário 3 (Complexidade: 0,34).

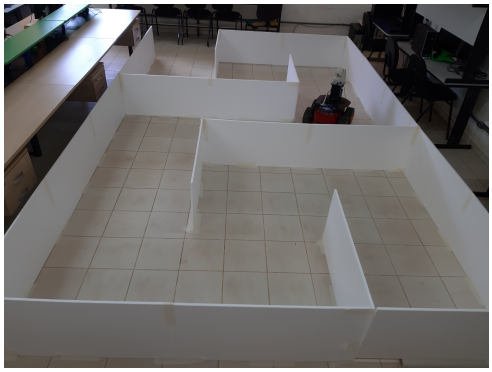


d) Cenário 4 (Complexidade: 0,58).



e) Cenário 5 (Complexidade: 0,63).

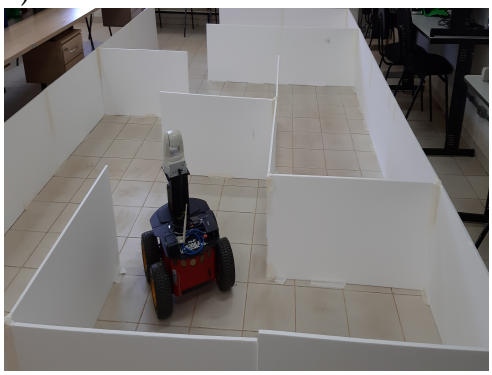
Figura 21 – Cenários no Simulador Gazebo.



a) Cenário Real 1.



b) Cenário Real 2.



c) Cenário Real 3.



d) Cenário Real 4.



e) Cenário Real 5.

Figura 22 – Cenários Reais.

meio da ferramenta é possível criar ambientes de treinamentos personalizados. Portanto, todo o conjunto de funcionalidades disponíveis na API está disponível para uso com o *Framework*. Desta forma o RLoRDE utiliza os principais conceitos da ferramenta para criar o ambiente de treinamento de RL para o problema a ser solucionado.

5.1.2 Representação de Estados e Ações

O espaço de observação neste experimento é baseado no componente sensor e na discretização utilizada no modelo do sistema robótico, neste caso composto por 2% (10 feixes) do total de 540 feixes de laser disponíveis no LMS100 utilizado no experimento . A faixa de leitura do laser utilizada nos testes foi de 0 a 6 m, considerando o tamanho do cenário.

O espaço de ação para este tipo de robô foi definido como um par representando os valores de velocidade linear no eixo X e velocidade angular no eixo Z. A faixa de velocidade linear foi definida de -0,1 m/s e limitada a 0,2 m/s. Essa limitação de velocidade foi necessária devido à limitação de espaço do cenário do experimento. Velocidades mais altas anularam a possibilidade de executar o controlador Avoidance devido à energia cinética durante os testes em ambiente real. A faixa de velocidade angular foi ajustada de -0,5 a 0,5 m/s.

5.1.3 Função de Recompensa

Reconhecer quando o alvo é atingido pelo agente pode ser um desafio em muitos problemas de RL, pois requer acesso a informações específicas sobre o ambiente. Neste problema o agente robótico executa uma ação a_t em um estado s_t recebe uma recompensa r_t representada pelo objetivo alcançado, objetivo não alcançado (no caso de um toque na parede do cenário ou número de ações máximo excedida). Portanto, o aprendizado do agente é direcionado pela função de recompensa R. A implementação da recompensa $R(s_t, a_t, s'_{t+1})$ realizando uma ação a_t no estado atual s_t levando a um novo estado s'_{t+1} pode ser descrita da seguinte forma:

$$R(s_t, a_t, s'_{t+1}) = \begin{cases} 100 & \textit{if isMissionFinished()} \\ -100 & \textit{if isMissionFailure()} \\ 0 & \textit{if isVisited()} \\ Distance() & \textit{otherwise} \end{cases} \tag{5.1}$$

Durante o treinamento cada episódio pode ser finalizado *isMissionFinished()*

quando o agente alcança seu objetivo, recebendo uma recompensa de +100. Ao entrar em contato com uma parede do cenário uma falha na execução do episódio *isMissionFailure()* é registrada e o agente recebe uma punição no valor de -100 pontos. Se o agente ultrapassa o número máximo de passos executados para tentar solucionar o problema, que foi definido como 1500 interações, também é entendido como uma falha e o robô é punido com -100 pontos. Cada etapa compreende um intervalo de tempo no qual os comandos são enviados ao robô. Nesse caso, esse intervalo foi de 0,2 segundos.

Uma recompensa parcial *Distance()* também foi criada durante a execução do treinamento para agilizar o treinamento. Essa recompensa foi baseada na distância percorrida em relação à posição no instante anterior a cada ação. Dessa forma, foi possível amenizar um dos maiores problemas do aprendizado por reforço, que são as recompensas esparsas. O robô recebe a recompensa apenas em locais onde ainda não passou, o que favorece o robô explorar o ambiente.

5.2 Resultados

O treinamento, que ocorreu no primeiro cenário, foi realizado utilizando os códigos gerados para simulação no ambiente Gazebo, ROS e GYM. Utilizamos a biblioteca numérica TensorFlow para os cálculos e acompanhamento do resultado, que pode ser visto na Figura 23. O treinamento durou 24h e 46m. A passagem do tempo no simulador foi acelerada em 10x o tempo normal. Essa possibilidade de aceleração do tempo no simulador é um dos benefícios alcançados com a utilização do *Framework* proposto. 11.290 episódios foram realizados para completar o treinamento. Os resultados e o *Framework* podem ser observados no repositório no GitHub (SILVA, 2022).

Considerando as métricas para missão de navegação propostas por Lampe e Chatila (2006), avaliamos a velocidade média, distância percorrida, duração da missão e taxa de sucesso da missão comparando a execução em um ambiente simulado e real. Em cada cenário o robô teve 10 tentativas para atingir seu objetivo e completar a missão. A taxa de sucesso, apresentada na Figura 24, mostra que o robô conseguiu atingir a meta todas as vezes no primeiro cenário tanto no ambiente simulado quanto no real.

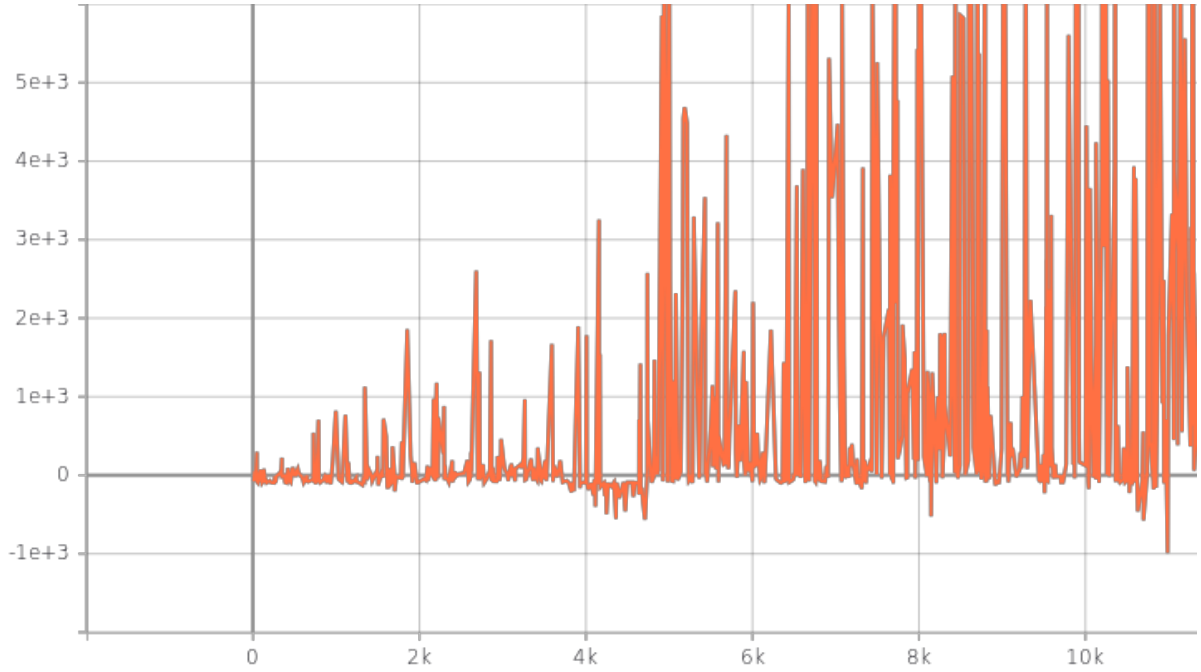


Figura 23 – Total de Recompensas por Episódio.

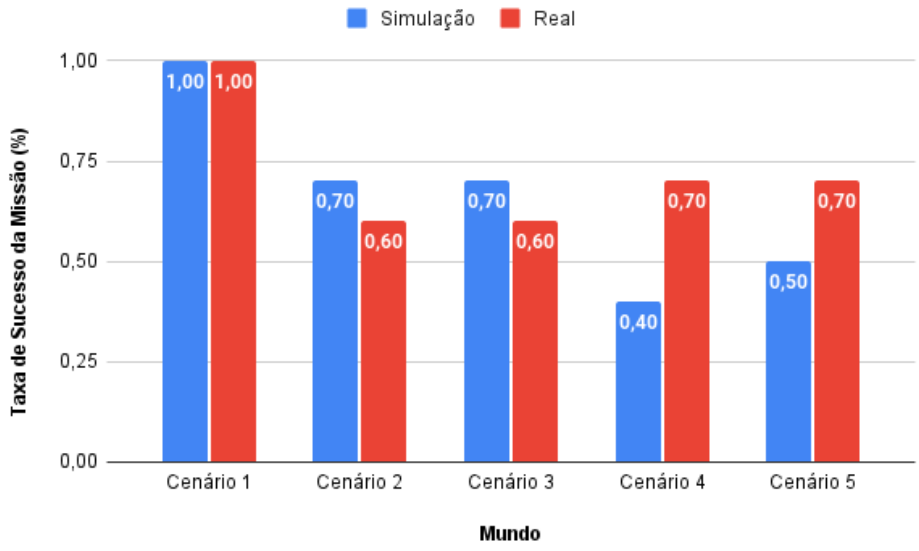


Figura 24 – Taxa de Sucesso da Missão.

Isso era esperado, pois o treinamento ocorreu nesse cenário. Nos cenários 2 e 3 a taxa de sucesso foi maior em ambiente simulado, porém, apenas uma volta de diferença. Nos cenários 4 e 5 houve vantagem nos testes realizados em ambiente real. Com isso, é possível perceber que tanto no ambiente simulado quanto no real, o robô, por meio da rede neural, foi capaz de se adaptar às mudanças ocorridas no ambiente.

O resultado da comparação da velocidade média, calculada após 10 voltas em cada um dos cenários, pode ser visto na Figura 25. Neste caso, é possível notar que

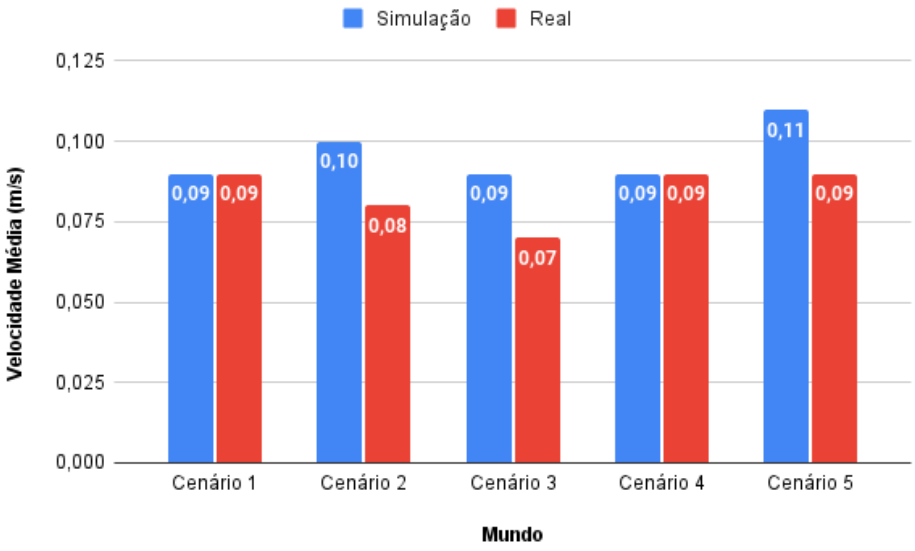


Figura 25 – Velocidade Média.

a aceleração média tende a ser maior em um ambiente simulado. Isso é facilmente explicado pelo fato de se tratar de um ambiente totalmente estruturado e onde o robô não sofre influências desse ambiente ou de si mesmo, como, por exemplo, a calibração dos pneus, a aderência do piso ou até mesmo o desgaste de seus componentes mecânicos.

Outro detalhe que pode ser observado é que no cenário 1 do ambiente real o robô teve desempenho superior em relação ao ambiente simulado. Isso aconteceu porque era o cenário onde o robô era treinado e, portanto, já tinha plena consciência dos movimentos que deveria realizar com base nos dados sensoriais. Em cenários onde houve modificação do ambiente, foi perceptível uma hesitação de movimentos quando os dados do sensor não foram reconhecidos como padrões já aprendidos. No entanto, mesmo nesses casos, na maioria das vezes, o robô conseguiu seguir em frente.

Em relação ao resultado da distância média percorrida, apresentado na Figura 26, é possível notar que houve uma grande variação no resultado entre os cenários. Nos cenários 1 e 2 o robô percorreu uma distância maior no ambiente simulado. Isso se deve ao fato de que não há grandes diferenças entre os dois cenários além do *layout*.

A diferença entre os cenários 3 e 4 pode ser explicada pelo fato de haver uma mudança significativa no cenário. Por exemplo, no cenário 3, uma bifurcação foi adicionada, o que resultou em outro caminho que o robô poderia seguir. Esse caminho não

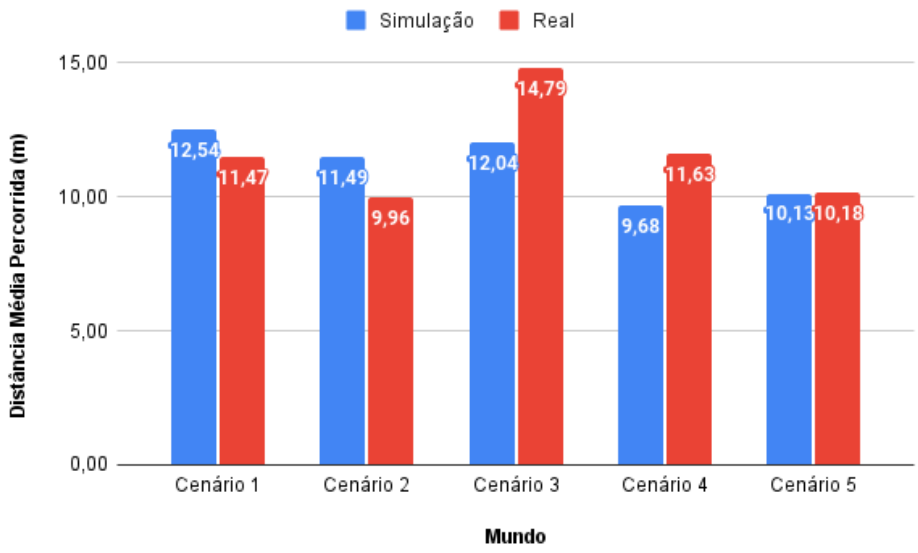


Figura 26 – Distância Média Percorrida.

teve saída e em alguns casos de sucesso na tarefa o robô conseguiu retornar. No entanto, isso acabou contribuindo para o aumento da distância percorrida. No caso do cenário 4, houve um ponto de hesitação no início dos testes em que o robô continuou se movendo para frente e para trás em um curto espaço por um tempo. Isso pode ser visto na Figura 29d.

No caso da duração da missão, é possível ver na Figura 27 que houve uma grande diferença nos três primeiros cenários. Isso pode ser explicado pela maior inércia do robô no ambiente real, o que dificultou sua trajetória em ambiente reduzido. No caso do cenário 4 em ambiente simulado, houve 3 casos em que o robô levou mais de 8 minutos para atingir o objetivo. Isso também foi causado pelo ponto de hesitação no início do teste, onde os dados do sensor eram diferentes dos dados aprendidos no cenário 1. Isso contribuiu para essa discrepância no resultado. O mesmo ocorreu no cenário 5.

Para complementar os resultados, as Figuras 28 e 29, apresentam os caminhos percorridos pelo robô durante a execução da missão. Cada Figura apresenta os 10 tentativas em cada cenário pelo robô. A legenda apresenta o número da tentativa, sua cor definida na trajetória percorrida e a forma de uma bolinha que representa uma missão bem-sucedida ou um x que representa uma tentativa não concluída.

Na Figura 28a é possível notar que todas as rodadas foram concluídas com

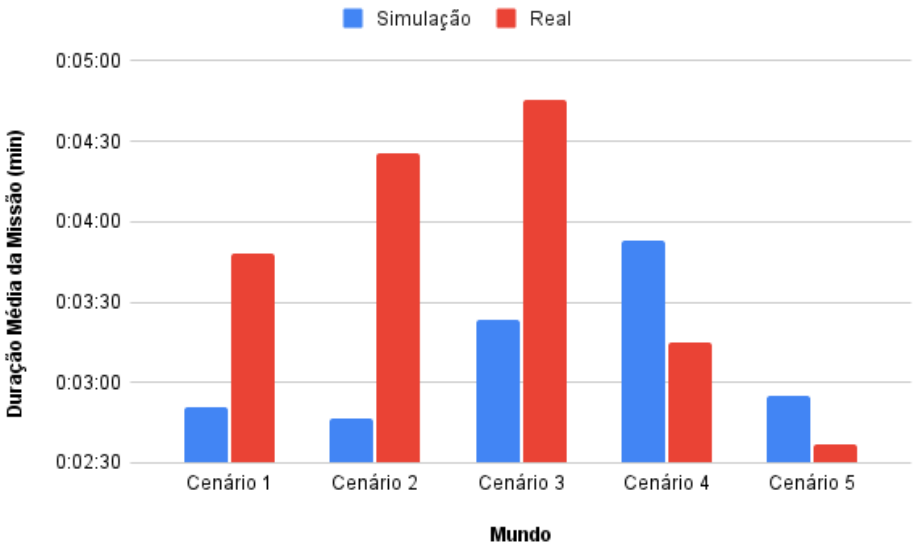
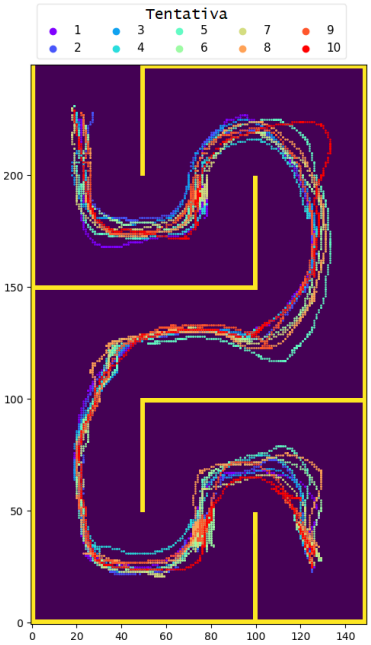


Figura 27 – Duração Média da Missão.

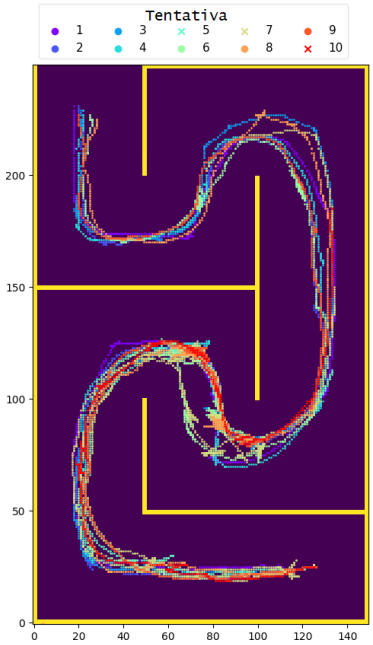
sucesso. Na Figura 28b apenas as rodadas 5, 7 e 10 não foram concluídas com sucesso. O mesmo número de rodadas inacabadas, 3, 5 e 6, pode ser visto na Figura 28c. A Figura 28d mostra que o robô teve maior dificuldade para completar a missão, sendo executado com sucesso nas rodadas 1, 2, 6 e 8. A Figura 28e mostra uma pequena melhora em relação ao cenário anterior, com 50% de aproveitamento. Neste, os casos de sucesso foram as rodadas 1, 2, 7, 8 e 9.

Nos testes realizados em ambiente real, Figura 29, é possível notar algumas irregularidades no caminho percorrido, como o caminho sobre a parede ou obstáculo no caso da Figura 29e . Isso foi causado pelo acúmulo de erro de posição no cálculo da odometria do robô. Apesar dessas irregularidades, é notória a capacidade adquirida pelo robô para atingir seu objetivo.

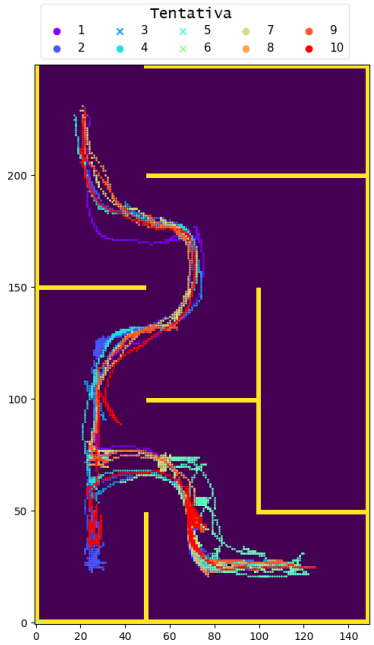
Figure 29a mostra o resultado positivo de todas as tentativas de execução da missão. Na Figura 29b é possível ver que houve o mesmo número de falhas que ocorreram no cenário simulado, sendo elas nas rodadas 5, 8 e 10. A Figura 29c mostra que houve falha apenas nas rodadas 4, 5, 7 e 10. A Figura 29d mostra que o robô teve uma boa habilidade para realizar a tarefa, executando com sucesso a maioria das rodadas. Apenas nas rodadas 4, 6 e 7 houve uma falha. O mesmo ocorreu no cenário 5 (Figura 29e), com casos de falha ocorrendo nas rodadas 2, 4 e 8.



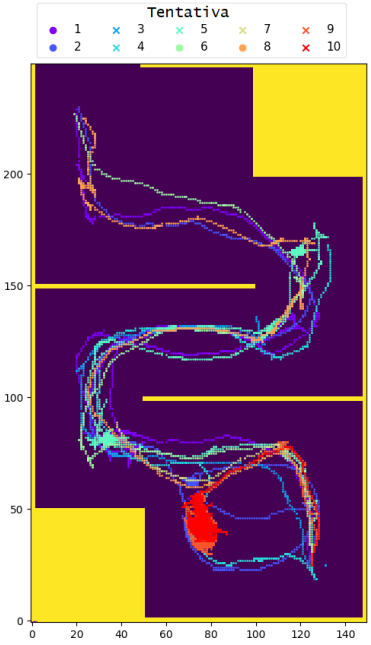
(a) Trajetória do Robô no Cenário 1.



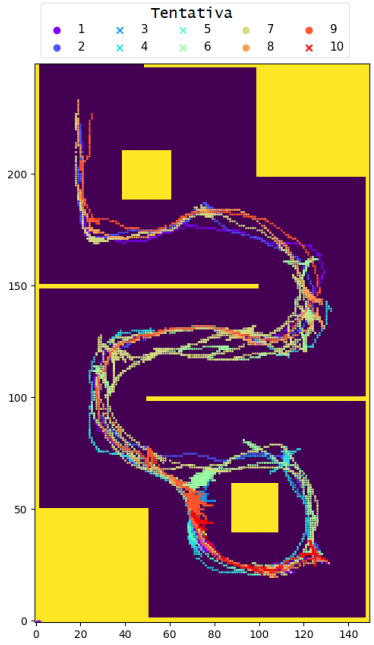
(b) Trajetória do Robô no Cenário 2.



(c) Trajetória do Robô no Cenário 3.

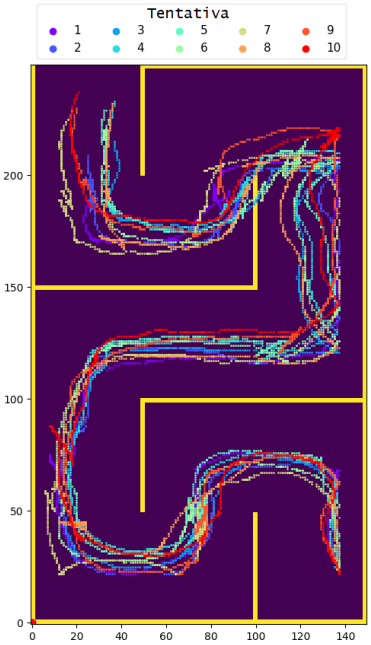


(d) Trajetória do Robô no Cenário 4.

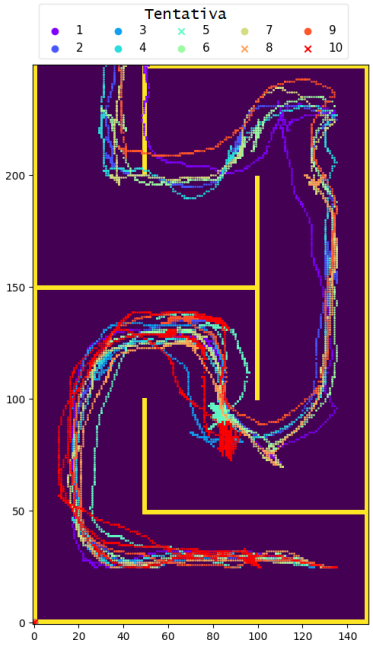


(e) Trajetória do Robô no Cenário 5.

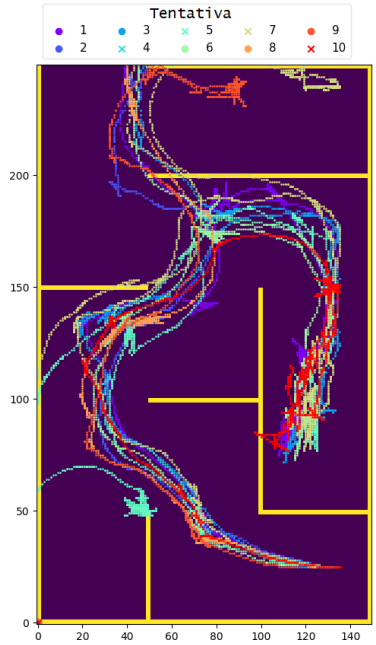
Figura 28 – Trajetória Percorrida nos Testes no Simulador.



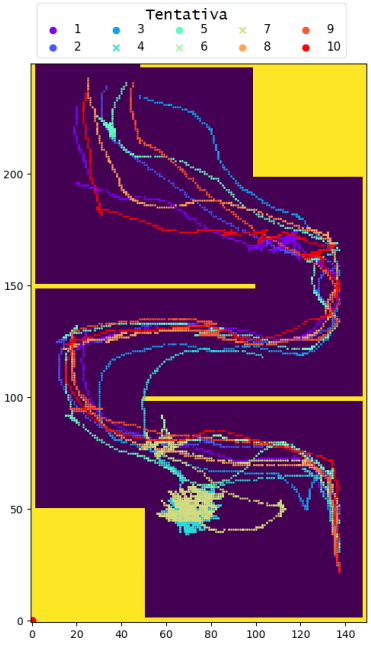
(a) Trajetória do Robô no Cenário 1.



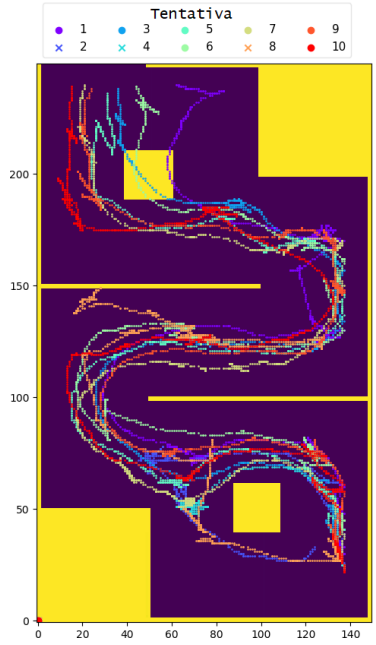
(b) Trajetória do Robô no Cenário 2.



(c) Trajetória do Robô no Cenário 3.



(d) Trajetória do Robô no Cenário 4.



(e) Trajetória do Robô no Cenário 5.

Figura 29 – Trajetória Percorrida nos Testes no Ambiente Real.

6

Conclusão

Neste Capítulo são apresentadas as considerações finais a respeito da pesquisa realizada neste trabalho. Também são apresentadas, na Seção 6.3, as limitações do *Framework* além dos possíveis trabalhos futuros na Seção 6.4.

6.1 Considerações Finais

A capacidade de autonomia dos robôs é um aspecto fundamental para a evolução da robótica como área do conhecimento e a propagação de seus benefícios para a sociedade. O aprendizado de máquina trouxe uma enorme expectativa na resolução de problemas enfrentados pela robótica clássica. No entanto, sua aplicação em ambientes reais ainda carece de estudos aprofundados em relação a situações inesperadas, que ainda são um problema evidente dado o alto grau de incerteza no ambiente.

A pilha de navegação do robô, que consiste em módulos de autolocalização, mapeamento, planejamento global e local, comumente usados em robótica, não são suficientes para aplicação em ambientes altamente dinâmicos, e isso deteriora seu desempenho. Neste tipo de aplicação, muitas vezes é necessário usar mapas estáticos e a programação *ad hoc* está sujeita a erros. Além disso, eles precisam de uma etapa de reconhecimento do ambiente para gerar os mapas, o que gera atrasos e aumenta os custos.

A reutilização de código é outro fator que dificulta a redução de custos e o uso de robôs. A codificação do robô normalmente é feita para atender especificidades de hardware e destina-se a executar poucas tarefas em um ambiente limitado. Caso seja

necessária sua aplicação em outros cenários, mesmo que compartilhem algumas características semelhantes, há a necessidade de reprogramação. Baseando-se neste contexto, este trabalho teve como questão de pesquisa: "**Agregar aspectos de aprendizado por reforço em uma família de metamodelos de desenvolvimento de software robótico para que o robô possa utilizar o conhecimento adquirido proporcionará uma melhor adaptação ao ambiente inserido?**".

Diante disso, nesta Tese propomos um *Framework* denominado RLoRDE para geração de ambientes de treinamento para robôs móveis com aprendizado por reforço. Este *Framework* é o primeiro que emprega na modelagem do software robótico o uso de aprendizado por reforço. Seu principal objetivo é mitigar os problemas enfrentados no controle usando aprendizado de máquina com a adição de técnicas de MDE para promover a integração das capacidades aprendidas por meio do aprendizado por reforço com as técnicas clássicas de controle. O *Framework* possui uma interface para criação dos modelos que diminui substancialmente a complexidade envolvida no desenvolvido do software. Desta forma o desenvolvedor pode se abster de algumas necessidades de programação dos aspectos relacionados ao aprendizado por reforço e robótica, que ficam a cargo da ferramenta.

Além disso, a autoadaptação de robôs móveis foi investigada usando métricas de desempenho e grau de complexidade usando uma rede neural em um ambiente simulado e real. Os resultados mostraram que existe uma capacidade de autoadaptação com o uso da rede neural. Os resultados experimentais mostram que o *Framework* é promissor no sentido de que a média da taxa de sucesso da missão em cenários onde o robô não foi treinado foi de 69%. Ao complementar o robô com outros controladores convencionais, foi possível tornar sua execução em ambientes reais mais segura e eficaz.

6.2 Contribuições da Pesquisa

As principais contribuições desta pesquisa de doutorado são:

- Identificação, análise e classificação das principais abordagens de MDE aplicadas a área de robótica por meio da RSL realizada nesta pesquisa. Resultado que gerou

a publicação de um *Survey* (SILVA et al., 2021).

- Criação do metamodelo de robótica com suporte a algoritmos de aprendizado por reforço e independente de plataforma.
- Criação do metamodelo para subsidiar a transformação do modelo robótico para o modelo de arquitetura ROS.
- Criação da ferramenta gráfica que apoia a adoção do *framework* RLoRDE reduzindo a complexidade envolvida no desenvolvimento do software robótico e agregando algoritmos de aprendizado por reforço contribuindo para o aumento da capacidade de utilização dos robôs em ambientes desconhecidos.

6.3 Limitações

A ideia do *Framework* RLoRDE é ser uma ferramenta o mais completa possível. No entanto, o *Framework* possui algumas limitações.

A primeira limitação está relacionada à diversidade de ambientes onde os robôs podem ser executados. Portanto, o *Framework* RLoRDE provê, a princípio, robôs em um ambiente 3D inicialmente vazio, sendo necessária a criação do mundo 3d. A adição do ambiente tridimensional é feita separadamente do *Framework* e, portanto, é adicionada somente após a geração dos códigos e antes do treinamento.

A segunda limitação é que o aprendizado por reforço requer a criação de funções de recompensa que representam algum ganho real. Como esse ganho depende de diversos fatores, como ambiente, sensores, missão, etc., construir uma função de recompensa que atenda a diversos requisitos torna-se muito difícil. A forma de mitigar esse problema foi adotar a locomoção do robô como função de recompensa. No entanto, o usuário pode adicionar sua própria função de recompensa com base nos requisitos do problema.

6.4 Trabalho Futuro

Para pesquisas futuras, pretendemos estender a plataforma adicionando suporte para treinamento de braços robóticos. Desta forma poderemos criar aplicações, de forma simples e prática, integrando robôs móveis e braços robóticos. O metamodelo proposto também pode servir de base para novos geradores de código sendo utilizados em outros middlewares e robôs.

O comportamento deliberativo também pode ser melhor explorado dentro da ferramenta proposta, dando ao robô a capacidade de planejar suas ações com base nos controladores disponibilizados após o treinamento.

O experimento realizado na pesquisa buscou validar a utilidade da ferramenta na geração de código para treinamento de controladores baseados em aprendizado por reforço e em qual seria o comportamento da rede neural diante de mudanças no cenário. No entanto, é preciso realizar mais experimentos utilizando a ferramenta em outros casos de uso em cenários com caraterísticas diferentes e com robôs diferentes.

Referências

ACCELEO. *Acceleo | Overview*. 2022. Disponível em: <https://www.eclipse.org/acceleo/overview.html>.

ADAM, K. et al. Model-driven separation of concerns for service robotics. In: *Proceedings of the International Workshop on Domain-Specific Modeling*. [S.l.: s.n.], 2016. p. 22–27.

AGRELA, L. *Amazon já faz entregas usando seu próprio robô*. 2019. Disponível em: <https://exame.abril.com.br/tecnologia/amazon-ja-faz-entregas-usando-seu-proprio-robo/>.

ALAMI, R.; CHATILA, R. Reactive motion planning for robots. *IEEE Transactions on Robotics and Automation*, p. 213–222, 1994.

ALDRICH, J. et al. Model-based adaptation for robotics software. *IEEE Software*, IEEE, v. 36, n. 2, p. 83–90, 2019.

ALI, A.; MAHMOUD, S. *Technologies for autonomous navigation in unstructured outdoor environments*. Tese (Doutorado) — University of Cincinnati, 2003.

ARKIN, R. C.; ARKIN, R. C. et al. *Behavior-based robotics*. [S.l.]: MIT press, 1998.

ARNE, N. et al. A survey on domain-specific modeling and languages in robotics. *Journal of Software Engineering in Robotics*, Università degli studi di Bergamo, v. 7, n. 1, p. 75–99, 2016.

ARREGUIN, J. M. R. *Automation and robotics*. [S.l.]: Citeseer, 2008.

BAUMGARTL, J. et al. Towards easy robot programming-using dsls, code generators and software product lines. p. 548–554, 2013.

BEKEY, G. A. *Autonomous robots: from biological inspiration to implementation and control*. [S.l.]: MIT press, 2005.

BELLMAN, R.; KALABA, R. E. *Dynamic programming and modern control theory*. [S.l.]: Citeseer, 1965. v. 81.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modeling Language User Guide (Addison-Wesley Object Technology Series)*. [S.l.]: Addison-Wesley Professional, 1999.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–182, 2012.

BROCKMAN, G. et al. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

BRUGALI, D.; GHERARDI, L. Hyperflex: A model driven toolchain for designing and configuring software control systems for autonomous robots. In: *Robot Operating System (ROS)*. [S.l.]: Springer, 2016. p. 509–534.

BUCHMANN, T. et al. Robots and their variability–a societal challenge and a potential solution. p. 27–30, 2015.

BUCKLAND, K. *Para Toyota, robôs que ajudam em tarefas domésticas estão próximos*. 2019. Disponível em: <https://exame.abril.com.br/tecnologia/para-toyota-robos-que-ajudam-em-tarefas-domesticas-estao-proximos/>.

CAZANGI, R. R.; FIGUEIREDO, M. Sistema autônomo inteligente baseado em computação evolutiva aplicado à navegação de robôs móveis. *Departamento de Informática, Universidade Estadual de Maringá–UEM, proceedings do V SBAI, Canela*, 2001.

CICCOZZI, F. et al. Adopting mde for specifying and executing civilian missions of mobile multi-robot systems. *IEEE Access*, v. 4, p. 6451–6466, 2016.

DEELSTRA, S. et al. Model driven architecture as approach to manage variability in software product families. In: *Proc. of Model Driven Architecture: Foundations and Applications*. [S.l.: s.n.], 2003. p. 109–114.

DÍAZ, V. G. et al. A brief introduction to model-driven engineering. *Tecnura*, Universidad Distrital Francisco José de Caldas, v. 18, n. 40, p. 127–142, 2014.

DJUKIĆ, V.; POPOVIĆ, A.; TOLVANEN, J.-P. Domain-specific modeling for robotics: from language construction to ready-made controllers and end-user applications. In: ACM. *Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering*. [S.l.], 2016. p. 47–54.

DOORAKI, A. R.; LEE, D.-J. An innovative bio-inspired flight controller for quad-rotor drones: Quad-rotor drone learning to fly using reinforcement learning. *Robotics and Autonomous Systems*, Elsevier, v. 135, p. 103671, 2021.

DRAGULE, S.; MEYERS, B.; PELLICCIONE, P. A generated property specification language for resilient multirobot missions. In: SPRINGER. *International Workshop on Software Engineering for Resilient Systems*. [S.l.], 2017. p. 45–61.

DU, A. M. L. *Robôs substituirão os garçons em restaurante de Pequim*. 2019. Disponível em: <https://exame.abril.com.br/tecnologia/robos-substituirao-os-garcons-em-restaurante-de-pequim/>.

DUDEK, G.; JENKIN, M. *Computational principles of mobile robotics*. [S.l.]: Cambridge university press, 2010.

EDWARDS, G. et al. Architecture-driven self-adaptation and self-management in robotics systems. In: IEEE. *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. [S.l.], 2009. p. 142–151.

ESTÉVEZ, E. et al. A novel model-based approach to support development cycle of robotic arm applications. *IFAC Proceedings Volumes*, Elsevier, v. 47, n. 3, p. 3465–3470, 2014.

ESTEVEZ, E. et al. An uml based approach for designing and coding automatically robotic arm platforms. *REVISTA IBEROAMERICANA DE AUTOMATICA E INFORMATICA INDUSTRIAL*, COMITE ESPANOL AUTOMATICA CEA C VERA 14, APDO 22012, VALENCIA, E-46071, SPAIN, v. 14, n. 1, p. 82–93, 2017.

FIGAT, M.; ZIELIŃSKI, C. Parameterised robotic system meta-model expressed by hierarchical petri nets. *Robotics and Autonomous Systems*, Elsevier, p. 103987, 2022.

FONG, T.; NOURBAKHSH, I.; DAUTENHAHN, K. A survey of socially interactive robots. *Robotics and autonomous systems*, Elsevier, v. 42, n. 3-4, p. 143–166, 2003.

GARCIA, S. et al. An architecture for decentralized, collaborative, and autonomous robots. In: IEEE. *2018 IEEE International Conference on Software Architecture (ICSA)*. [S.l.], 2018. p. 75–7509.

GAZEBO. 2022. Disponível em: <https://gazebosim.org/>.

GEORGAS, J. C.; TAYLOR, R. N. Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In: *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*. [S.l.: s.n.], 2008. p. 105–112.

GERARD, S.; BABAU, J.-P.; CHAMPEAU, J. *Model driven engineering for distributed real-time embedded systems*. [S.l.]: Wiley-IEEE Press, 2010.

GOLNAZARIAN, W.; HALL, E. L. Intelligent industrial robots. *Center for Robotics Research*, v. 1050, p. 72, 2000.

GRONDMAN, I. et al. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, IEEE, v. 42, n. 6, p. 1291–1307, 2012.

GULLAPALLI, V. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural networks*, Elsevier, v. 3, n. 6, p. 671–692, 1990.

GUSMãO, G. *Cachorros robôs são nova aposta para agilizar entregas*. 2019. Disponível em: <https://exame.abril.com.br/tecnologia/empresa-quer-combinar-carros-e-cachorros-robos-para-agilizar-entregas/>.

HAUSKNECHT, M.; STONE, P. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

HEARTY, J. *Advanced Machine Learning with Python*. [S.l.]: Packt Publishing Ltd, 2016.

HOCHGESCHWENDER, N. et al. Graph-based software knowledge: Storage and semantic querying of domain models for run-time adaptation. p. 83–90, 2016.

HRABIA, C.-E.; LEHMANN, P. M.; ALBAYRAK, S. Increasing self-adaptation in a hybrid decision-making and planning system with reinforcement learning. In: IEEE. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. [S.l.], 2019. v. 1, p. 469–478.

JAMSHIDI, P. et al. Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. In: IEEE. *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. [S.l.], 2019. p. 39–50.

JEONG, R. et al. Self-supervised sim-to-real adaptation for visual robotic manipulation. In: IEEE. *2020 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2020. p. 2718–2724.

JESUS, J. C. et al. Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In: *2019 19th International Conference on Advanced Robotics (ICAR)*. [S.l.: s.n.], 2019. p. 362–367.

JOUAULT, F. et al. Atl: a qvt-like transformation language. In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. [S.l.: s.n.], 2006. p. 719–720.

KALASHNIKOV, D. et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

KANG, K. C. et al. *Feature-oriented domain analysis (FODA) feasibility study*. [S.l.], 1990.

KELLY, S.; TOLVANEN, J.-P. *Domain-specific modeling: enabling full code generation*. [S.l.]: John Wiley & Sons, 2008.

KITCHENHAM, B. A.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Disponível em: <https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf>.

LAMPE, A.; CHATILA, R. Performance measure for the evaluation of mobile robot autonomy. In: IEEE. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* [S.l.], 2006. p. 4057–4062.

LEWIS, F. L. Neural network control of robot manipulators. *IEEE Expert*, IEEE, v. 11, n. 3, p. 64–75, 1996.

LI, H.; ZHANG, Q.; ZHAO, D. Deep reinforcement learning-based automatic exploration for navigation in unknown environment. *IEEE Transactions on Neural Networks and Learning Systems*, v. 31, n. 6, p. 2064–2076, 2020.

LI, M. et al. *Robot Intelligence Technology and Applications 2*. [S.l.]: Springer Basel, Switzerland:, 2014.

LIKMETA, A. et al. Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving. *Robotics and Autonomous Systems*, Elsevier, v. 131, p. 103568, 2020.

LILLICRAP, T. P. et al. *Continuous control with deep reinforcement learning*. arXiv, 2015. Disponível em: <https://arxiv.org/abs/1509.02971>.

LUH, G.-C.; LIU, W.-W.; LIN, H.-K. Reactive navigation of a mobile robot in unknown environments. *IFAC Proceedings Volumes*, Elsevier, v. 41, n. 2, p. 6804–6809, 2008.

MARTINEZ-TENOR, A. et al. Towards a common implementation of reinforcement learning for multiple robotic tasks. *Expert Systems with Applications*, Elsevier, v. 100, p. 246–259, 2018.

MEDEIROS, A. A. A survey of control architectures for autonomous mobile robots. *Journal of the Brazilian Computer Society*, SciELO Brasil, v. 4, p. 35–43, 1998.

MELO, F. S. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, p. 1–4, 2001.

MENS, T.; GORP, P. V. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 152, p. 125–142, 2006.

MNIH, V. et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

MOHAMMED, M.; KHAN, M. B.; BASHIER, E. B. M. *Machine learning: algorithms and applications*. [S.l.]: Crc Press, 2016.

MÜHLBACHER, C. et al. Improving dependability of industrial transport robots using model-based techniques. In: IEEE. *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. [S.l.], 2016. p. 3133–3140.

MUSIL, A. et al. Patterns for self-adaptation in cyber-physical systems. In: *Multi-disciplinary engineering for cyber-physical production systems*. [S.l.]: Springer, 2017. p. 331–368.

NAGABANDI, A. et al. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

NANDY, A.; BISWAS, M. *Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python*. [S.l.]: Apress, 2017.

NORDMANN, A.; WREDE, S.; STEIL, J. Modeling of movement control architectures based on motion primitives using domain-specific languages. p. 5032–5039, 2015.

OMG, M. Guide version 1.0.1. *Object Management Group*, v. 62, p. 34, 2003.

OMG, Q. *Meta Object Facility (MOF) 2.0 Query*. 2011. Disponível em: <http://www.omg.org/spec/MOF/2.4.1>.

OXFORD, D. *Oxford dictionaries*. 2019. Disponível em: <https://en.oxforddictionaries.com/definition/robot">.

PARASCHOS, A.; SPANOUDAKIS, N. I.; LAGOUDAKIS, M. G. Model-driven behavior specification for robotic teams. In: INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. [S.l.], 2012. p. 171–178.

QIU, C. et al. Deep deterministic policy gradient (ddpg)-based energy harvesting wireless communications. *IEEE Internet of Things Journal*, v. 6, n. 5, p. 8577–8588, 2019.

RAMASWAMY, A.; MONSUEZ, B.; TAPUS, A. Model-driven software development approaches in robotics research. In: ACM. *Proceedings of the 6th International Workshop on Modeling in Software Engineering*. [S.l.], 2014. p. 43–48.

RASCHKA, S. *Python machine learning*. [S.l.]: Packt Publishing Ltd, 2015.

RASCHKA, S.; MIRJALILI, V. *Python machine learning*. [S.l.]: Packt Publishing Ltd, 2017.

RIASCOS, L. A. Fundamentos de robótica. *São Paulo: Plêiade*, 2010.

RINGERT, J. O.; RUMPE, B.; WORTMANN, A. Tailoring the montiarcautomaton component & connector adl for generative development. p. 41–47, 2015.

ROBOTICS, S. Robotics 2020 multi-annual roadmap for robotics in europe. *SPARC Robotics, EU-Robotics AISBL, The Hauge, The Netherlands, accessed Feb*, v. 5, p. 2018, 2016.

ROMERO, R. A. et al. Robótica móvel. *Sao Paulo: LTC*, 2014.

RUMMERY, G. A.; NIRANJAN, M. *On-line Q-learning using connectionist systems*. [S.l.]: University of Cambridge, Department of Engineering Cambridge, UK, 1994. v. 37.

SANZ, P. Robotics: Modeling, planning, and control (siciliano, b. et al; 2009)[on the shelf]. *IEEE Robotics & Automation Magazine*, IEEE, v. 16, n. 4, p. 101–101, 2009.

SCHLEGEL, C.; STECK, A.; LOTZ, A. Robotic software systems: From code-driven to model-driven software development. In: *Robotic Systems-Applications, Control and Programming*. [S.l.]: IntechOpen, 2012.

SHALAL, N. et al. A review of autonomous navigation systems in agricultural environments. University of Southern Queensland, 2013.

SHANTIA, A. et al. Two-stage visual navigation by deep neural networks and multi-goal reinforcement learning. *Robotics and Autonomous Systems*, Elsevier, v. 138, p. 103731, 2021.

SILVA, A. R. D. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, Elsevier, v. 43, p. 139–155, 2015.

SILVA, E. *Reinforcement Learning for Robotic model-Driven Engineering*. [S.l.]: GitHub, 2022. <https://github.com/EdsonASilva/RLoRDE>.

SILVA, E. de A. et al. A survey of model driven engineering in robotics. *Journal of Computer Languages*, Elsevier, p. 101021, 2021.

SOMMERVILLE, I.; ARAKAKI, R.; MELNIKOFF, S. S. S. *Engenharia de software*. [S.l.]: Pearson Prentice Hall, 2008.

SPÍNOLA, R. O.; DIAS-NETO, A. C.; TRAVASSOS, G. H. Abordagem para desenvolver tecnologia de software com apoio de estudos secundários e primários. In: SN. *Experimental Software Engineering Latin American Workshop (ESELAW)*. [S.l.], 2008. p. 25.

STAHL, T.; VOELTER, M.; CZARNECKI, K. Model-driven software development: Technology, engineering, management. John Wiley & Sons, 2006.

STECK, A.; LOTZ, A.; SCHLEGEL, C. Model-driven engineering and run-time model-usage in service robotics. v. 47, n. 3, p. 73–82, 2011.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

SYKES, D. et al. From goals to components: a combined approach to self-management. In: *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*. [S.l.: s.n.], 2008. p. 1–8.

TensorFlow. 2022. Disponível em: <https://www.tensorflow.org/?hl=pt-br>.

UML, O.; MOF, I. *The Unified Modeling Language UML*. 2010. Disponível em: <http://www.omg.org/spec/UML/2.3/Infrastructure/PDF>.

VASILEV, I. et al. *Python Deep Learning*. [S.l.]: Packt Publishing Ltd, 2019.

WATKINS, C. J. C. H. Learning from delayed rewards. King's College, Cambridge, 1989.

WEYNS, D.; MALEK, S.; ANDERSSON, J. Forms: a formal reference model for self-adaptation. In: *Proceedings of the 7th international conference on Autonomic computing*. [S.l.: s.n.], 2010. p. 205–214.

WIGAND, D. L. et al. Modularization of domain-specific languages for extensible component-based robotic systems. p. 164–171, April 2017.

WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, Springer, v. 8, n. 3, p. 229–256, 1992.

WITTEN, I. H. An adaptive optimal controller for discrete-time markov environments. *Information and control*, Elsevier, v. 34, n. 4, p. 286–295, 1977.

YAN, C.; XIANG, X.; WANG, C. Fixed-wing uavs flocking in continuous spaces: A deep reinforcement learning approach. *Robotics and Autonomous Systems*, Elsevier, v. 131, p. 103594, 2020.

YANG, C.-H. H. et al. Enhanced adversarial strategically-timed attacks against deep reinforcement learning. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2020. p. 3407–3411.

ZOU, A.-M. et al. Neural networks for mobile robot navigation: a survey. In: SPRINGER. *International Symposium on Neural Networks*. [S.l.], 2006. p. 1218–1226.