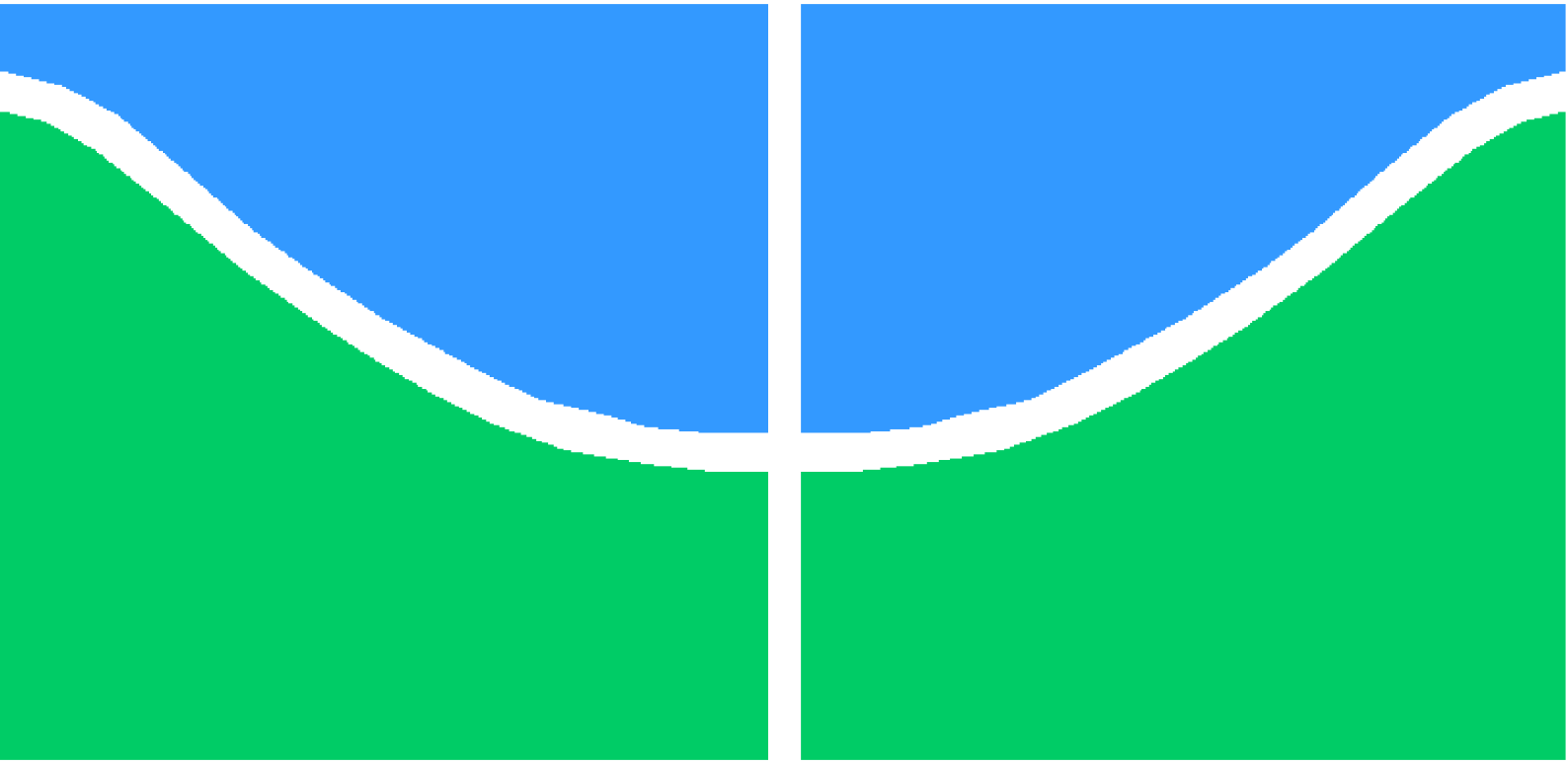

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Máquina de Raciocínio Lógico para Tomada de Decisões Estratégicas em Robótica Educacional

Autor: Carolina Barros Ramalho
Orientador: Profª. Drª. Milene Serrano

Brasília, DF
2016

Carolina Barros Ramalho

Máquina de Raciocínio Lógico para Tomada de Decisões Estratégicas em Robótica Educacional

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Profª. Drª. Milene Serrano

Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF

2016

Carolina Barros Ramalho
Máquina de Raciocínio Lógico para Tomada de Decisões Estratégicas em Robótica Educacional/ Carolina Barros Ramalho. – Brasília, DF, 2016-
95 p. : il. (algumas color.) ; 30 cm.

Orientador: Profª. Drª. Milene Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. robótica educacional. 2. programação dinâmica. I. Profª. Drª. Milene Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Máquina de Raciocínio Lógico para Tomada de Decisões Estratégicas em Robótica Educacional

CDU 02:141:005.6

Carolina Barros Ramalho

Máquina de Raciocínio Lógico para Tomada de Decisões Estratégicas em Robótica Educacional

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 05 de julho de 2016:

Profª. Drª. Milene Serrano
Orientador

Prof. Dr. Maurício Serrano
Coorientador

Prof. Ms. Giovanni Almeida Santos
Convidado 1

Brasília, DF
2016

Este trabalho é dedicado à minha mãe, Maria das Graças Barros Ramalho, ao meu pai, Francisco de Assis Ramalho e à minha irmã, Maysa Ramalho Lima, que fizeram de mim tudo o que sou hoje. Dedico também ao meu namorado, Wisley Rocha da Silva, por estar ao meu lado e me apoiar durante essa longa e árdua jornada.

Agradecimentos

Agradeço primeiramente à Deus, por ter me guiado até aqui e ter me dado força para continuar, mesmo quando eu achava que já não as tinha.

Agradeço à minha mãe, Maria das Graças Barros Ramalho, pelo amor, carinho, compreensão e por todo sacrifício que fez para que hoje eu estivesse aqui.

Agradeço ao meu pai, Francisco de Assis Ramalho, por ter me ensinado todos os valores da vida e por me amar incondicionalmente.

Agradeço à minha irmã, Maysa Ramalho Lima, por me amar como uma filha, por sempre me apoiar e me dar condições de chegar onde cheguei, acreditando em mim a todo momento.

Agradeço ao meu namorado, Wisley Rocha da Silva, por estar ao meu lado, por multiplicar minhas alegrias, dividir minhas tristezas e sonhar meus sonhos comigo.

Agradeço à Profª. Milene Serrano pela atenção e carinho, pelas orientações ao longo desse trabalho, pelo conhecimento que comigo compartilhou. Você é uma referência de pessoa e profissional que quero ser.

Agradeço ao Prof. Maurício Serrano pela coorientação nesse trabalho, pelas horas que disponibilizou para me ajudar, pelos "puxões de orelha"que me deu ao longo do curso. Tenho certeza que serei uma engenheira de software melhor agora.

Agradeço ao Prof. Paulo Meirelles e ao Prof. Hilmer Neri por me darem a oportunidade de abrir novos horizontes na minha formação acadêmica.

Agradeço ao Prof. Ricardo Chaim pelos projetos desenvolvidos ao longo do curso.

Agradeço ao amigo Rodrigo Rincon por toda disponibilidade e colaboração para a conclusão deste trabalho.

Resumo

A robótica está cada vez mais inserida no âmbito educacional, desde a escola até a universidade, proporcionando aos alunos uma experiência única de aprendizagem. Com o lançamento do kit LEGO Mindstorms, a robótica ficou cada vez mais interessante aos olhos dos jovens, pois montar um robô com sensores, motores e programá-lo ficou mais acessível. Ao se popularizar, o kit LEGO Mindstorms tornou-se um dos principais integrantes dos torneios de robótica, até chegar ao ponto de existirem torneios somente para ele, como o Torneio de Robótica FIRST LEGO League. Nesses torneios, a maior preocupação é conseguir ganhar a maior quantidade de pontos possíveis em um espaço de tempo relativamente curto, sendo necessária a elaboração de estratégias para otimizar o gasto do tempo. O objetivo desse trabalho é a criação de uma máquina de raciocínio para otimizar o ganho de pontos, durante a quantidade de tempo estabelecida, priorizando, dentre as missões não realizadas, quais devem ser executadas pelo robô. Este trabalho será *open-source*, desenvolvido na linguagem de programação Prolog, e testado no robô LEGO Mindstorms NXT, por meio de uma conexão *bluetooth*.

Palavras-chaves: robótica educacional. máquina de raciocínio. prolog. programação dinâmica. NXT. LEGO.

Abstract

The robotics is more and more inserted in the education extent, since the school until the university, providing to the pupils a single learning experience. With the launch of the kit LEGO Mindstorms, the robotics is more and more interesting to the young people's eyes, because to mount a robot with sensors, motors and to plan it is more accessible. With its popularizing, the kit LEGO Mindstorms became one of the main integrants of the tournaments of robotics, until reaching the point of create tournaments only for it, like the Tournament of Robotics FIRST LEGO League. In these tournaments, the biggest preoccupation is to win the biggest quantity of possible points in short time, and it's necessary the preparation of strategies to optimize the waste of the time. Through the dynamic programming, the objective of this work is the creation of a machine of reasoning to optimize to the profit of points during the established quantity of time, prioritizing, among the missions that weren't realized, those that must be executed by a robot. This work will be open-source, developed in the language of programming Prolog, and tested in the robot LEGO Mindstorms NXT, through a connection bluetooth.

Key-words: educational robotics. reasoning engine. prolog. dynamic programming. NXT. LEGO.

Lista de ilustrações

Lista de abreviaturas e siglas

API	Application Program Interface
ARM	Advanced RISC Machine
BPMN	Business Process Model and Notation
GPL	General Public License
HDMI	High Definition Multimedia Interface
IDE	Integrated Development Environment
KB	KiloByte
MB	MegaByte
MHz	MegaHertz
MIT	Massachusetts Institute of Technology
PD	Programação dinâmica
RAM	Random Access Memory
RCA	Radio Corporation of America
RIS	Robotics Invention System
RMA	Robôs Móveis Autonomos
SLD	Selection-rule driven Linear resolution for Definite clauses
SMPA	Sense-Model-Plan-Act
USB	Universal Serial Bus

Lista de símbolos

$\subseteq$	Contido em ou igual a
$<$	Menor
$\in$	Pertence
$\cup$	União
$\emptyset$	Vazio

Sumário

1 Introdução

A robótica costuma ser uma área bem atrativa para qualquer pessoa, seja ela conhecedora ou não do assunto. Devido a esse interesse, o ramo da robótica vem se tornando particularmente popular nos últimos anos. A robótica móvel, em especial, vem agaranhando fãs ao redor do mundo, que buscam a possibilidade de construir seus próprios robôs e programá-los para uma função específica. Os kits de robótica ganharam espaço no mercado devido à sua completude, pois contém motores e sensores diversos e também devido ao seu baixo custo, se comparado com robôs mais avançados. Com a popularização do uso dos kits de robótica, vários torneios de robótica foram iniciados, atraindo jovens estudantes, universitários e profissionais da área.

Esse capítulo está organizado em seis sessões, sendo essas: Contextualização, Justificativa, Questão de Pesquisa, Objetivos, Metodologia e Organização dos Capítulos, todos devidamente detalhados a seguir.

1.1 Contextualização

O termo robô surgiu em meados do século XX, derivado da palavra tcheca *robota*, que significa trabalhador forçado (ou escravo) (SILVA, 2009). Desde esta época, a propagação do conceito de robôs está acontecendo de forma acelerada, seja por filmes de ficção científica, ou documentários, ou desenhos animados. Os robôs saíram, de fato, da ficção científica em 1961, quando Joseph Engelberger desenvolveu o primeiro robô comercial, o UNIMATE e, desde então, estão cada dia mais inseridos em meio a sociedade, seja como elevadores, caixas eletrônicos, robôs de entretenimento ou de chão de fábrica (MURPHY, 2000 apud SILVA, 2009). Segundo (SILVA, 2009), um robô deve ter, idealmente, os seguintes elementos:

- atuadores: são os meios utilizados para que o robô se locomova e/ou altere a forma de seu corpo. Exemplo: pernas, rodas, articulações, garras, dentre outros;
- sensores: são os meios utilizados pelo robô para medir e conhecer o ambiente, detectando objetos, calor ou luz, e convertendo essa informação em símbolos processados por computadores;
- computador: é o responsável por controlar o robô através de algoritmos nele implementados;
- equipamentos ou mecanismos: são ferramentas ou equipamentos mecânicos.

Um robô pode ser categorizado em um dos três grupos, atualmente conhecidos: manipuladores, móveis e híbridos. Os robôs manipuladores são fixos ao seu local de trabalho; os móveis se locomevem por meio dos atuadores, e os híbridos são um composto das duas categorias anteriores (RUSSEL; NORVIG, 2004 apud SILVA, 2009).

A robótica é uma ciência, em rápida ascenção, que envolve áreas do conhecimento como: microeletrônica, computação, engenharia mecânica, inteligência artificial, física, neurociência, entre outras. Portanto, estuda tecnologias associadas ao projeto, fabricação, teoria e aplicação dos robôs.

Sendo a robótica uma área tão inter e multidisciplinar, usá-la como instrumento de aprendizagem é um tanto quanto benéfico, pois ensina à criança e/ou ao jovem a trabalhar em equipe, desenvolver o raciocínio lógico através de problemas concretos, e estimula a leitura, exploração, investigação, criatividade e organização. Além de aprimorar a parte motora do indivíduo ao trabalhar com o hardware do robô, também é aprimorado o raciocínio lógico e a abstração ao programar o software do robô (SILVA, 2009).

Em meados dos anos 60, Seymourt Papert, reconhecido matemático, educador e pesquisador do MIT[1] (Instituto de Tecnologia de Massachusetts), criou a linguagem de programação LOGO. Essa linguagem foi utilizada nos kits educacionais da LEGO, conferindo o início do sistema educacional LEGO-LOGO. Nesse sistema, as crianças têm a possibilidade de construir seus robôs protótipos com os blocos de montagem e outros recursos do kit educacional da LEGO bem como programar com a linguagem LOGO, gerando o comportamento desejado nesses protótipos (SILVA, 2009).

Um dos kits de robótica mais populares criados pela LEGO é o Mindstorms (SILVA, 2009), que combina um computador programável, RCX, NXT ou EV3, dependendo da versão, com motores elétricos, engrenagens, peças de encaixe, polias, roscas, dentre outros. Este kit contêm cerca de mil peças LEGO, incluindo o computador, o CD-ROM do software Mindstorms, um transmissor infravermelho para envio de programas para o robô (apenas para a versão RCX), um guia do construtor, motores, sensores, rodas, pneus, conectores e outros. Para aprimorar o aprendizado, a LEGO disponibiliza, além do kit de peças para montagem dos robôs, um tapete de missões a serem realizadas pelos mesmos. Cada missão do tapete tem uma pontuação máxima, que poderá ser alcançada se a missão for completada dentro do tempo e sem penalidades. É possível completar 'n' missões dentro do tempo limite.

1.2 Justificativa

Retomando aspectos apontados na contextualização, seguem algumas preocupações intrínsecas desse contexto, com as quais se procura justificar as necessidades desta

[1] http://web.mit.edu

pesquisa: (i) de uma investigação mais detalhada quanto à literatura associada; (ii) da elicitação de soluções candidatas, apoiadas na Engenharia de Software, na Inteligência Artificial e no projeto e análise de algoritmos, e (iii) da implementação de uma solução dentre as elicitadas.

O curso de Engenharia de Software da UnB/FGA oferece uma disciplina chamada Princípios de Robótica Educacional. Dentre os objetivos dessa disciplina, tem-se a intenção de formar grupos de granduandos para competições em robótica nos âmbitos regional, nacional e internacional. Nesse contexto, são estabelecidos desafios aos grupos de alunos matriculados na disciplina. Esses desafios orientam-se pelas propostas de atividades da First LEGO League (KAMEN; KRISTIANSEN, 2010). Uma preocupação intrínseca dos grupos participantes consiste em lidar com diferentes variáveis, em tempo de competição. No caso, destacam-se: tempo total estabelecido para conclusão das missões, localização atual do robô e pontuação das missões.

No intuito de colaborar na formação de uma equipe competitiva, têm-se como preocupações a serem investigadas nesse Trabalho de Conclusão de Curso, principalmente:

- O fato do tempo total para a conclusão das missões ser relativamente curto para que todas sejam realizadas, logo são necessários algoritmos que permitam a seleção dessas missões considerando o tempo como um fator impactante.
- Seleção das missões de forma apropriada, na qual outras variáveis, além do tempo, devem ser consideradas, tais como:
  - Localização atual do robô, pois dependendo da posição do robô em relação ao tapete o tempo de locomoção para realizar uma missão irá mudar.
  - Pontuação das missões, pois pode existir uma missão 'X' que gaste o mesmo tempo para ser realizada que uma missão 'Y', porém 'X' tem uma maior pontuação que 'Y', logo será mais valoroso executar a missão 'X' do que a missão 'Y'.

Diante do exposto, acredita-se que a elaboração de uma base de conhecimento orientada ao Paradigma Lógico (TUCKER; NOONAN, 2009), especificamente implementada com base em algoritmos da Inteligência Artificial, conferirá ao robô a capacidade de gerar um roteiro de missões, sendo esse o de maior pontuação possível de ser realizada dentro do tempo restante.

Pretende-se, com tal esforço, agregar valor na formação de uma equipe competitiva em robótica, disponibilizando uma máquina de raciocínio apoiada em algoritmos avançados e instigando os membros da equipe a refinarem essa máquina de forma contínua e evolutiva, ou seja, ir aprimorando-a de acordo com as necessidades.

1.3 Questão de pesquisa

Este TCC buscará responder ao seguinte questionamento: É possível implementar inteligência artificial no robô LEGO Mindstorms, isto é, uma máquina de raciocínio lógico, de tal forma que dada uma posição (X0, Y0) e um tempo restante, esse robô consiga gerar um roteiro de missões a serem executadas para alcançar a maior pontuação possível?

1.4 Objetivos

Os objetivos deste trabalho foram classificados quanto a objetivo geral e objetivos específicos, sendo ambos apresentados a seguir.

1.4.1 Objetivo geral

Desenvolver uma máquina de raciocínio lógico, considerando um algoritmo específico para tomada de decisões, com a qual o robô deverá ser capaz de selecionar as missões, compondo um roteiro, de forma que a pontuação seja maximizada em relação ao tempo disponível, previamente estabelecido.

1.4.2 Objetivos específicos

Com base no objetivo geral, foram propostos os seguintes objetivos específicos deste TCC:

- Investigar algoritmos candidatos à solução da questão de pesquisa considerando, em um primeiro escopo dessa atividade, o estudo de algoritmos gulosos, programação dinâmica e paradigma lógico;

- Implementar uma solução que agregue ao robô a capacidade de gerar roteiros de missões de máxima pontuação a partir de um ponto inicial e um tempo restante.

- Estudar o funcionamento do kit educacional LEGO Mindstorms visando a realização de uma pesquisa exploratória e experimental que permita a verificação quanto a pertinência da solução estabelecida para o contexto investigado.

- Estabelecer uma metodologia de desenvolvimento, orientada às boas práticas da Engenharia de Software, no intuito de conduzir o processo investigativo, a implementação da solução bem como a verificação dessa solução no contexto acordado.

1.5 Metodologia de Pesquisa

Este trabalho é classificado quanto à natureza como pesquisa aplicada, pois tem por objetivo gerar conhecimento para a solução de um problema específico (MORESI et al., 2003), o desenvolvimento de uma máquina de raciocínio para tomada de decisões estratégicas em robótica educacional.

Quanto à abordagem, esta pesquisa é classificada como qualitativa, pois a fonte direta de coleta de dados é o ambiente natural estudado, campeonatos de róbotica da Fisrt Lego League, e não requer o uso de métodos e técnicas estatísticas (MORESI et al., 2003).

Quanto à tipologia, este trabalho é classificado como descritivo, pois objetiva estabelecer relações entre variáveis (MORESI et al., 2003), sendo elas: tempo restante de competição, pontuação de cada missão e tempo de execução de cada missão. Essas váriáveis estão contidas em uma base de conhecimento, e a partir dela foi criado um algoritmo, baseado em programação dinâmica, que gera um roteiro de missões que devem ser executadas pelo robô, para que a maior pontuação seja atingida.

Quanto aos procedimentos técnicos, são utilizados nesse trabalho a pesquisa bibliográfica, que tem como base livros e artigos científicos, e a experimental que será baseada em cenários de uso, cada um com o conjuto diferente de missões disponíveis para a geração do roteiro.

Quanto às técnicas de coleta de dados, neste trabalho serão utilizados a bibliografia e a observação sistemática. A Figura 1 apresenta de forma resumida essa classificação.

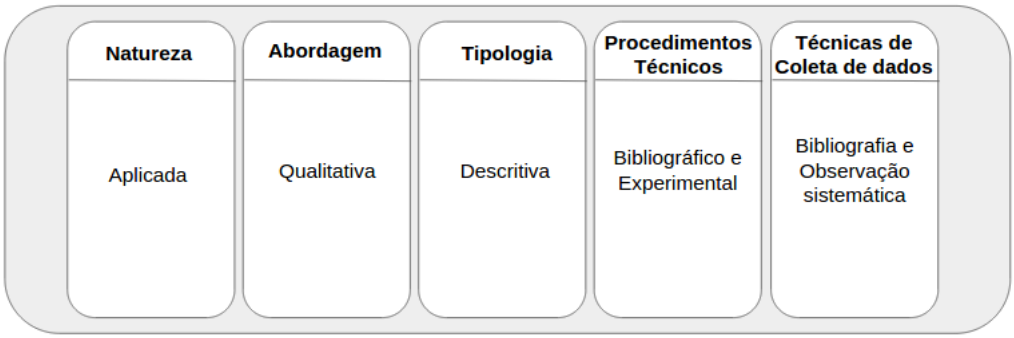

Figura 1 – Classificação da metodologia de pesquisa

Segundo André (2008), uma pesquisa é executada de acordo com três fases: Planejamento, Coleta de dados e Análise de dados.Tais fases foram adotadas por este trabalho e estão organizadas na Figura 2.

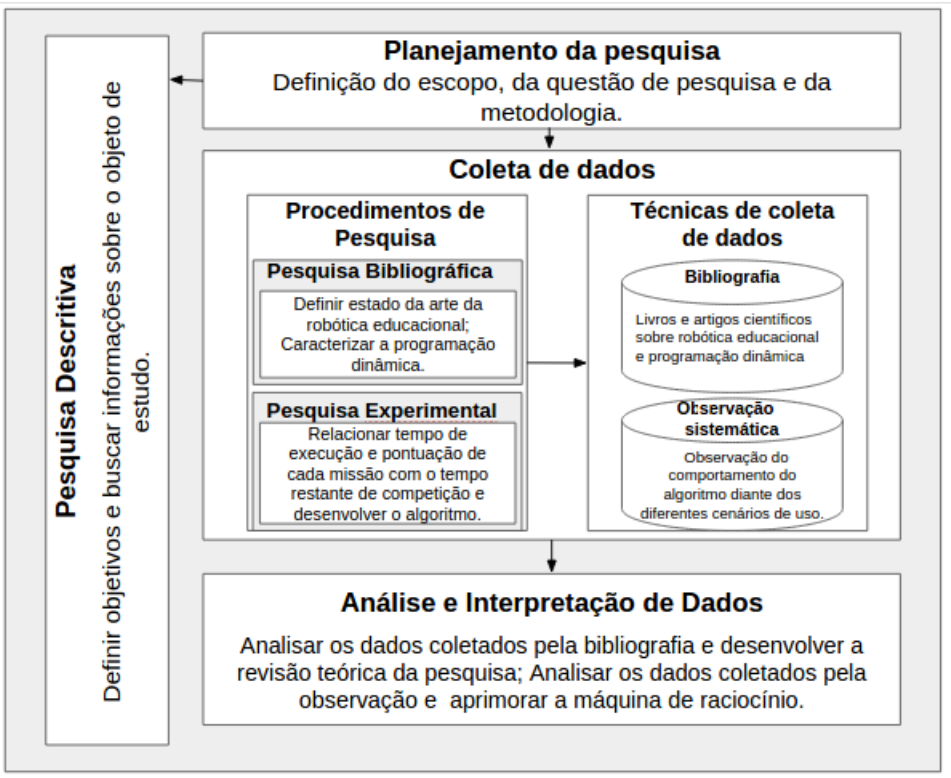

Figura 2 – Fases da pesquisa

A fase de planejamento consiste em definir o objeto de estudo, o escopo do estudo, a questão de pesquisa e a metodologia de pesquisa, com os devidos procedimentos e instrumentos de coleta. Na fase de coleta de dados, se aplicam os procedimentos de coleta. Por fim, na última fase os dados são analisados e os resultados são relatados.

Na fase de coleta de dados, são utilizadas as técnicas de revisão bibliográfica e observação sistemática, detalhadas a seguir.

- **Revisão bibliográfica**

  Pesquisa realizada nas principais bases de dados e livros, com foco na caracterização: (i) da evolução da robótica até a sua utilização no âmbito educacional; (ii) do estado da arte da robótica móvel; (iii) dos algoritmos de otimização, como algoritmo guloso e programação dinâmica, e (iv) do paradigma lógico de programação.

- **Observação sistemática**

  Observação conduzida de acordo com um planejamento para responder ao seguinte questionamento: a máquina de raciocínio está satisfatória? Para chegar a resposta desta questão, são utilizados critérios de avaliação, previamente estabelecidos.

1.6 Organização dos capítulos

Este trabalho de conclusão de curso está organizado em seis capítulos, sendo o primeiro dedicado à Introdução. Os demais são brevemente descritos a seguir:

- Referencial teórico: explana sobre a evolução da robótica educacional, principais conceitos do Paradigma Lógico e da programação dinâmica.
- Suporte tecnológico: descreve as tecnologias utilizadas no desenvolvimento da máquina de raciocínio, apresentando ferramentas, *plugins* e *kits* utilizados.
- Prolego: A máquina de raciocínio : aborda, dentre outros detalhes, a descrição dos componentes que compõe a máquina de raciocínio, bem como o detalhamento da montagem do robô, do tapete de missões, das missões, da arquitetura do robô e do Prolego.
- Cenários de teste: detalha cada cenário de teste utilizado bem como seus resultados, a fim de responder ao questionamento da observação sistemática.
- Conclusão: efetua uma reflexão sobre o trabalho de forma a concluir se todos os objetivos estabelecidos foram atingidos e a questão de pesquisa respondida, bem como relaciona trabalhos futuros.

2 Referencial teórico

Neste capítulo, serão apresentados alguns conceitos importantes para a melhor compreensão do tema abordado por este projeto. O capítulo de referencial teórico foi organizado em dois escopos, um mais voltado à bibliografia investigada no domínio da Robótica Educacional e outro mais voltado à Engenharia de Software.

2.1 Robótica educacional

O uso dos robôs pelos seres humanos é datado de antes de Cristo, mas a definição da palavra robótica é recente. A robótica como instrumento de ensino está tomando força nos últimos anos, e vem apoiando muitos educadores na tarefa de fazer o aluno se sentir mais atraído pelo conteúdo, e aprender através de observações e manipulações. A seguir é abordado brevemente a história da robótica, bem como sua definição, seu uso no âmbito educacional e os fundamentos da robótica móvel.

2.1.1 Perspectiva histórica da robótica

Os robôs são máquinas que podem substituir o homem na execução das tarefas, e consequentemente, destinados a melhorar a produção e a qualidade de vida (BARRIENTOS et al., 1997 apud SANTOS, 2002). Desde os primórdios, o homem sente-se atraído por esse mundo das máquinas, como os antigos Egípcios que adicionaram braços mecânicos às estátuas de seus deuses para serem operados por sarcedotes; e os Gregos que construíram estátuas movidas hidraulicamente; e os Chineses que entre os séculos XVIII e XIX contruíram bonecas que transportavam chá (SANTOS, 2002). Os autômatos projetados pelos Gregos tinham uma função mais estética e contemplativa, já os projetados pelos Árabes tinham funcionalidade, deixando a par preocupações estéticas e de entretenimento (SANTOS, 2002).

É perceptível que os homens, há muitos séculos, utilizam os robôs, mas sem o conhecimento da palavra, que foi divulgada a primeira vez em 1921, pelo checoslovaco Karel Capek, no seu romance *Rossum's Universal Robots*. Esse autor descreveu os robôs como máquinas com braços trabalhando duas vezes mais que os humanos, de forma incansável, eficiente e obediente, mas que se tornariam malévolos e dominariam o mundo (SANTOS, 2002). Porém em 1950, Isaac Asimov defendeu, em sua obra literária *I Robot*, que a contrução de robôs seguiria uma linha positiva e benéfica (NOF, 1999 apud SANTOS, 2002), concebendo-os como autômatos de aparência humana mas desprovidos de sentimentos (SANTOS, 2002). Asimov, em sua obra, fala sobre a inteligência dos robôs

de acordo com as seguintes leis da robótica (SANTOS, 2002):

- 1ª Lei: Um robô não pode prejudicar um ser humano, ou quando inativo, deixar um ser humano exposto ao perigo;
- 2ª Lei: Um robô deve obedecer às ordens dadas pelo ser humano, exceto se tais ordens estiverem em contradição com a 1ª Lei.
- 3ª Lei: Um robô deve proteger a sua própria existência desde que essa proteção não entre em conflito com a 1ª e a 2ª Leis.

Dessa forma, Asimov popularizou o termo robô e idealizou as três leis fundamentais da robótica (SILVA et al., 2008).

2.1.2 Definição de robô e robótica

A Divisão Internacional de Robótica da Sociedade de Engenharia de Manufatura define um robô como sendo um manipulador reprogramável, multifunções, utilizado para deslocar outros materias ou objetos específicos através da programação de movimentos (SILVA et al., 2008). Um computador é manipulado pelo homem, logo não é considerado um robô, pois um robô é um mecanismo inteligente que funciona de forma autônoma (CURCIO, 2008).

(PIO; CASTRO; JÚNIOR, 2006) definem a robótica como:

> a ligação inteligente entre a percepção e a ação. Trabalhar em Robótica significa estudar, projetar e implementar sistemas ou dispositivos que, com a utilização de percepção e de certo grau de "inteligência", sejam úteis na realização de uma determinada tarefa, pré-definida ou não, que envolva interação física entre o sistema (ou dispositivo) e o meio onde a tarefa está sendo realizada.

Porém, uma definição mais formal para o termo robótica seria "uma ciência da engenharia aplicada, tida como uma combinação da tecnologia de máquinas operatrizes e ciência da computação"(GROOVER, 1989 apud REDEL; HOUNSELL, 2004). Assim, a robótica envolve várias disciplinas como engenharia mecânica, elétrica, inteligência artificial e utiliza o robô como principal instrumento (CURCIO, 2008). Os robôs podem ser categorizados em primeira, segunda e terceira geração, sendo progressivamente mais inteligentes (BARRIENTOS et al., 1997 apud SANTOS, 2002).

- Primeira geração: A única função inteligente desses robôs é a apredizagem de uma sequência de ações de manipulação, coordenadas por um operador humano usando uma unidade de comando. Os robôs dessa geração repetem sistematicamente as tarefas e ignoram possíveis alterações no meio externo, causando restrições no seu uso,

como posicionamento no espaço, relacionamento com outras máquinas e a segurança das pessoas que ficam próximas ao robô.

- Segunda geração: esses robôs tiveram a adição de um processador à sua configuração, possibilitando a adaptabilidade ao ambiente, mesmo que de forma mínima, com utilização de sensores para auxiliar na localização, detecção de esforços e adaptação de seus movimentos às informações coletadas. Geralmente a área de atuação destes robôs está ligada à manufatura autômata.
- Terceira geração: é uma versão mais recente dos robôs, na qual é incorporado à sua configuração, processadores múltiplos em que, cada operação em sicronia desempenha uma tarefa diferente, tornando um mesmo robô multitarefas.

2.1.3 Robótica como instrumento educacional

A róbótica educacional é um meio de inserir a tecnologia no meio educativo como forma de aprendizado, oferencendo aos alunos uma oportunidade de vivenciarem experiências semelhentes às que vivem na vida real.

2.1.3.1 Teorias de aprendizagens

A teoria de aprendizado que fundamentou a róbotica educacional foi a construcionista, criada por Seymour Papert, baseada na teoria construtivista criada por Jean Piaget. Dentre as várias teorias de aprendizado existentes, algumas são brevemente descritas a seguir, segundo (ZILLI et al., 2004).

1. **Teoria das inteligências múltiplas:** Howard Gardner, professor e psicólogo da *Harvard Graduate School of Education*, dedicou-se, desde a década de 80, ao estudo das capacidades simbólicas em crianças com QI normal e alto, desenvolvendo o que ele chamou de "inteligências múltiplas". Essas podem ser vistas como contendo três princípios fundamentais:
   - A inteligência não é algo que pode ser visto como tendo múltiplas habilidades. Existem múltiplas inteligências, cada uma com habilidades distintas entre si.
   - As inteligências são interdependentes. Logo, caso seja avaliada a competência de uma pessoa em uma área, esta avaliação não servirá para outras áreas.
   - Apesar de serem interdependentes, as inteligências podem sim trabalhar em conjunto; caso contrário, nada seria feito e nenhum problema seria resolvido.

   Gardner identificou essas inteligências distintas e classificou da seguinte forma:
   - Inteligência linguística: é relacionada à habilidade da pessoa em produzir a liguagem escrita e falada.

- Inteligência lógico-matemática: é relacionada à habilidade para explorar relações, categorias e padrões, por meio da manipulação de objetos ou símbolos, e à habilidade de resolver problemas por meio do raciocínio.
- Inteligência musical: é relacionada, como o próprio nome já diz, ao reconhecimento da estrutura musical, à sensibilidade dos sons, à criação de melodias e ritmos, à percepção das qualidades dos tons e à habilidade para tocar instrumentos.
- Inteligência espacial: é relacionada à habilidade do indivíduo de visualizar um objeto e ter uma percepção acurada de diferentes ângulos, relações de espaço, dentre outros.
- Inteligência cinestésica: é representada pela capacidade de manipular objetos habilmente e controlar os movimentos do corpo.
- Inteligência interpessoal: é relacionada à capacidade de diferenciar e dar uma resposta aos estados de humor, temperamentos, desejos e motivações das outras pessoas.
- Inteligência intrapessoal: é relacionada à habilidade para ter acesso aos próprios sentimentos, aos estados interiores do ser, e saber utilizá-los na solução de problemas pessoais.

Segundo Gardner, a escola valoriza mais a inteligência linguística e a lógico-matemática, e o uso do computador pode colaborar para o amadurecimento das outras inteligências, por ser uma ferramenta adaptável às mais diversas formas de uso.

2. **Teoria do ensino por competências:** Philippe Perrenoud, professor e sociólogo suiço, é referência para diversos educadores com suas idéias pioneiras e vanguardistas sobre profissionalização de professores e avaliação de alunos. Perrenoud afirma que "a trilogia das habilidades de ler, escrever, contar, que fundou a escolaridade obrigatória no século XIX não está mais a altura das exigências da nossa época". Assim, Perrenoud propôs o ensino por competências que considera que os saberes são ferramentas para a ação e que se aprende a usá-los. O Centro de Referência Educacional elencou dez competências que devem ser trabalhadas pela escola:

   - Respeitar as identidades e as diferenças;
   - Utilizar-se das linguagens como meio de expressão, comunicação e informação;
   - Inter-relacionar pensamentos, ideias e conceitos;
   - Desenvolver o pensamento crítico e flexível e a autonomia intelectual;
   - Adquirir, avaliar e transmitir informações;
   - Compreender os princípios das tecnologias e suas relações integradoras;

- Entender e ampliar fundamentos científicos e tecnológicos;
- Desenvolver a criatividade;
- Saber conviver em grupo;
- Aprender a aprender.

3. **Teoria do construtivismo:** O suiço Jean Piaget estudou como o aprendiz passa de um estado de menor conhecimento para outro de maior conhecimento, e desenvolveu a teoria do construtivismo que foca no conhecimento científico na perspectiva do indivíduo que aprende. Nessa teoria, o sujeito é um ser ativo que estabelece relação de troca com o meio-objeto, relações essas que devem ser vivenciadas e significativa. Assim, o indivíduo incorpora novas informações, que passam a tornar parte do conhecimento. Segundo (ZILLI et al., 2004),

   > O construtivismo defende que o conhecimento não é uma cópia da realidade, mas sim uma construção do ser humano em conseqüência da sua interação com o ambiente e resultado de suas disposições internas.

   Piaget classifica os períodos de inteligência em estágios, organizados do nascimento até a fase adulta. Esses estágios indicam saltos bruscos nas capacidades do indivíduo, pois as capacidades cognitivas sofrem uma forte reestruturação.

4. **Teoria do construcionismo:** Seymour Papert, psicólogo do Laboratório de Inteligência Artificial do MIT, adaptou os princípios do construtivismo definido por Piaget e criou a teoria construcionista, que considera o computador como uma ferramenta para a construção do conhecimento e desenvolvimento do aluno. Duas ideias principais sobre a construção do conhecimento fazem com que o construcionismo se diferencie do construtivismo:

   - O aprendiz é quem constrói o conhecimento, através das coisas que faz;
   - O aprendiz constrói algo do seu interesse e que o motiva.

   Segundo Papert, a criança aprende de forma mais eficaz quando, por ela mesma, atinge o conhecimento específico de que precisa. O contrucionismo tem como meta ensinar de forma a produzir o maior conhecimento possível com o mínimo de ensino.

2.1.3.2 Conceito da robótica educacional

(MAISONNETTE, 2002) define róbotica como sendo

> o controle de mecanismos eletro-eletrônicos através de um computador, transformando-o em uma máquina capaz de interagir com o meio ambiente e executar ações decididas por um programa criado pelo programador a partir destas interações.

(MAISONNETTE, 2002) também afirma que a robótica educacional é a aplicação dessa tecnologia como mais uma ferramenta de ensino para que os alunos vivenciem de forma real o que estudam, tendo a oportunidade de propor e solucionar problemas difíceis ao invés de apenas observar a solução. Através da robótica educacional, os alunos podem explorar ideias e descobrir novos caminhos na aplicação de conceitos que aprendem em sala de aula. Dessa forma, adquirem a capacidade de produzir hipóteses, averiguar soluções, estabelecer relações e inferir conclusões (BENITTI et al., 2009). Além de benéfico ao aluno, a robótica educacional colabora com o professor no ensino de muitos conceitos teóricos de difícil compreensão, motivando o aluno a observar, abstrair e inventar. O conhecimento adquirido através de observação, abstração e do esforço do aluno tem muito mais significado para ele, e se adapta melhor às suas estruturas mentais (ZILLI et al., 2004).

2.1.3.3 Objetivos da robótica educacional

Segundo (ZILLI et al., 2004), além de propiciar o conhecimento da tecnologia atual, a robótica educacional pode desenvolver as seguintes competências:

- raciocínio lógico;
- habilidades manuais e estéticas;
- relações inter e intrapessoais;
- utilização dos conceitos aprendidos em diversas áreas do conhecimento para o desenvolvimento de projetos;
- investigação e compreensão;
- representação e comunicação;
- trabalho com pesquisa;
- resolução de problemas por meio de tentativa e erro;
- aplicação das teorias às atividades concretas;
- utilização da criatividade em diversas situações, e
- capacidade crítica.

(ZILLI et al., 2004) também apresentam uma classificação dos objetivos da robótica educacional:

- Objetivos Gerais

  - Contruir maquetes que utilizem lâmpadas, motores e sensores;
  - Trabalhar conceitos de desenho, física, álgebra e geometria;
  - Conhecer e aplicar princípios de eletrônica digital, e
  - Construir ou adaptar elementos dinâmicos como engrenagens, redutores de velocidade de motores, entre outros.

- Objetivos Psicomotores
  - Desenvolver a motricidade fina;
  - Proporcionar a formação de habilidades manuais;
  - Desenvolver a concentração e observação, e
  - Motivar a precisão de seus projetos.

- Objetivos Cognitivos
  - Estimular a aplicação das teorias formuladas às atividades concretas;
  - Desenvolver a criatividade dos alunos;
  - Analisar e entender o funcionamento dos mais diversos mecanismos físicos;
  - Ser capaz de organizar idéias a partir de uma lógica mais sofisticada de pensamento;
  - Selecionar elementos que melhor se adequem à resolução dos projetos;
  - Reforçar conceitos de matemática e geometria;
  - Desenvolver noções de proporcionalidade;
  - Desenvolver noções topológicas;
  - Reforçar a aprendizagem da liguagem Logo;
  - Introduzir conceitos de robótica;
  - Levar à descoberta de conceitos de física de forma intuitiva;
  - Utilizar conceitos aprendidos em outras áreas do conhecimento para o desenvolvimento de um projeto, e
  - Proporcionar a curiosidade pela investigação levando ao desenvolvimento intelectual do aluno.

- Objetivos Afetivos
  - Promover atividades que geram a cooperação em trabalhos em grupo;
  - Estimular o crescimento individual através da troca de projetos e ideias;
  - Garantir que o aluno se sinta interessado em participar de discussões e trabalhos em grupo;

- Desenvolver o senso de responsabilidade;
- Despertar a curiosidade;
- Motivar o trabalho de pesquisa;
- Desenvolver a autoconfiaça e a auto-estima, e
- Possibilitar a resolução de problemas por meio de erros e acertos.

2.1.4 Fundamentos da robótica móvel

O foco da robótica vem se modificando ao longo dos anos. Tempos atrás, os braços mecânicos, também chamados de manipuladores, tinham um maior destaque junto a mídia e a sociedade em geral (WOLF et al., 2009) sendo até um dos principais fatores para a Revolução Industrial. Nos tempos atuais, o destaque é a robótica móvel, que trata de máquinas capazes de se movimentar de forma independente, podendo ser terrestre, aquático, voador ou até espacial (VIEIRA, 2005 apud RINCON, 2014), movidos por rodas, esteiras, patas, hélices, dentre outros.

Segundo (WOLF et al., 2009), os Robôs Móveis Autônomos (RMA) possuem como características fundamentais:

> as capacidades de locomoção e de operação de modo semi ou completamente autônomo. Também deve ser considerado que maiores níveis de autonomia serão alcançados somente à medida que o robô passe a integrar outros aspectos considerados da maior importância, como: *capacidade de percepção* (sensores que conseguem "ler" o ambiente onde ele atua), *capacidade de agir* (atuadores e motores capazes de produzir ações, tais como o deslocamento do robô no ambiente), *robustez e inteligência* (capacidade de lidar com as mais diversas situações, de modo a resolver e executar tarefas por mais complexas que sejam).

O termo Controle Robótico Inteligente é referente ao uso de técnicas de planejamento e controle para a navegação e operação autônoma dos robôs, que permitem que os RMAs tenham a capacidade de executar as mais diversas e complexas tarefas. Esse controle inteligente é possível devido ao uso de diversos sensores e atuadores que combinados conferem ao robô a possibilidade de planejar e realizar o acionamento de seus dispositivos de modo a executar a ação desejada. Alguns sensores e atuadores para robôs móveis são listados, respectivamente, nas Figuras 3 e 4, a seguir apresentadas.

Sensor	Principal Função	Exemplos
De Posição e Orientação	Determinar a posição absoluta	GPS (Sistema de Posicionamento Global)
	ou direção de orientação do robô	Bússola [Compass]
		Inclinômetro
		Triangulação usando marcas (Beacons)
De Obstáculos	Determinar a distância até um objeto	Sensor Infra-Vermelho (IR - Infrared)
	ou obstáculo	Ultrassom (Sonar)
		Radar
		Sensor Laser (Laser rangefinder)
		Sistemas de Visão Estéreo (Stereo Vision)
De Contato	Determinar o contato com um objeto	Sensores de Contato (Bumbers, Switches)
	ou posição de contato com marcação	Antenas e "bigodes" (Animal whiskers)
		Marcações (barreiras óticas e magnéticas)
De Deslocamento	Medir o deslocamento do robô	Inercial (Giroscópio, Acelerômetros)
e Velocidade	Medidas relativas da posição e	Odômetro (Encoders: Optical, Brush)
	orientação do robô	Potenciômetros (Angular)
		Sensores baseados em Visão
Para Comunicação	Envio e recepção de dados e sinais	Sistemas de Visão e Sensores Óticos
	externos (troca de informação)	Sistemas de Comunicação (RF)
Outros tipos	Sensores magnéticos, indutivos, capacitivos, reflexivos	
	Sensores de temperatura, carga (bateria), pressão e força, etc.	
	Detectores: detector de movimento, de marcações, de gás/odores	

Figura 3 – Tipos de sensores para robôs móveis (WOLF et al., 2009)

Atuador	Principal Tipo/Função	Exemplos
Base Fixa	Braço robótico com base fixa	Robôs industriais PUMA
Base Móvel: Rodas	2 Rodas independentes (diferencial)	Robôs Khepera e Pioneer P3-DX
	3 Rodas (triciclo, omni-directionais)	Robô BrainStem PPRK
	4 Rodas (veículos robóticos - ackermann)	Stanley - Stanford (Darpa Challenge)
Base Móvel: Esteira	Esteira (Slip/Skid locomotion - tracks)	Tanques e veículos militares
Base Móvel: Juntas e	Bípedes	Robôs Humanóides
Articulações	4 Patas (quadpods)	Robôs Sony Aibo, BigDog
	6 Patas (hexapods)	Robôs Inseto (Lynxmotion Hexapods)
Base Móvel: Propulsão	Veículos aéreos com hélices	Aviões, Helicópteros e Dirigíveis
Hélices ou Turbinas	Veículos aquáticos com hélices	Barcos autônomos
	Veículos sub-aquáticos	Submarinos autônomos
Outros tipos	Braços manipuladores com base móvel	Garras (Grippers) embarcadas
	Garras com ou sem feed-back sensorial	Mão robótica
	Mecanismos de disparo	Disparo do chute (futebol de robôs)

Figura 4 – Tipos de atuadores para robôs móveis (WOLF et al., 2009)

Os sensores conferem ao robô uma visão parcial, incompleta e sujeita a erros, sendo tarefa do controle inteligente adquirir, unificar e tratar as informações providas pelos sensores. Os comandos dos atuadores também são imprecisos, pois estes estão sujeitos a erros de posicionamento do robô e acionamento dos motores, além das forças externas, como fricção, gravidade, colisão com obstáculos, derrapagem de rodas, dentre outros. Nesse contexto, cabe ao controle inteligente prover técnicas que permitam contornar estes cenários, corrigindo os erros de modo que as tarefas do robô sejam executadas corretamente (WOLF et al., 2009).

Um projeto de sistema de controle de robô deve levar em conta uma série de quesi-

tos como: i)tipo de tarefa do robô; ii)tipo e precisão dos sensores embarcados, e iii)tipo e precisão dos atuadores. Além de adotar uma arquitetura específica de controle que seja capaz de realizar algumas tarefas específicas ou, no melhor dos casos, todas as tarefas, sendo elas: i) fusão de sensores; ii)desviar de obstáculos; iii)auto-localização; iv)mapeamento do ambiente v)planejamento de trajetórias; vi)planejamento de ações; vii)navegação robótica e viii)interação e comunicação. O detalhamento de cada tópico pode ser conferido na íntegra em (WOLF et al., 2009). Para que essas tarefas sejam planejadas e executadas gerenciando todos os dispositivos embarcados no robô, é necessário fazer o uso das arquiteturas computacionais de controle de robôs móveis autômatos. Essas arquiteturas computacionais são relativas à percepção, raciocínio/decisão e ação em relação ao ambiente (WOLF et al., 2009). Dentre os diversos tipos de arquiteturas computacionais, as mais conhecidas e reconhecidas são controle reativo, controle deliberativo, controle hierárquico e controle híbrido. Cada uma delas é descrita a seguir.

2.1.4.1 Controle reativo

O controle reativo consiste em um sistema de reação sensorial-motora que considera apenas as leituras sensoriais realizadas no presente para fins de tomada de decisão e geração de comandos de ação (WOLF et al., 2009). Esse tipo de arquitetura computacional tem por objetivo possibilitar a implementação de sistemas de controle com rápida resposta a um grande número de ocorrências ou situações de ambiente, permitindo que os robôs atuem em ambientes dinâmicos, devido a simplicidade no tratamento das informações sensoriais e a forma direta pela qual a percepção está associada com a ação (GRASSI, 2006 apud SANTOS, 2009).

(WOLF et al., 2009) defendem que o sistema reativo é muito útil para implementar comportamentos.

> Um sistema reativo é bastante útil para implementar comportamentos elementares como: desviar de obstáculos (*avoid collision behaviour*: reage a presença de um obstáculo), seguir um objeto (*wall-following behaviour*: acompanhar um elemento guia) e seguir uma fonte luminosa (*phototaxis behaviour*: mover em direção a uma fonte de luz).

2.1.4.2 Controle deliberativo

A arquitetura deliberativa consiste em aplicar um mecanismo de planejamento e tomada de decisão que estabeleça um plano prévio de execução de uma sequência de ações, baseado no modelo interno de conhecimento do mundo que o robô possui. Esse conhecimento prévio do mundo confere ao robô a possibilidade de otimizar o desempenho em relação ao modelo interno.

O robô pode ter seu modelo interno representado de duas formas diferentes: simbólica e geométrica. O modelo simbólico é baseado em lógica utilizando, muitas vezes,

a inteligência artificial. Já no modelo geométrico o mundo é representado espacialmente, por espaços livres e com obstáculos (SANTOS, 2009).

Tendo em vista que o robô possui esse modelo interno de mundo previamente estabelecido, ele possui limitações quando colocado em um mundo dinâmico, pois será necessário reconhecer o novo obstáculo, acrescentar ao seu modelo e reestabelecer o plano de execução (SANTOS, 2009). Devido a essa limitação, (WOLF et al., 2009) dizem que o melhor seria a combinação dos modelos reativos e deliberativos, assim o robô teria capacidade de reação e de planejamento de tarefas complexas.

2.1.4.3 Controle hierárquico

O controle hierárquico consiste na combinação de múltiplos módulos de controle (reativo e/ou deliberativo) dispostos em forma hierárquica, podendo ser com decomposição vertical ou horizontal para definir um esquema de prioridades em relação às múltiplas camadas do sistema (WOLF et al., 2009).

O controle hierárquico com decomposição vertical possui os módulos de controle reativo dispostos de forma vertical, mantendo a prioridade de baixo para cima, sendo a camada mais baixa a mais prioritária. Um exemplo de uso desse modelo é quando a ação de desviar de um obstáculo, que está mais baixa, tem prioridade em relação a ação de seguir em frente, que está mais acima.

O controle hierárquico com decomposição horizontal é adotado para controle de módulos deliberativos, implementando um *"pipeline"* que decompõe as funções em camadas executadas em cascata, formando uma linha de ação horizontal. Um exemplo desse modelo é o tipo *Sense-Model-Plan-Act* (SMPA) que possui um longo ciclo de resposta ao estímulo de entrada devido as suas quatro camadas horizontais, as quais serão executadas em cascata para cada estímulo.

2.1.4.4 Controle híbrido

A arquitetura de controle híbrido nada mais é do que um controle reativo sendo planejado por um controle deliberativo. Inicialmente, é realizado um plano de execução pelo módulo deliberativo. Depois de pronto, o plano é executado por um módulo reativo de maior prioridade, o qual pode intervir nesse plano quando encontra algum obstáculo em meio à execução das ações. Nesse controle, pode-se ainda ter um módulo de autolocalização que monitora constantemente a posição atual do robô e a corrige, se necessário.

Este tipo de arquitetura é uma das mais sofisticadas e adotadas na implementação dos sistemas de controle dos robôs móveis autônomos modernos, devido a sua eficiência em atingir os objetivos do robô.

2.2 Engenharia de software

O estudo da robótica na Universidade tem um foco considerável na inteligência artificial, que é um dos métodos da otimização de problemas NP. A seguir são brevemente descritos os dois algoritmos de otimização abordados por este trabalho, o algoritmo guloso e a programação dinâmica, bem como o paradigma lógico, também tema deste trabalho.

2.2.1 Algoritmo guloso

Os algoritmos gulosos tomam uma decisão levando em consideração apenas a informação disponível no dado momento, sem levar em conta as consequências da decisão, nunca reconsiderando uma decisão já tomada. Dessa forma, o algoritmo guloso toma uma decisão ótima naquele momento (decisão ótima local), e espera que essa decisão seja também a ótima global (ROCHA, 2004).

2.2.1.1 Características do algoritmo guloso

Segundo (ROCHA, 2004), os problemas e os algoritmos gulosos que os resolvem são caracterizados pelos itens a seguir.

- Para que exista uma solução ótima de um problema deve existir um conjunto de candidatos;
- Ao executar um algoritmo guloso dois conjuntos são criados: um que contêm os elementos que foram analisados e rejeitados, e outro os elementos que foram analisados e escolhidos;
- Um algoritmo guloso contêm quatro funções:
  - a primeira avalia se o conjunto candidato produz uma solução para o problema;
  - a segunda verifica a viabilidade do conjunto de candidatos, ou seja, se é possível acrescentar mais candidatos a esse conjunto;
  - a terceira função, chamada de função seleção, avalia qual dos candidatos restantes é o melhor para ser acrescentado no conjunto;
  - a quarta e última função, chamada função objetivo, retorna o valor da solução encontrada.

É possível dizer que um algoritmo guloso trabalha da seguinte forma: inicialmente contém um conjunto S que está vazio, ou seja, nenhum candidato foi escolhido. A cada passo a função seleção é utilizada para determinar qual o melhor candidato. Se o elemento analisado não é viável, ignora-se o termo que está sendo avaliado no momento. Se for viável, adiciona-se o elemento ao conjunto S. O elemento avaliado, sendo ele aceito ou

rejeitado, será avaliado apenas uma vez, não havendo reconsiderações. A cada vez que o conjunto de candidatos escolhidos (S) é ampliado, é também verificado se a solução do problema foi obtida (ROCHA, 2004).

2.2.1.2 Elementos da estratégia gulosa

Como exposto, a estratégia utilizada pelo algoritmo guloso para encontrar a solução ótima do problema em questão é, em cada passo, escolher a solução que parece ser ótima naquele momento. Nem sempre uma estratégia gulosa é capaz de chegar à solução de um problema, mesmo esta existindo. Existem duas características que podem indicar se o problema pode ou não ser resolvido através de uma estratégia gulosa.

- **Propriedade da escolha gulosa:** segundo a propriedade da escolha gulosa, uma solução globalmente ótima pode ser alcançada escolhendo uma solução localmente ótima. Para se provar que uma escolha localmente ótima em cada um dos passos irá levar a uma solução ótima, é necessário examinar uma solução ótima global para algum subproblema. Posteriormente, deve-se mostrar que essa solução pode ser modificada em uma solução gulosa. Tal estratégia irá resultar em um subproblema menor, mas similiar.
- **Subestrutura ótima:** esta característica também é importante quando se utiliza a programação dinâmica, vide Seção 2.2.2. Um problema possui uma subestrutura ótima quando uma solução ótima para o problema contém, dentro dela, soluções ótimas para os subproblemas.

2.2.1.3 Fundamentos do algoritmo guloso

A teoria de algoritmos gulosos envolve o conceito de matróide que por definição é um par (E,I) onde E é um conjuto finito e I $\subseteq 2^E$, que satisfaz os seguintes axiomas (CASTALONGA; LEMOS et al., 2007):

(i) $\emptyset \in$ I;

(ii) Se X $\subseteq$ Y e Y $\in$ I, então X $\in$ I;

(iii) Se X, Y $\in$ I e |X| < |Y|, então existe e $\in$ Y - X tal que X $\cup$ e $\in$ I.

Esse tipo de estrutura não cobre todo tipo de problema que pode ser resolvido pelo algoritmo guloso, mas cobre muitos casos de interesse prático.

Lema: todo conjunto independente maximal em um matróide tem o mesmo tamanho. **Prova:** Por contradição. Se existem dois conjuntos independentes maximais A e B e |A| < |B|, deve existir um elemento x $\in$ (B - A) tal que (A + x) $\in$ I. Isso contradiz o fato de que A é um conjunto maximal.

2.2.2 Programação dinâmica

A programação dinâmica, assim como os métodos de "dividir para conquistar", combina as soluções dos subproblemas para resolver um problema. Esses métodos particionam os problemas em subproblemas de forma que, resolvendo os subproblemas recursivamente, resolve-se o problema. Já a programação dinâmica aplica-se quando os subproblemas se particionam em subsubproblemas. Um algoritmo comum resolve repetidamente essas partições, trabalhando mais que o necessário. Em contrapartida, o algoritmo dinâmico resolve cada subsubproblema apenas uma vez, salvando o resultado em uma tabela, evitando calcular repetidas vezes o mesmo problema (CORMEN, 2009). Um problema pode ter várias soluções possíveis, cada uma com um valor, e a solução com o melhor valor (maior ou menor) é dita a solução ótima para o problema. Segundo (CORMEN, 2009), para se desenvolver uma solução de programação dinâmica, devem ser seguidos quatro passos:

1. Caracterizar a estrutura de uma solução ótima;
2. Recursivamente definir o valor de uma solução ótima;
3. Calcular o valor de uma solução ideal, normalmente de forma ascendente, e
4. Construir uma solução ótima a partir de informações computadas.

Os três primeiros passos formam a base de uma solução dinâmica. Mas quando utilizar a programação dinâmica para resolver um problema? O primeiro passo para resolver um problema de otimização por programação dinâmica é caracterizar uma subestrutura ótima, sempre que um problema exibí-la. Para caracterizar o espaço de subproblemas, deve-se manter o espaço o mais simples possível e, em seguida, expandí-lo se necessário. Uma subestrutura ótima varia entre os domínios de problema de duas formas:

(i) quantos subproblemas uma solução ideal utiliza para o problema, e

(ii) quantas escolhas existem para determinar quais subproblemas usar em uma solução ótima.

Ou seja, pode ser que um problema tenha apenas um subproblema (de tamanho n - i), mas deve-se considerar n escolhas para i, afim de determinar quais escolhas produzem uma solução ótima.

2.2.2.1 *Memoization* vs Iteração sobre subproblemas

A palavra *memoization* vem do termo *memoize* que deriva de um termo latim *memorandum* que quer dizer relembrando. Essa é uma técnica *top-down* que armazena em

uma tabela a solução de cada subsubproblema, evitando assim o retrabalho e aumentando a eficiência. Essa técnica utiliza um algoritmo recursivo.

A iteração sobre subproblemas é uma técnica *bottom-up* que utiliza um algoritmo iterativo para calcular os subproblemas. Essa técnica começa a calcular do menor subproblema para o maior.

2.2.3 Paradigma lógico

Os literais são fórmulas atômicas positivas ou negativas, que, em conjunto, formam uma cláusula. Um tipo de cláusula é a chamada cláusula de Horn (ou cláusula definitiva), onde cada cláusula tem no máximo um literal positivo. Essas cláusulas são utilizadas pela resolução SLD (*Selection-rule driven Linear resolution for Definite clauses*), que restringe o uso de literais para tornar as implementações mais eficientes. Essa resolução recebe como entrada um programa lógico P e uma consulta N, e efetua a resolução entre a consulta N e alguma regra de P. O ganho de eficiência vem do fato de que, para cada regra é possível a seleção de apenas um literal para a resolução com N (RODRIGUES, 2010). A programação lógica surgiu da aplicação da SLD no processamento da linguagem natural da inteligência artificial, e esse processamento serviu como primeira aplicação e motivo imediato para o desenvolvimento do Prolog (*Programation en Logique*) (RODRIGUES, 2010).

2.2.3.1 Prolog

O Prolog é uma linguagem de programação utilizada para resolver problemas que envolvam objetos e relações entre objetos. O termo objeto em Prolog não se refere a uma estrutura de dados que pode herdar variáveis e métodos de uma classe, mas se refere às coisas que podemos representar usando termos (CLOCKSIN; MELLISH, 2003). Um programa Prolog consiste em um conjunto de cláusulas, no qual cada cláusula ou é um fato sobre a informação dada ou uma regra sobre como a solução pode relacionar ou ser inferida a partir dos fatos dados (CLOCKSIN; MELLISH, 2003). Segundo (CLOCKSIN; MELLISH, 2003), a programação em Prolog consiste em:

- especificar algum fato sobre os objetos e seus relacionamentos;
- definir algumas regras sobre objetos e seus relacionamentos, e
- fazer perguntas sobre os objetos e seus relacionamentos.

O Prolog pode fazer muito mais do que apenas responder sim ou não às perguntas feitas, ele permite que um computador seja usado como um armazém de fatos e regras, fornecendo maneiras de se fazer inferências a partir de um fato ou outro, e encontrar os valores das

variáveis que levam a uma dedução lógica. Algumas definições base da linguagem Prolog são descritas a seguir.

- **Fato:** um fato é a definição das relações entre os objetos. Para implementar essas definições, algumas regras devem ser seguidas i) os nomes de todas as relações e objetos devem começar com letra minúscula; ii) a relação é escrita primeiro seguida do(s) objeto(s) entre parênteses (caso tenha mais de um, separar com vírgula); iii) um fato deve terminar com ponto-final. A relação entre os objetos é chamada de predicado, e os objetos da relação são chamados de argumentos.

- **Questão:** a questão tem a escrita muito parecida com o fato, exceto que começa com um ponto de interrogação. Para cada pergunta feita ao interpretador de Prolog, ele irá procurar na sua base de conhecimento fatos ou regras que comprovem aquele questionamento.

- **Variáveis:** é um nome com letra maiúscula. Quando utilizado em uma pergunta, faz com que o intepretador de PROLOG procure na base de conhecimento um ou mais objetos que sejam equivalentes àquela variável. Por exemplo, em uma determinada base de conhecimento exitem três fatos:

```
gosta(maria, pedro).
gosta(maria, joao).
gosta(laura, joao).
```

  Quando for perguntado ao prolog:

```
?- gosta(maria, X).
```

  O interpretador Prolog responderá:

```
X = pedro;
X = joao.
```

- **Conjunção:** é o operador vírgula (,) tendo função de "e". Quando utilizado na pergunta, o interpretador de Prolog procura na base de conhecimento fatos que comprovem as duas perguntas. Usando a base de conhecimento do item anterior, se for perguntado ao interpretador de Prolog:

```
?- gosta(maria, X) , gosta(laura, X).
```

  ele responderá:

```
X = joao.
```

  Para responder a esse questionamento, o interpretador de Prolog procura primeiramente qual o objeto que pode ser a variável X, achando inicialmente o argumento

pedro. Em seguida, ele procura se existe algum fato que Laura goste de Pedro. Caso não exista esse fato na base de conhecimento, ele volta para buscar outro objeto que possa ser a variável X, encontrando o argumento *joão*. Ao buscar por algum fato que comprove que Laura gosta de João, o interpretador encontra e retorna "sim"para a pergunta, ou seja, Maria gosta de João e Laura também. Caso existissem mais pessoas em comum que Maria e Laura gostassem, era só digitar ";"seguido de "Enter"e o interpretador de Prolog buscaria por outros objetos para X.

- **Regras:** uma regra é uma declaração geral sobre objetos e seus relacionamentos. Em Prolog, uma regra é comumente utilizada para dizer que um fato depende de um conjunto de outros fatos. Cada regra contém uma cabeça e um corpo, ligados pelo símbolo ":- ", que é constituído por dois pontos e um hífen, e é pronunciado "se".

2.2.4 Otimização

Lobato (2008) defende que a otimização pode ser classificada segundo as seguintes categorias: (i) Por tentativa e erro ou Por função; (ii) Unidimensional ou Multidimensional; (iii) Dinâmica ou Estática; (iv) Discreto ou Contínuo; (v) Restritos ou Não Restritos; (vi) Clássicos ou Randômicos e, (vii) Mono-objetivo ou Multi-objetivos.

A otimização por tentativa e erro se dá quando não se tem uma formulação matemática para o processo, fazendo-se necessário ajuste de parâmetros, já a otimização por função é o oposto, é quando se conhece o processo através de sua formulação matemática (LOBATO, 2008).

Os problemas de otimização unidimensional caracterizam-se por haver apenas uma variável de entrada, enquanto na multidimensional podem existir múltiplas variáveis (LOBATO, 2008).

Otimização dinâmica tem o resultado com dependência do tempo, ou seja, a variável tempo influencia na saída da otimização, já a otimização estática não tem tal dependência (LOBATO, 2008).

O processo de otimização discreta é caracterizado por ter um número finito para os valores das variáveis de entrada, enquanto que na otimização contínua os valores podem ser infinitos (LOBATO, 2008).

A otimização restrita é aquela que apresenta algum tipo de restrição no processo, já a não restrita permite que o processo seja executado sem tipo de variável restritiva algum (LOBATO, 2008).

A otimização clássica tenta alcançar a solução ótima através de uma sequência de passos, partindo de uma configuração inicial do problema, enquanto que a otimização

randômica utiliza um conjunto de configurações iniciais do problema, baseado em cálculos de probabilidade, para o alcance da solução ótima (LOBATO, 2008).

Por fim, a otimização mono-objetivos é aquela que é representada por uma única função objetivo, enquanto que a multiobjetivo busca atender a várias funções objetivos simultaneamente (LOBATO, 2008).

O problema abordado no presente TCC assemelha-se ao problema da mochila. Este último consiste em: dado um conjunto de itens, cada um com uma massa e um valor associado, determinar quais itens devem ser incluídos no conjunto de modo que a massa total seja menor ou igual à capacidade máxima da mochila e o valor do conjunto seja o maior possível (PRAKASAM; SAVARIMUTHU, 2015).

Baseado nessa definição, e nas definições das categorias de otimização citadas, o problema da mochila é um problema de otimização discreta, pois lida com valores de variáveis finitas, ou seja, o número de itens que serão analisados é finito, assim como a massa e o valor dos mesmos.

2.3 Considerações parciais

Trabalhar a robótica no âmbito escolar, colabora no aprendizado de forma multidisciplinar. A robótica móvel está sendo muito utilizada pelas escolas, principalmente o kit da LEGO, para trabalhar nas crianças a parte motora, ao montar o robô; a parte intelectual, para construir os programas; e a parte social, para aprender a trabalhar em grupo.

Já na universidade, a robótica está muito associada à ideia de inteligência artificial, tendo um foco maior para as áreas de automação e otimização. Na área de otimização, o algoritmo guloso e a programação dinâmica são conceitos bem trabalhados e conhecidos. A utilização da linguagem de programação Prolog na área da robótica não é tão vasta quanto de outras linguagens, como C e C++, mas está sendo cada vez mais adotada.

Diante do conteúdo estudado neste capítulo, o foco desse trabalho se dará na utilização da programação dinâmica em conjunto com a linguagem de programação Prolog e o kit de robótica móvel Lego Mindstorms NXT.

3 Suporte tecnológico

Para dar suporte ao desenvolvimento do projeto foi feito um levantamento de ferramentas candidatas, sendo este refinado e descrito nesse capítulo. Assim como o capítulo anterior, este capítulo, de suporte tecnológico, foi organizado em dois escopos, um mais voltado ao suporte tecnológico investigado no domínio da Robótica Educacional e outro mais voltado à Engenharia de Software.

3.1 Robótica educacional

Um dos maiores precursores da robótica no âmbito educacional foi Seymour Papert ao criar a linguagem LOGO com o intuito de incentivar a aprendizagem da matemática, baseado nas idéias do suiço Jean Piaget que dizia, *"as funções essenciais da inteligência consistem em compreender e inventar, em outras palavras, construir estruturas estruturando o real"*, ou seja, é essencial para a formação da inteligência a ação sobre objetos, e por meio dessa descobrir propriedades através de abstração. Desde então, a robótica educacional vêm se tornando uma plataforma atraente para criar envolvimento nos estudantes, incentivando o estudo da ciência e da tecnologia. As plataformas iterativas/lúdicas vêm ganhando espaço ao longo dos anos como ferramentas auxiliares às metodologias de ensino que utilizam a robótica educacional. Dentre essas plataformas pode-se citar: Robomind (ROBOMIND...,), Scratch[1], Arduino[2], Raspberry Pi[3] e o Kit Lego Mindstorms[4]. Essas plataformas serão brevemente descritas a seguir.

3.1.1 Robomind

O Robomind é uma IDE para a programação dos movimentos de um robô em um mundo bidimensional, por meio de uma linguagem de programação bem simples e intuitiva, ideal para iniciantes. A interface da IDE é composta por quatro partes: à cima contêm o menu e os atalhos para as funções mais utilizadas, como inserir qualquer movimento; à esquerda encontra-se a área de edição do código; à direita contém o mundo bidimensional, no qual é possível visualizar o robô realizando os movimentos, previamente determinados pelo código, em um mapa; e na parte inferior encontra-se o controle de execução do programa juntamente à área de mensagens utilizadas pela IDE para informar erros sintáticos ou durante a execução do código, e a situação na qual o robô se encontra.

[1] www.scratch.mit.edu
[2] www.arduino.cc
[3] www.raspberrypi.org
[4] mindstorms.lego.com

O Robomind é gratuito para teste durante 30 dias. Após esse periodo, é possível comprá-lo por USD 14.00 por ano. Ele pode ser instalado em vários idiomas, dentre eles o Português[5], além de ser multiplataforma (Linux, Mac e Windows). Com o Robomind só é possível simular o comportamento do que é implementado, não trabalhando a montagem de um robô, programação embarcada e utilização de sensores.

3.1.2 Scratch

O Scratch é um ambiente e uma linguagem de programação multimídia, produzido pelo *MediaLab*[6] do MIT, para ser utilizado por crianças a partir dos oito anos de idade na criação de histórias iterativas, animações, jogos, músicas, dentre outros, ditos projetos Scratch. Um projeto Scratch contém personagens programáveis que podem tratar eventos vindos do teclado ou do mouse, para mudança de posição e direção, dentre outras funcionalidades. Assim como o Robomind, ele utiliza uma linguagem de programação em blocos, com encaixe seletivo ideal para programadores iniciantes. O Scratch é gratuito, *online* e também tem a opção de utilizá-lo em português. Similar ao Robomind, no Scratch só é possível simular de forma limitada a implementação, não sendo possível testar em um robô físico nem utilizar sensores.

3.1.3 Arduino

O Arduino é uma plataforma de prototipagem eletrônica *open-source*, criada em 2005 pelo italiano Massimo Banzi, para auxiliar no ensino de eletrônica visando o baixo custo para os alunos. A plataforma é composta por um *hardware* e um *software* bastante flexíveis. O *hardware* é constituído de uma placa com um microprocessador; porta USB, usada para comunicação serial com o computador; pinos digitais, utilizados para detecção ou transmissão de controles digitais; pinos analógicos, usados para leitura de sinais de sensores; e pinos de alimentação, usados para alimentação de circuitos externos. O *software* é uma IDE, na qual é programado o código, conhecido como *sketch*, e por meio desta será passado à placa através de uma comunicação serial. Nesta IDE, é utilizada a linguagem de programação Arduino, mas quando é passado para a placa, esta linguagem é traduzida para a linguagem estruturada C. A IDE é multiplataforma, podendo ser instalada em Linux, Mac e Windows. O Arduino tem a vantagem de utilizar programação embarcada no robô e possuir entradas para diversos sensores, porém a montagem do robô fica a critério do desenvolvedor não tendo padronização alguma.

[5] www.robomind.net/pt/
[6] www.media.mit.edu

3.1.4 Raspberry Pi

O Raspberry Pi é um computador versátil e flexível, criado pela *Raspberry Pi Foundation*[7] em conjunto com a Universidade de Cambrigde, no ano de 2012, com o objetivo de estimular o ensino da ciência da computação em jovens do ensino básico. O Raspberry Pi possui dois modelos, ambos equipados com processador multimídia *Broadcom* BCM2835 *system-on-chip* (SoC) de 700 Mhz com placa gráfica integrada VideoCore IV; entrada para cartão de memória, que se faz necessário pois o Raspberry Pi não possui armazenamento interno; interface HDMI; entrada USB e RCA; processador ARM 7 com capacidade de processamento de 32 bits, não sendo possível a instalação do Windows, mas aceitando qualquer distribuição Linux, como o Raspbian que é baseado no sistema operacional Debian. O que difere nas duas versões é a memoria RAM. Um modelo é oferecido com 256 MB e o outro com 512 MB. Assim como o Arduino, a desvantagem do Raspberry Pi é a não padronização da montagem dos robôs. Entretanto, é possível a utilização de sensores e programação embarcada.

3.1.5 Kit educacional Lego Mindstorms

A LEGO lançou em janeiro de 2006, na feira *Consumer Electronics Show*[8] em Las Vegas, a linha Mindstorms NXT, que é uma versão mais avançada que a já consagrada RCX, possuindo um processador Atmel ARM 32 bits com *clock* de 48MHz, HD de 256KB de memória *flash*, memória RAM de 64KB, software próprio, sensores de luz, toque e som. Esse suporte confere ao robô noções de distância, sendo inclusive capaz de reagir a movimentos, ruídos e cores. Essa nova versão também executa movimentos com maior grau de precisão que seu antecessor. O kit Lego Mindstorms é bem flexível quanto à montagem do robô, porém a Lego disponibiliza um manual de montagem de um robô padrão; uma vantagem ao se comparar com os concorrentes mencionados anteriormente. A escolha do kit Lego Mindstorms para o dado projeto se deu, além da vantagem já conhecida da padronização do robô, por ele já conter os sensores necessários no kit e por esse material ser utilizado na matéria Princípios de Robótica Educacional, ministrada na FGA pelo prof. Dr. Maurício Serrano, co-orientador deste projeto. Como o kit tem uma liguagem de programação própria, e a liguagem de programação escolhida por este trabalho foi o Prolog, se fez necessário a busca de uma ferramenta que tornasse possível essa integração, sendo esta ferramenta brevemente descrita a seguir.

3.1.6 Traveller

O *framework* Traveller (RINCON, 2014), contém algoritmos para definir qual caminho o robô deve seguir de modo a chegar a um destino de forma rápida e eficiente.

[7] www.raspberrypi.org/about/
[8] www.cesweb.org

Utilizando o robô Lego Mindstorms NXT, o Traveller definirá qual será a trajetória para o robô alcançar determinado ponto no mapa desviando dos obstáculos. O Prolego, após definir o roteiro de missões, irá passar os pontos iniciais e finais das missões, o Traveller, por sua vez, será responsável em executar o trajeto.

3.2 Engenharia de software

De modo a gerenciar o processo de desenvolvimento deste trabalho, bem como desenvolver, manter e disponibilizar o código e artefatos gerados, o uso de algumas ferramentas se faz necessário. Tais ferramentas, que serão utilizadas por este trabalho, são brevemente descritas a seguir.

3.2.1 SWIProlog

O SWIProlog é uma implementação, de código aberto, para a linguagem de programação Prolog, executando em modo texto, por meio de comandos no terminal do sistema. Sob a licença *Lesser GNU Public License*[9], pode ser utilizado nas plataformas Windows, Linux e MacOS. Possui diversas ferramentas de edição gráfica, tais como: J-Prolog Editor e SWI-Prolog-Editor. Permite ainda a utilização da linguagem Prolog por outras linguagens, tais como: C/C++ e Java.

3.2.2 Eclipse

O Eclipse[10] é uma IDE para desenvolvimento, que suporta diversas linguagens, dentre elas Java, C, C++, Python, PHP, dentre outras. Possui o código livre, e a versão a ser utilizada por esse trabalho é a Mars 2.0, especificamente para trabalhar com o código Java do *framework* Traveller.

3.2.3 NotePad++

O NotePad++[11] é um editor de texto com código aberto sob a licença GPL, suporta diversas linguagens de programação, e será utilizada por esse trabalho para desenvolver o código em Prolog.

[9] http://www.gnu.org/licenses/lgpl.html
[10] https:eclipse.org
[11] https:notepad-plus-plus.org

3.2.4 Bizagi Process Modeler

O BizAgi Process Modeler[12] é um aplicativo gratuito utilizado para criar e documentar modelos de processos em BPMN[13] (*Business Process Model and Notation*). Esse aplicativo faz parte de uma suíte de software, chamada Bizagi, composta por dois produtos: o Bizagi Process Modeler e o BizAgi BPM Suite.

3.2.5 Git

O Git[14] é um dos mais consagrados e utilizados sistemas de controle de versão. É *open-source* e gratuito, distribuído sob a licença GNU GPLv2[15], e projetado para lidar com qualquer tamanho de projeto mantendo a rapidez e a eficiência.

3.2.6 Github

O GitHub é um repositório web planejado para utilizadores do sistema de controle de versão Git. Possui opções de utilização de repositórios privados e públicos, sendo o pago e gratuito, respectivamente. No GitHub, é possível, dentre tantas outras funcionalidades, visualizar o código no navegador, criar *issues*, revisar e aceitar mudanças, baixar o repositório como *.zip*, criar *branchs* ou até mesmo fazer o *fork* de outro repositório.

3.2.7 LaTeX

O Latex[16] é um sistema de preparação de documentos para composição tipográfica de alta qualidade, desenvolvido inicialmente por Leslie Lamport, em 1985, baseado na linguagem TeX criada por Donald E. Knuth, no final da década de 70, na Universidade de Stanford. Atualmente, o Latex é mantido e desenvolvido pelo *The Latex3 Project*[17], sendo disponibilizado gratuitamente. O Latex é mais amplamente utilizado no meio técnico-científico para escrita de documentos de médio a grande porte, devido a sua facilidade de produzir fórmulas e símbolos matemáticos.

3.2.8 TexMaker

O TeXMaker[18] é um editor de texto Latex gratuito, multiplataforma para Linux e MacOSX, com suporte a unicode, verificação ortográfica, auto-completar e contém um visualizador de pdf embutido. Com uma interface simples, o TexMaker contém botões de

[12] http://www.bizagi.com/en/bpm-suite/bpm-products/modeler
[13] www.bpmn.org
[14] http://www.git-scm.com/
[15] http://www.gnu.org/licenses/gpl-2.0.html
[16] http://www.latex-project.org/
[17] http://latex-project.org/latex3.html
[18] http://www.xm1math.net/texmaker/

atalho para as funções mais utilizadas como: estruturas (part/chapter/section), referências (ref/cite), tamanho e estilo de letra, negrito, itálico, alinhamento (direta/esquerda, centro) e inserção de diversos símbolos matemáticos.

3.3 Considerações parciais

As ferramentas descritas nesse capítulo foram investigadas com o intuito de verificar as diversas formas de se trabalhar com a robótica na área educacional e auxiliar na compreensão do domínio do problema. O estudo dessas ferramentas serviu de insumo para a elaboração da proposta do algoritmo de programação dinâmica, produto de trabalho desse TCC. Já as ferramentas relacionadas à engenharia de software citadas nesse capítulo visam auxiliar no desenvolvimento do trabalho, tanto na parte escrita, quanto na de planejamento e implementação.

4 Prolego: A máquina de raciocínio

Este capítulo traz o detalhamento dos componentes que integram o Prolego, bem como a descrição da montagem do robô, do tapete, das missões, da arquitetura do robô e do Prolego. Este capítulo aborda ainda a configuração do ambiente e a integração do Prolego com o Traveller.

4.1 Contextualização

A proposta deste trabalho foi a construcão de uma máquina de raciocínio, utilizando programação dinâmica e implementada na linguagem Prolog, que fosse capaz de gerar um roteiro de missões otimizado, isto é, que conseguisse a maior pontuação em um determinado tempo. As entradas da máquina de raciocínio são: o tempo total para a execução das missões e uma base de conhecimento que contém as missões ainda não realizadas. A partir dessas entradas, a máquina de raciocínio avalia quais missões devem ser executadas para que se alcance a quantidade máxima de pontos possível. A base de conhecimento contendo as missões a serem executadas foi implementada seguindo a seguinte estrutura:

```
missao(Valor, Tempo).
```

Desta forma, o Prolego avalia o valor e o tempo, em segundos, para executar cada missão. Visando limitar o escopo de atuação, foram estabelecidos alguns pontos para este trabalho:

- toda missão tem início no ponto X = 0 e Y = 0 e, a ele, deve retornar, pois se tal limitação não fosse estabelecida, o tempo para execução de cada missão seria variável de acordo com a localização;
- para contabilizar a pontuação da missão, não foram consideradas possíveis penalidades;
- o tempo total da missão é igual ao tempo de deslocamento (tempo necessário pro robô sair do ponto inicial e chegar ao ponto final) somado ao tempo de execução da missão;
- o tempo de deslocamento da missão foi medido utilizando o caminho gerado pelo Traveller de acordo com os pontos inicial e final de cada missão, e
- o tempo de execução da missão foi uma estimativa, considerando o tempo que o robô leva para fazer o movimento acrescido de uma margem de erro de 2 segundos.

Para estimar esse tempo, foram analisados alguns vídeos[1,2] de competição da turma de Princípios de Robótica Educacional da UnB - Faculdade Gama.

A Figura 5 demonstra a comunicação entre: o Prolego, o Traveller e os módulos do Traveller entre si.

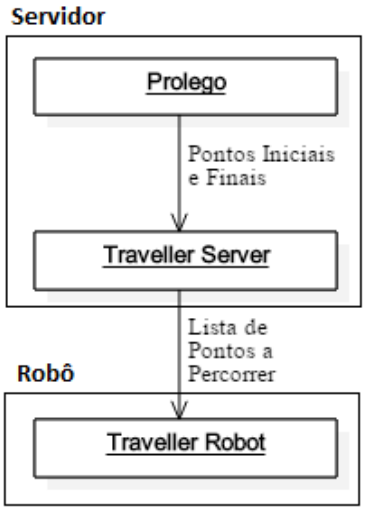

Figura 5 – Comunicação entre o Prolego e o Traveller Framework

Inicialmente, o Prolego seria hospedado no robô mas, como se pode observar na Figura 5, a comunicação do mesmo é feita com o módulo Server do Traveller. Logo, por questão de otimização, o Prolego ficou hospedado no servidor e a comunicação com o robô é realizada via *Bluetooth*.

4.2 Arquitetura do robô

(VIEIRA, 2005 apud RINCON, 2014) define uma arquitetura para robôs móveis baseada em cinco camadas: percepção, decisão, planejamento de caminho, geração de trajetória e sistema de controle. Essa arquitetura pode ser visualizada na Figura 6.

A primeira camada, denominada Camada de Percepção, trata da percepção do robô acerca do mundo ao seu redor, utilizando os sensores que o compõe. Essa camada é responsável, por exemplo, por identificar os obstáculos contidos no tapete. Neste trabalho, os obstáculos foram mapeados manualmente.

A segunda camada, denominada Camada de Decisão, é a responsável por decidir as ações que o robô irá executar. Esta camada é o cerébro do robô, sendo nela que o trabalho proposto age. A máquina de raciocínio que atua nesta camada tem o cálculo da

[1] https://www.youtube.com/channel/UCDWIi63k2Io4hFEag19QE0Q
[2] https://www.youtube.com/channel/UCd1jb1ci3ERC_5V1Qo3LGw

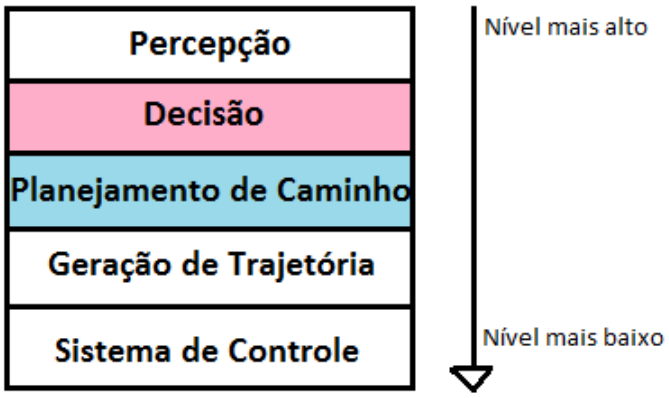

Figura 6 – Arquitetura em camadas de robôs móveis - (VIEIRA, 2005 apud RINCON, 2014)

trajetória, definida na quarta camada, através do *framework* Traveller. Com o tempo da trajetória conhecida e somado ao tempo de execução da missão, tem-se o tempo total da missão, e a máquina de raciocínio decide quais missões devem ser executadas.

A terceira camada define a trajetória que o robô irá executar para chegar à posição desejada, é nessa camada que o *framework* Traveller age. Com o conhecimento do ambiente, anteriormente levantado pela Camada de Percepção, o Traveller define um percurso por onde o robô deve seguir para não colidir com nenhum obstáculo.

A quarta camada recebe o plano da trajetória feito pela camada anterior e define quais ações devem ser feitas sobre o *hardware* para executar o plano. Essa camada tem conhecimento das dimensões e limitações do robô. A última camada atua diretamente no *hardware* para garantir que os atuadores estão recebendo o sinal e agindo conforme planejado.

4.3 Montagem do robô

O robô utilizado por este trabalho é o Bauen, um tipo de montagem do robô NXT. Nessa montagem, o robô é feito com três motores, dois para as esteiras e um preparado para receber uma garra ou outro adereço. O motor da esquerda deve ser ligado na porta B, o da direita na porta C e o da "garra"na porta A. O tutorial completo, contendo o passo-a-passo para a montagem do robô pode ser encontrado no site da LEGO[3].

Na Figura 7, é possível visualizar a aparência do robô quando montado.

[3] http:lego.brickinstructions.comlego_instructionsset8547Mindstorms_NXT_2.0

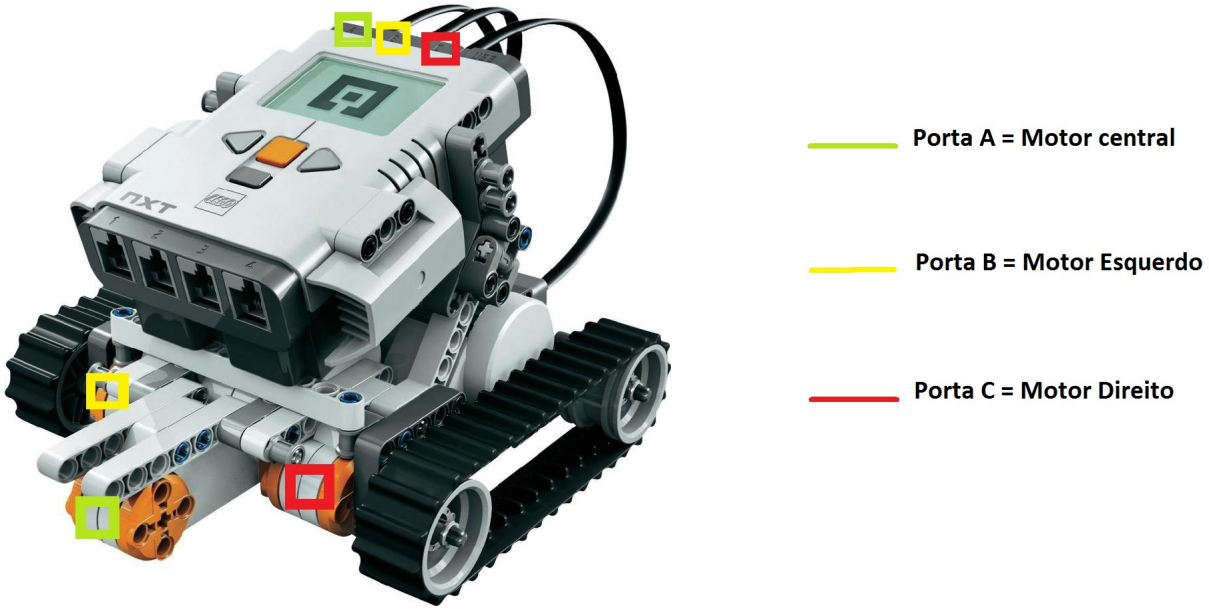

Figura 7 – Montagem do robô Bauen - Lego Mindstorms NXT

4.4 Tapete de missões

O tapete de missões escolhido para esse trabalho é o *Nature's Fury* (LEAGUE, 2013), utilizado como desafio do torneio *First Lego League* de 2013, que tem como temática uma tsunami que está devastando as cidades. Esse tapete tem caráter educacional, ensinando quanto aos lugares seguros e não seguros para permanecer quando uma tsunami está em atividade. Os lugares seguros durante uma tsunami, demarcados no tapete, são bonificados com uma pontuação extra, caso o robô fique nesse lugar. Já os lugares inseguros também são demarcados no tapete. Porém, esses lugares representam penalidades, com perda de pontuação, caso o robô passe pelos mesmos. Existem missões de salvar os animais de estimação, salvar pessoas, levar água e mantimentos para os lugares seguros. O tapete *Nature's Fury* foi escolhido para ser objeto de estudo deste projeto, uma vez que é utilizado nas aulas de Princípios de Robótica Educacional, sendo assim, mais acessível do que outros tapetes. Na Figura 8, é possível visualizar o tapete, bem como os obstáculos que possui.

4.5 Descrição das missões

As missões contidas no tapete *Nature's Fury* (LEAGUE, 2013) são descritas a seguir:

Figura 8 – Tapete de missões *Nature's Fury* - Lego Mindstorms NXT

- **missão caminhão de fornecimento:** consiste em levar o caminhão até a área amarela do mapa. Pontuação: 20 pontos;
- **missão sinal de evacuação:** consiste em levantar a placa com o sinal de evacuação. Pontuação: 30 pontos;
- **missão avião de carga:** consiste em fazer o avião chegar à área azul ou amarela do tapete. Pontuação: 20 pontos para a área amarela, 30 pontos para a área azul;
- **missão galho da árvore:** consiste em retirar o galho da árvore sem deixá-lo encostar nos fios de alta tensão. Pontuação: 30 pontos;
- **missão tsunami:** consiste em fazer com que as três ondas que estão suspensas toquem o tapete. Pontuação: 20 pontos;
- **missão ambulância:** consiste em levar a ambulância até a área amarela do tapete. Pontuação: 25 pontos;
- **missão pista limpa:** consiste em retirar qualquer objeto da pista de pouso. Pontuação: 30 pontos;
- **missão realocação de construção:** consiste em retirar todas as peças cinzas encontradas na construção da área verde do tapete. Pontuação: 20 pontos;

- **missão teste da base de isolamento:** consiste em encostar na base de isolamento e derrubar apenas um dos prédios. Pontuação: 30 pontos;

- **missão construção:** consiste em empilhar os segmentos na área rosa do tapete. Pontuação: 5 pontos para cada segmento empilhado;

- **missão obstáculo:** consiste no robô passar por cima de obstáculos chegando às áreas coloridas do tapete. Nesse caso, para cada cor do tapete, existe uma pontuação. Pontuação: 10 pontos para a área azul, 16 pontos para a área verde, 23 pontos para a área roxa e 31 pontos para a área vermelha;

- **missão elevar a casa:** consiste no robô abaixar a alavanca que eleva a casa. Pontuação: 25 pontos;

- **missão progresso:** consiste em rodar um círculo de cores. Pontuação: 2 pontos para cada cor que o ponteiro passar;

- **missão família:** consiste em juntar as pessoas que estão no tapete em uma única área colorida. Pontuação: 33 para duas pessoas juntas, ou 66 para 3 pessoas juntas;

- **missão água:** consiste em juntar ao menos uma pessoa com uma garrafa de água na mesma região. Pontuação: 15 pontos para cada pessoa+água;

- **missão segurança:** consiste em levar pelo menos uma pessoa à região vermelha ou amarela do tapete. Pontuação: 12 pontos para cada pessoa na área amarela ou 18 pontos para cada pessoa na área vermelha;

- **missão *pets*:** consiste em juntar ao menos um animal à uma pessoa numa área colorida. Pontuação: 15 pontos para cada pet+pessoa;

- **missão equipamentos e suprimentos:** consiste em levar pelo menos um item, exceto a água, para a região vermelha ou amarela do tapete. Pontuação: 3 pontos para cada item na área amarela ou 4 pontos para cada item na área vermelha, e

- **missão lugar seguro:** consiste no robô estar na região vermelha no final do jogo. Pontuação: 25 pontos.

4.6 Configuração do Ambiente

O Sistema Operacional escolhido para ser utilizado por esse trabalho foi o Windows, mais especificamente a versão 10. Foi utilizada a IDE Eclipse para trabalhar com a parte Java do projeto, e o editor NotePad++ para trabalhar com a parte Prolog do

Projeto. O compilador Prolog utilizado foi o SwiProlog, e o executável pode ser baixado no próprio site[4].

Para trabalhar com o Java, foi necessário instalar o Java 7, obrigatoriamente na versão de 32 bits, pois o Lego não aceita a versão 64bits. Da mesma forma, o Eclipse teve que ser instalado na versão 32 bits. O NXT tem um *firmware* específico para se trabalhar com o Java, chamado leJOS. A instalação do leJOS deve ser feita no computador, no eclipse e no robô. Tais configurações são descritas a seguir.

Instalação do leJOS no Windows

O executável pode ser baixado no próprio site do leJOS[5], na versão 0.9.1 . Ao final da instalação do executável, é iniciado o NXJ Flash, aplicativo responsável por instalar o *firmware* do leJOS no robô, como mostra a Figura 9.

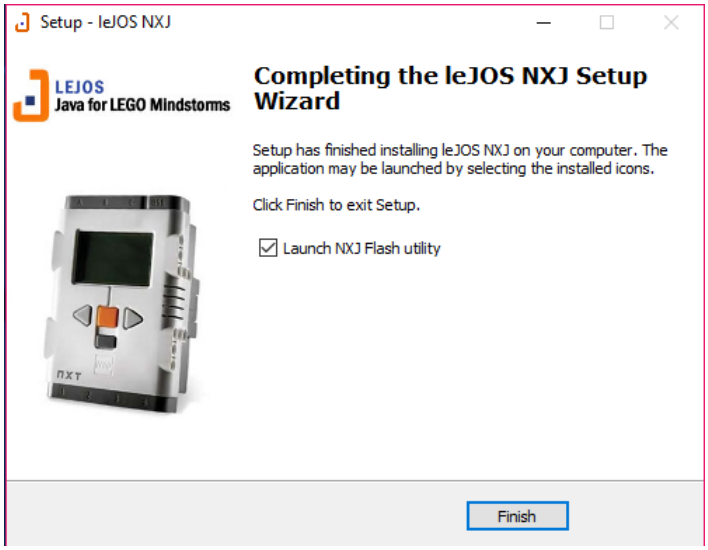

Figura 9 – Instalação do leJOS no Windows

Instalação do leJOS no Robô

O aplicativo NXJ Flash, instalado pelo executável do leJOS, é o responsável pela instalação do *firmware* no robô. Para iniciar a instalação, é necessário que o robô esteja conectado ao computador via cabo USB, ou seja, a instalação não é feita via *bluetooth*.

[4] http://www.swi-prolog.org/download/stable
[5] http://www.lejos.org/

A Figura 10 mostra a aplicação NXJ Flash.

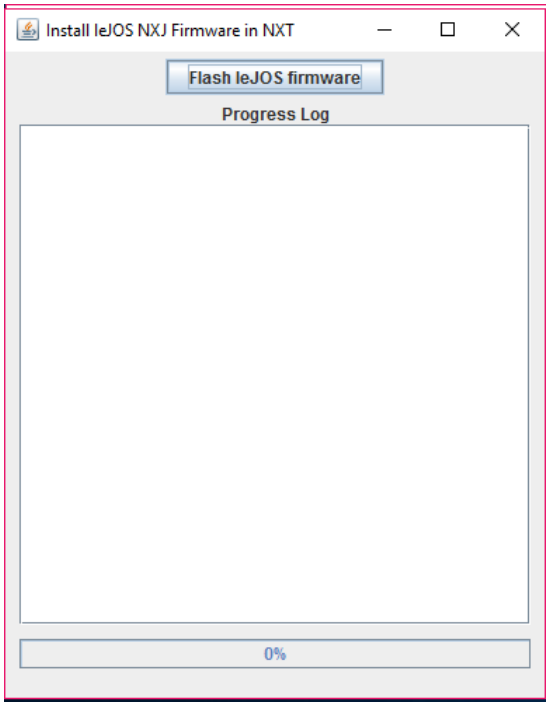

Figura 10 – Aplicativo NXJ Flash

Ao clicar no botão *Flash leJOS Firmware*, mostrado na Figura 10, é iniciada a instalação do *firmware* no robô.

Instalação do leJOS no Eclipse

O leJOS para o Eclipse é um *plugin*, que é instalado clicando na aba *Help*, no Eclipse, na opção *Install New Software*. A Figura11 mostra a janela para instalação do *plugin*.

Ao clicar em *add*, um pop-up é aberto com as opções *name* e *url*, que devem ser preenchidas como mostrado na Figura 12. Nesse caso, basta clicar em OK, selecionar o *plugin*, e clicar em *Finish*.

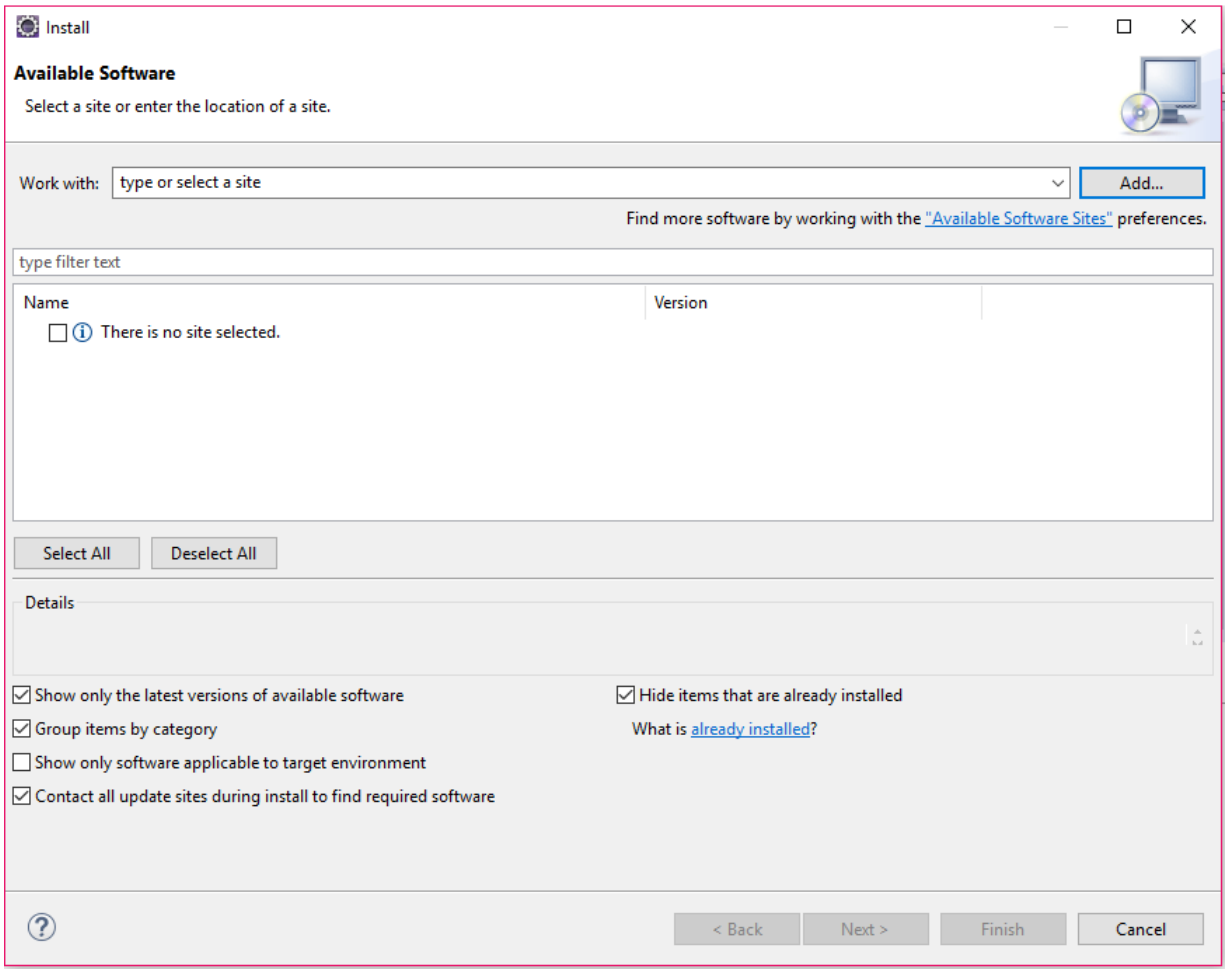

Figura 11 – Instalando Plugin no Eclipse

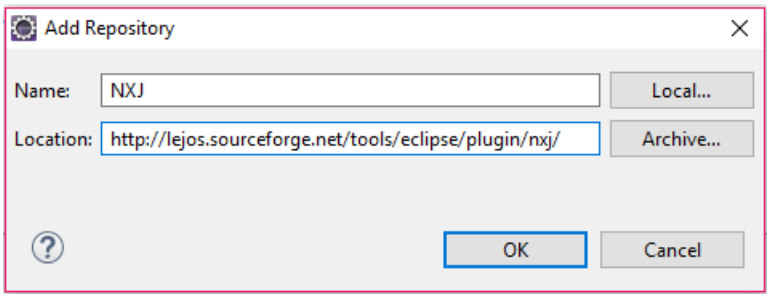

Figura 12 – Dados do Plugin do leJOS

Para finalizar, é necessário adicionar o caminho, a partir do qual o leJOS foi instalado, na variável NXJ_HOME, encontrada na aba *Window*, opção *Preferences*, clicando em leJOS NXJ, conforme mostrado na Figura 13.

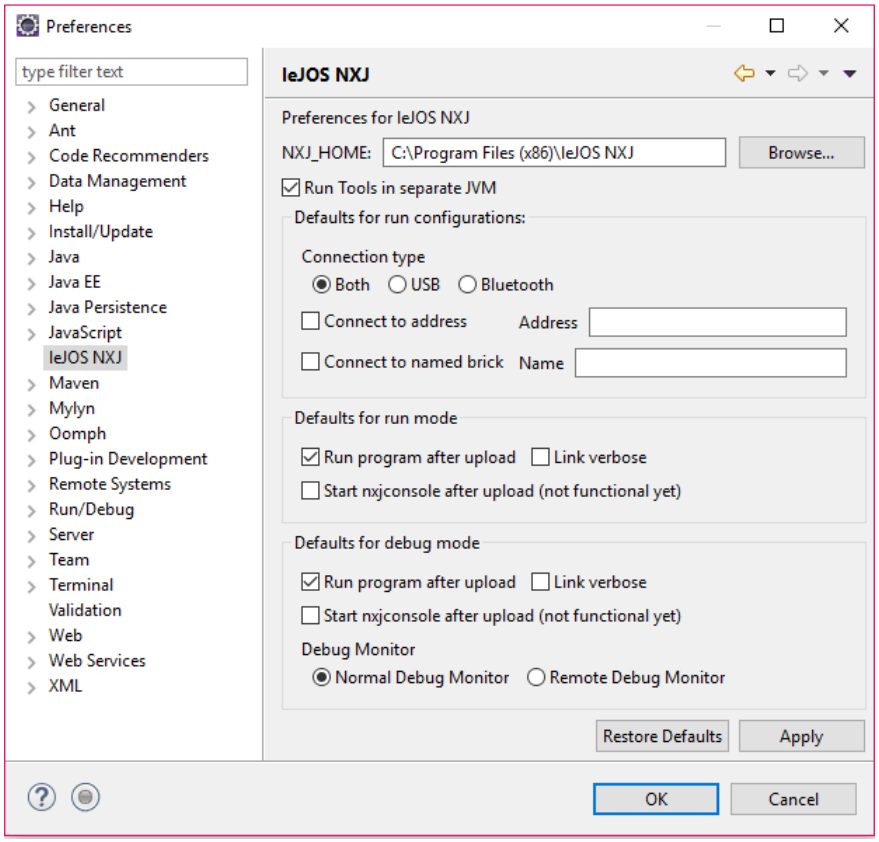

Figura 13 – Caminho do NXJ HOME

4.7 Arquitetura do Prolego

A arquitetura do robô costuma ser em camadas, e o Prolego age na camada de decisão, como mostrado na Seção 4.2. Porém, a arquitetura interna do Prolego é componentizada, onde a *Main* controla todos os outros de forma hierárquica, como mostra o diagrama da Figura 14. Para melhor visualização, esse diagrama foi elaborado em ramos, sendo esses descritos nas Figuras 15, 16, 17, 18.

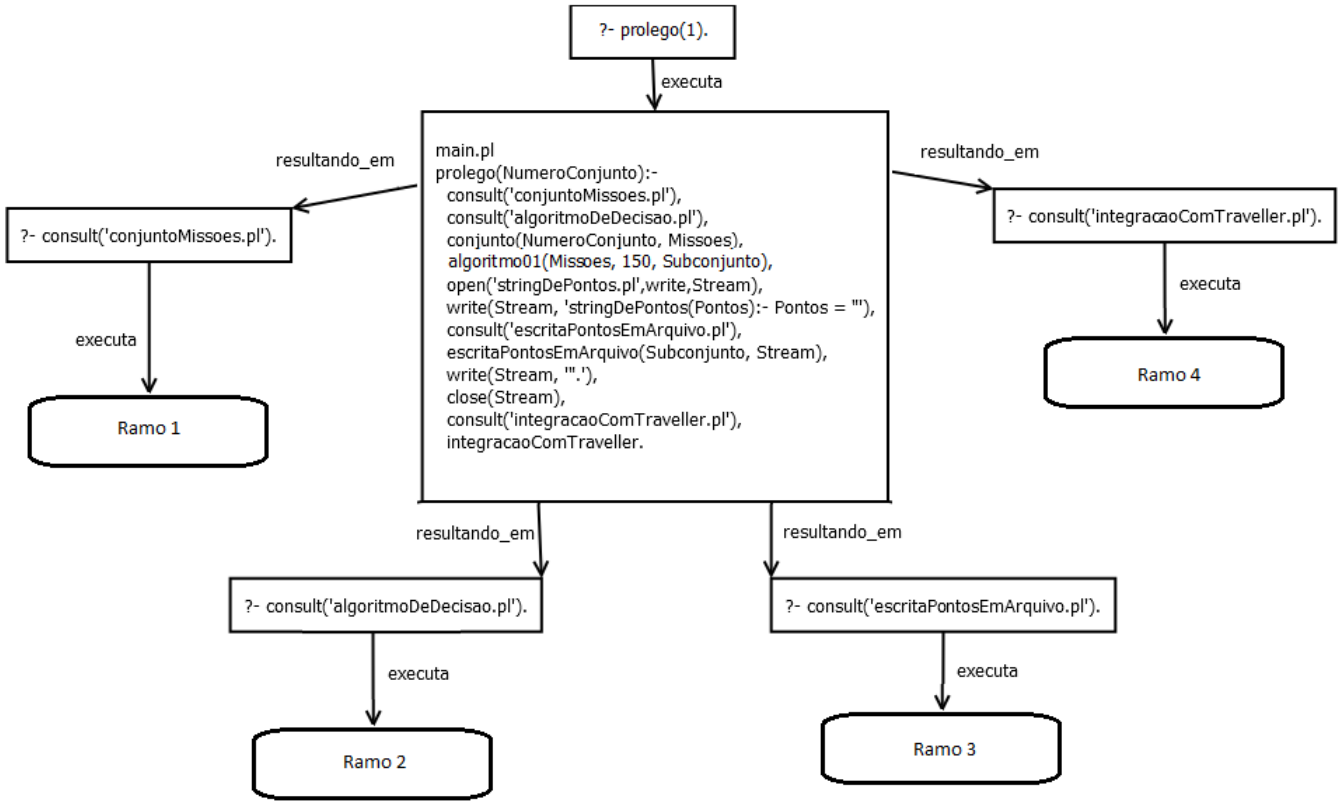

Figura 14 – Diagrama de Fluxo do Prolego

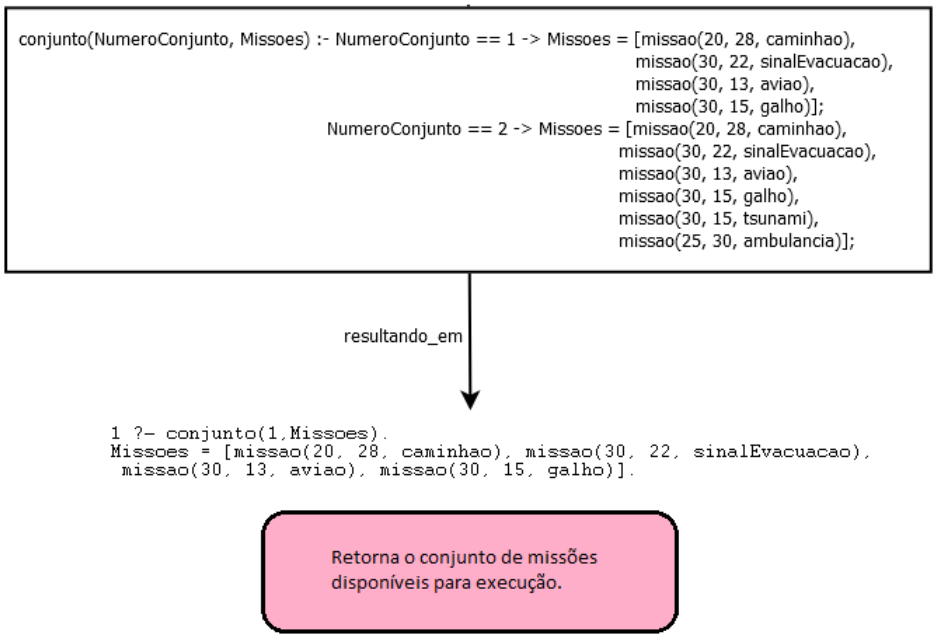

Figura 15 – Diagrama de Fluxo do Prolego - Ramo 1

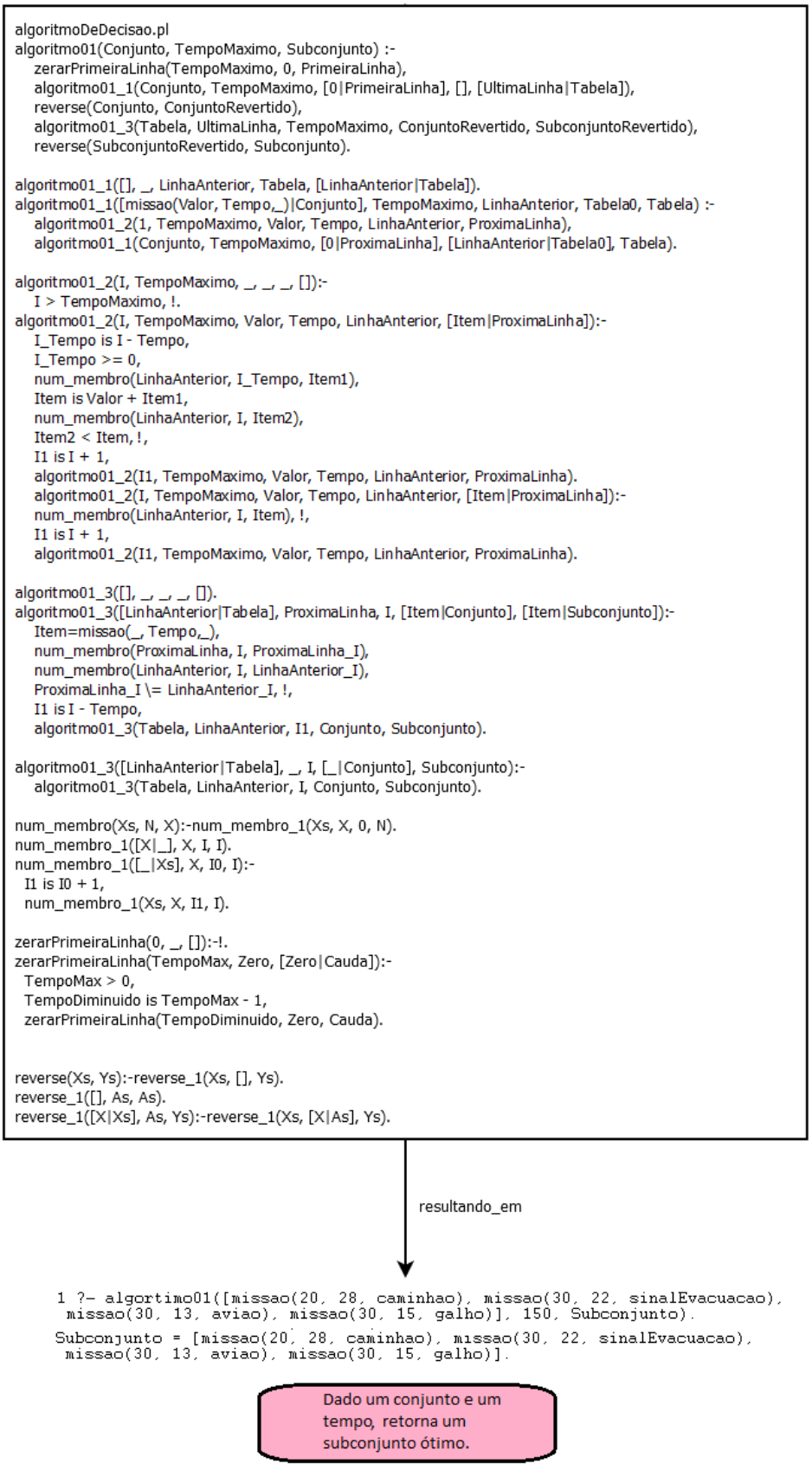

Figura 16 – Diagrama de Fluxo do Prolego - Ramo 2

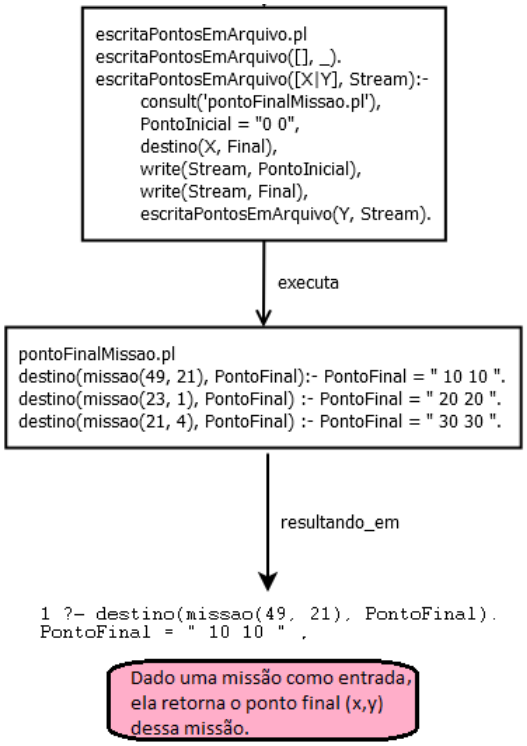

Figura 17 – Diagrama de Fluxo do Prolego - Ramo 3

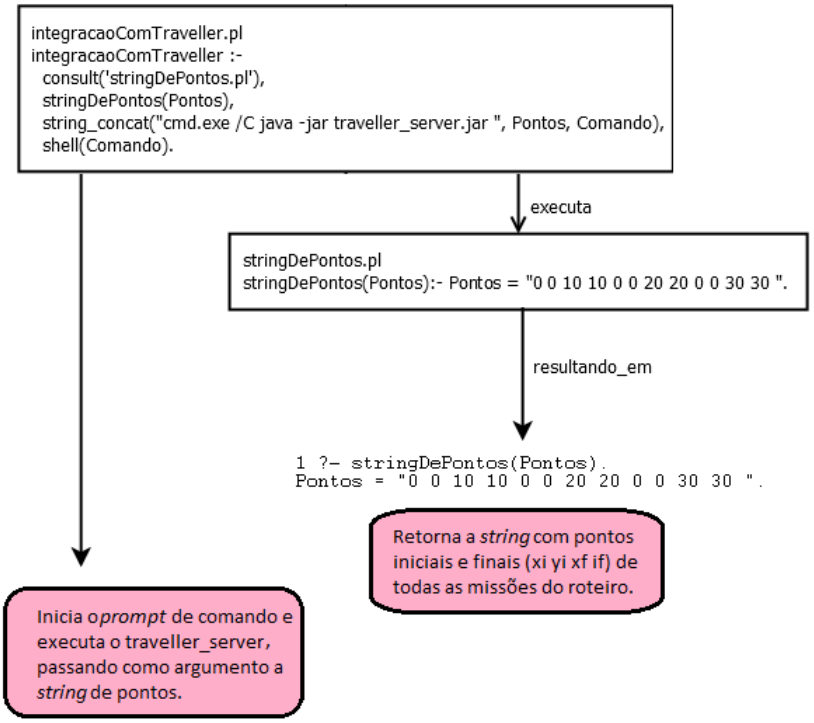

Figura 18 – Diagrama de Fluxo do Prolego - Ramo 4

As subseções a seguir explicam o comportamento de cada componente que compõe o Prolego.

4.7.1 *main*

O componente *main* tem apenas uma regra chamada prolego. Esta regra é responsável por pegar o conjunto de missões da classe conjuntoDeMissoes.pl, de acordo com o número do conjunto passado; acionar o componente *algoritmoDeDecisão.pl* passando, o conjunto juntamente com o tempo restante, e receber como retorno um subconjunto ótimo.

Em posse desse subconjunto, a *main* cria um arquivo para escrita, chamado *stringDePontos.pl*, no qual a regra *escritaDePontos* escreve os pontos iniciais e finais de todas as missões. A main, por fim, chama a regra *integracaoComTraveller*, que é responsável por integrar o Prolego com o *framework* Traveller.

4.7.2 *conjuntoMissoes*

Este componente contém as missões disponíveis para execução. Os conjuntos de missões estão associados a um número. Dependendo do número passado pelo usuário, para a regra *prolego* do componente *main*, tal componente retorna um conjunto diferente de missões. O componente *conjuntoMissoes* pode conter 'n' conjuntos, a serem cadastrados pelo usuário.

4.7.3 *algoritmoDeDecisao*

Este componente é uma adaptação do conhecido problema da mochila. O componente recebe um conjunto de missões, e através de programação dinâmica avalia a pontuação e o tempo de cada missão, retornando então um subconjunto ótimo de missões a serem executadas.

4.7.4 *pontoFinalMissao*

Este componente contém o ponto final de todas as missões. Quando passado a missão, ee retorna o ponto final da mesma. Não é possível obter todos os pontos finais do conjunto de uma só vez, pois é possível consultar apenas uma missão por vez.

4.7.5 *escritaPontosEmArquivo*

Este componente recebe o subconjunto do componente *main*, e de forma recursiva ele consulta o componente *pontoFinalMissão* para obter o ponto final da missão avaliada. Para cada missão este componente escreve no arquivo os pontos iniciais e finais da mesma.

4.7.6 *stringDePontos*

Este é um componente criado em tempo de execução, com os pontos iniciais e finais do subconjunto ótimo. Ele retorna uma *string* com todos os pontos.

4.7.7 *integracaoComTraveller*

Este componente é responsável por inicializar o módulo Server do Traveller, passando a *string* de pontos como argumento. Para inicializar o Traveller, este componente utiliza o método *shell* que possibilita executar qualquer linha de comando. Desta forma, para rodar o Traveller, primeiramente, inicializou-se o *prompt* de comando, e depois executou o *.jar* da aplicação.

4.8 Integração com o Traveller

A integração do Prolego com o Traveller foi efetuada em duas partes: a primeira foi a configuração do Traveller para trabalhar com o mapa e o robô definidos por este trabalho e a segunda foi a criação de métodos no Prolego para adaptar a saída à entrada do Traveller. Essas partes são descritar a seguir.

A Figura 19 mostra o mapa que foi colocado no Traveller, em forma de uma matriz booleana, onde "1"representa que existe um obstáculo naquela célula, e "0"representa uma célula livre. Cada célula da matriz equivale a 3 centímetros do mapa real. Todo o mapeamento das células foi feito de forma manual.

A Figura 20 mostra o método *attendRequest* da classe *Main*. É nesse método que é especificado o tamanho do robô utilizado, no caso desse trabalho 17cm (o tamanho do robô é o maior tamanho entre a largura e o comprimento), o tamanho da célula do mapa, nesse caso é 3cm.

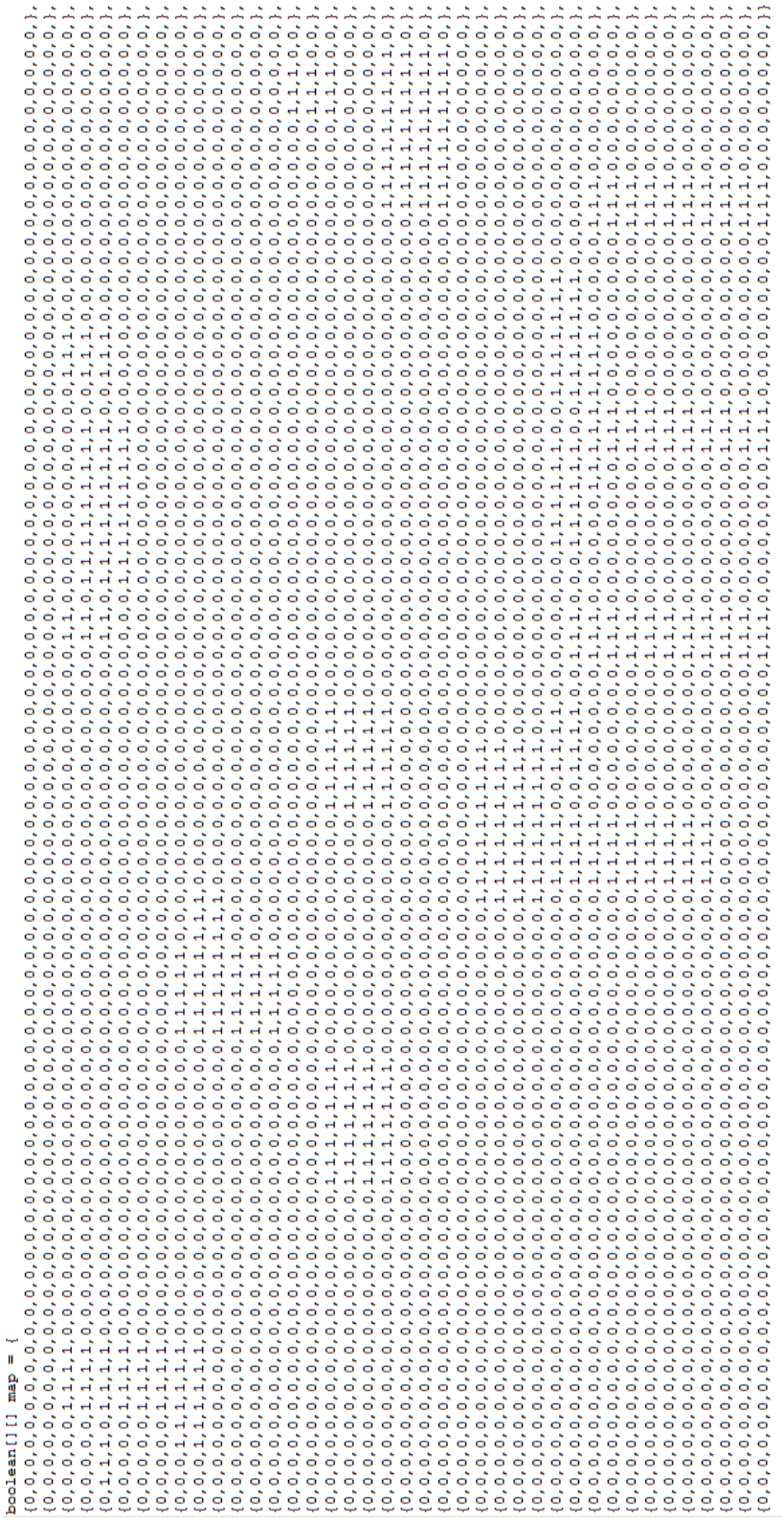

Figura 19 – Método *createMap* do classe *Main* - Traveller Server

Com essa configuração, o Traveller já tinha os dados necessários para criar o melhor caminho para o robô alcançar o ponto de destino, porém, ainda foi necessário configurar a conexão *bluetooth* entre o Traveller e o NXT.

```
private static void attendRequest(BluetoothCommunicator communicator) throws IOException{

    String path_planner = "VisibilityGraph";
    String best_path = "Djikstra";
    boolean[][] map =createMap();
    boolean free_value = false;
    float carWidth = 17;
    float cellWidth = 3;
    boolean expand_obtacles = true;
```

Figura 20 – Método *attendRequest* da classe *Main* - Traveller Server

Para realizar a conexão, foi utilizado o próprio aplicativo de *bluetooth* do computador, com o qual foi procurado e localizado o NXT. Ao emparelhar, foi utilizado o PIN 1234, que é o PIN do NXT. Este PIN pode ser modificado nas configurações do NXT por qualquer outro PIN de 4 dígitos.

Realizado o emparelhamento, o Traveller já era capaz de enviar o código do Traveller Robot para o NXT, via *bluetooth*.

A segunda parte da integração foi a criação dos métodos *escritaDePontosEmArquivos*, *stringDePontos* e *integracaoComTraveller*, no Prolego. Os dois primeiros métodos são responsáveis pela adaptação do retorno do *algoritmoDeDecisao* para uma entrada válida para o Traveller, ou seja, uma *string* com pontos iniciais e finais. O último método é responsável por inicializar o Traveller Server e passar os pontos para ele.

Com essas duas etapas concluídas foram inicializados os cenários de teste.

4.9 Considerações parciais

Através de uma obsrvação sistemática a máquina de raciocínio foi evoluída desde a sua primeira versão, principalmente em relação a modularização do código, de forma que cada módulo trate de um determinado escopo, obedecendo assim às boas práticas de programação.

A máquina de raciocínio foi evoluída também de acordo com as necessidades, pois a criação novos módulos foi indispensável para integrar o Prolego e o Traveller.

5 Cenários de Teste

Os cenários de teste tem como função apoiar no processo de observação sistemática, para que através deles a maquina de raciocínio seja avaliada e tal pergunta seja respondida: a máquina de racicínio está satisfatória?

Os cenários de teste consistem em 9 conjuntos de missões distintas, criados no componente *conjuntoMissoes.pl*, a fim de testar a saída do robô. Foram analisados os retornos do *algoritmoDeDecisao*, que são as missões que devem ser efetuadas, e da *stringDePontos*, que contém os pontos de cada missão escolhida.

O gabarito para cada cenário de teste foi criado da seguinte forma: para o *algoritmoDeDecisao*, foi feita uma análise manual da programação dinâmica, obtendo assim um resultado; e para a *stringDePontos,* foram verificados se todos os pontos iniciais das missões eram x=0 e y=0, e os pontos finais condiziam com os especificados no arquivo *pontoFinalMissao.pl*.

As Tabelas de 1 a 9 mostram os cenários de teste, especificando a pontuação e o tempo de cada um.

Tabela 1 – Cenário de Teste 1

Missão	Pontuação	Tempo
Caminhão	20	28
Sinal de Evacuação	30	22
Avião	30	13
Galho	30	15
Tsunami	30	15
Ambulância	25	30
Realocação	20	37
Base de Isolamento	30	26
Obstáculos	31	40
Elevar a Casa	25	15
Família	33	26
Segurança	36	26

Tabela 2 – Cenário de Teste 2

Missão	Pontuação	Tempo
Caminhão	20	28
Sinal de Evacuação	30	22
Avião	30	13
Galho	30	15
Tsunami	30	15
Ambulância	25	30
Realocação	20	37

Tabela 3 – Cenário de Teste 3

Missão	Pontuação	Tempo
Realocação	20	37
Base de Isolamento	30	26
Obstáculos	31	40
Elevar a Casa	25	15
Família	33	26
Segurança	36	26

Tabela 4 – Cenário de Teste 4

Missão	Pontuação	Tempo
Avião	30	13
Galho	30	15
Tsunami	30	15
Ambulância	25	30
Realocação	20	37
Base de Isolamento	30	26
Família	33	26

Tabela 5 – Cenário de Teste 5

Missão	Pontuação	Tempo
Avião	30	13
Tsunami	30	15
Realocação	20	37
Base de Isolamento	30	26
Obstáculos	31	40
Elevar a Casa	25	15
Família	33	26
Segurança	36	26

Tabela 6 – Cenário de Teste 6

Missão	Pontuação	Tempo
Caminhão	20	28
Sinal de Evacuação	30	22
Avião	30	13
Galho	30	15
Obstáculos	31	40
Elevar a Casa	25	15
Família	33	26
Segurança	36	26

Tabela 7 – Cenário de Teste 7

Missão	Pontuação	Tempo
Tsunami	30	15
Ambulância	25	30
Realocação	20	37
Base de Isolamento	30	26
Família	33	26
Segurança	36	26

Tabela 8 – Cenário de Teste 8

Missão	Pontuação	Tempo
Sinal de Evacuação	30	22
Tsunami	30	15
Ambulância	25	30
Realocação	20	37
Família	33	26
Segurança	36	26

Tabela 9 – Cenário de Teste 9

Missão	Pontuação	Tempo
Sinal de Evacuação	30	22
Avião	30	13
Galho	30	15
Tsunami	30	15
Ambulância	25	30
Realocação	20	37
Base de Isolamento	30	26
Obstáculos	31	40
Elevar a Casa	25	15
Família	33	26
Segurança	36	26

Ao iniciar o primeiro cenário de teste, foram verificados os retornos acima especificados, sendo que todos estavam de acordo com o gabarito. Porém, o robô não executou o trajeto. Depois de avaliar o código do Traveller, foi percebido que a conexão *bluetooth* estava sendo feita apenas em uma direção, que seria do computador para o NXT, tanto que era possível passar o código para o mesmo via *bluetooth*. Entretanto, o NXT não estava conectando ao computador como deveria, e por esse motivo, aparecia a seguinte mensagem no visor: *"No known devices"*.

O Traveller, para passar o caminho ao NXT, tentava criar uma conexão *bluetooth* utilizando o método *getKnownDevice*, que procura na lista de devices do NXT o nome do computador, e realiza a conexão. Como a lista do NXT estava vazia, a conexão não era efetuada.

A primeira tentativa para contornar o problema foi a utilização do método *inquire*, que pesquisa a lista de dispositivos encontrados, pelo *search* do robô, e com base nessa lista, ele adiciona o computador à lista de dispositivos conhecidos do NXT, utilizando o método *addDevice*. Entretanto, nessa abordagem, o retorno do *inquire* sempre vinha nulo, mesmo o NXT achando o computador quando realizada a busca manualmente.

Analisando melhor a comunicação entre os módulos do Traveller, foi averiguado que o *server* ficava sempre à espera da conexão que vinha do módulo *robot*. Essa dinâmica foi invertida, pois como quem inicializa o Traveller é o Prolego, logo não existe a necessidade dessa espera.

Para fazer o *robot* aguardar a conexão vinda do *server*, foi utilizado o método *waitForConnection*. Com o robô em espera da conexão, o *server* abre uma conexão com ele através do método *NXTInfo*, que recebe como argumentos o nome do robô, no caso GTX, e o endereço MAC dele.

Ao realizar essas modificações, a conexão foi efetuada com sucesso e o robô começou a executar os caminho dos pontos passados. No entanto, foi averiguado um problema de odometria ao executar o caminho, mais especificadamente em relação ao ângulo nas curvas que o robô executa, aparentemente o robô está virando mais do que deveria. Como esse problema não está no escopo deste trabalho, o mesmo foi desconsiderado.

Todos os cenários de teste seguintes obtiveram sucesso, e os resultados são apresendos nas seções a seguir.

5.1 Cenário de teste 1

A Tabela 10 representa o gabarito, realizado de forma manual, do cenário de teste 1. Nesta Tabela, as missões representadas em vermelho são as que, ao fim, não fizeram parte do subconjunto ótimo.

A Figura 21 apresenta o resultado do cenário de teste 1 aplicado no *algoritmoDeDecisao*. Como pode-se observar, tanto de forma manual quanto utilizando o algoritmo, foram selecionadas as mesmas missões para o subconjunto.

Tabela 10 – Gabarito do Cenário de Teste 1

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Avião(13)	0	30	30	30	30	30	30	30	30	30	30
{30} Galho(15)	0	30	60	60	60	60	60	60	60	60	60
{30} Tsunami(15)	0	30	60	90	90	90	90	90	90	90	90
{25} Elevar a Casa(15)	0	30	60	90	115	115	115	115	115	115	115
{30} Sinal de Evacuação (22)	0	30	60	90	115	120	120	120	120	120	120
{30} Base de Isolamento (26)	0	30	60	90	115	120	120	150	150	150	150
{33} Família (26)	0	30	60	90	115	120	120	150	183	183	183
{26} Segurança (26)	0	30	60	90	115	120	120	150	186	186	219
{20} Caminhão (28)	0	30	60	90	115	120	120	150	186	186	219
{25} Ambulância (30)	0	30	60	90	115	120	120	150	186	186	219
{20} Realocação (37)	0	30	60	90	115	120	120	150	186	186	219
{31} Obstáculos (40)	0	30	60	90	115	120	120	150	186	186	219

```
Subconjunto = [missao(30, 22, sinalEvacuacao), missao(30, 13, aviao), missao(30, 15, galho), missao(30, 15, tsunami),
 missao(30, 26, baseIsolamento), missao(33, 26, familia), missao(36, 26, seguranca)].
```

Figura 21 – Retorno do Cenário de Teste 1 aplicado no *algoritmoDeDecisao*

Quanto aos pontos X e Y das missões, a Tabela 11 mostra quais os pontos finais das missões dos subconjunto ótimo, lembrando que o ponto inicial é x = 0 e y = 0. A Figura 22 mostra o retorno do componente *stringDePontos* criado a partir componente *escritaPontosEmArquivo*, que recebeu como entrada o subconjunto ótimo do Cenário de Teste 1.

Comparando os pontos finais da Tabela e da Figura, é possível perceber que o componente gerou a saída corretamente.

Tabela 11 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 1

	Ponto Final	
Missão	X	Y
Sinal de Evacuação	65	18
Avião	31	2
Galho	14	33
Tsunami	29	14
Base de Isolamento	53	30
Família	75	12
Segurança	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 65 18 0 0 31 2 0 0 14 33 0 0 29 14 0 0 53 30 0 0 75 12 0 0 75 12 ".
```

Figura 22 – *String* De Pontos gerada para o Cenário de Teste 1

5.2 Cenário de teste 2

O segundo Cenário de Teste também obteve sucesso, pois o resultado gerado pela Tabela 12 é idêntico ao gerado pelo componente *algoritmoDeDecisão* mostrado na Figura 23.

Tabela 12 – Gabarito do Cenário de Teste 2

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Avião(13)	0	30	30	30	30	30	30	30	30	30	30
{30} Galho(15)	0	30	60	60	60	60	60	60	60	60	60
{30} Tsunami(15)	0	30	60	90	90	90	90	90	90	90	90
{30} Sinal de Evacuação (22)	0	30	60	90	90	120	120	120	120	120	120
{20} Caminhão (28)	0	30	60	90	90	120	120	140	140	140	140
{25} Ambulância (30)	0	30	60	90	90	120	120	145	145	165	165
{20} Realocação (37)	0	30	60	90	90	120	120	145	145	165	165

```
Subconjunto = [missao(20, 28, caminhao), missao(30, 22, sinalEvacuacao),
 missao(30, 13, aviao), missao(30, 15, galho), missao(30, 15, tsunami),
missao(25, 30, ambulancia)].
```

Figura 23 – Retorno do Cenário de Teste 2 aplicado no *algoritmoDeDecisao*

Os pontos X e Y da Tabela 13 conferem com os pontos da saída gerada pelo componente *stringDePontos* mostrado na Figura 24.

Tabela 13 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 2

	Ponto Final	
Missão	X	Y
Caminhão	12	21
Sinal de Evacuação	65	18
Avião	31	2
Galho	14	33
Tsunami	29	14
Ambulância	33	22

5.3 Cenário de teste 3

O terceiro Cenário de Teste obteve sucesso, assim como os seus anteriores. A Tabela 14 confere com a saída mostrada na Figura 25.

```
stringDePontos(Pontos):- Pontos = "0 0 12 21 0 0 65 18 0 0 31 2 0 0 14 33 0 0 29 14 0 0 33 22 ".
```

Figura 24 – *String* De Pontos gerada para o Cenário de Teste 2

Tabela 14 – Gabarito do Cenário de Teste 3

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{25} Elevar a Casa(15)	0	25	25	25	25	25	25	25	25	25	25
{30} Base de Isolamento (26)	0	25	30	55	55	55	55	55	55	55	55
{33} Família (26)	0	25	33	58	63	88	88	88	88	88	88
{26} Segurança (26)	0	25	36	61	69	94	99	124	124	124	124
{20} Realocação (37)	0	25	36	61	69	94	99	124	124	144	144
{31} Obstáculos (40)	0	25	36	61	69	94	99	124	130	155	155

```
Subconjunto = [missao(30, 26, baseIsolamento), missao(31, 40, obstaculos), missao(25, 15, elevarCasa),
 missao(33, 26, familia), missao(36, 26, seguranca)].
```

Figura 25 – Retorno do Cenário de Teste 3 aplicado no *algoritmoDeDecisao*

Os pontos X e Y mostrados na Tabela 15 e na Figura 26 também conferem.

Tabela 15 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 3

	Ponto Final	
Missão	X	Y
Base de Isolamento	53	30
Obstáculos	75	8
Elevar a Casa	34	26
Família	75	12
Segurança	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 53 30 0 0 75 8 0 0 34 26 0 0 75 12 0 0 75 12 ".
```

Figura 26 – *String* De Pontos gerada para o Cenário de Teste 3

5.4 Cenário de teste 4

O quarto Cenário de Teste, não diferente dos demais, obteve sucesso tanto no subconjunto ótimo, mostrado pela Tabela 16 e a Figura 27, quanto nos pontos, mostrados na Tabela 17 e na Figura 28.

Tabela 16 – Gabarito do Cenário de Teste 4

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Avião(13)	0	30	30	30	30	30	30	30	30	30	30
{30} Galho(15)	0	30	60	60	60	60	60	60	60	60	60
{30} Tsunami(15)	0	30	60	90	90	90	90	90	90	90	90
{30} Base de Isolamento (26)	0	30	60	90	90	120	120	120	120	120	120
{33} Família (26)	0	30	60	90	93	123	153	153	153	153	153
{25} Ambulância (30)	0	30	60	90	93	123	153	153	153	178	178
{20} Realocação (37)	0	30	60	90	93	123	153	153	153	178	178

```
Subconjunto = [missao(30, 13, aviao), missao(30, 15, galho), missao(30, 15, tsunami),
missao(25, 30, ambulancia), missao(30, 26, baseIsolamento), missao(33, 26, familia)].
```

Figura 27 – Retorno do Cenário de Teste 4 aplicado no *algoritmoDeDecisao*

Tabela 17 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 4

	Ponto Final	
Missão	X	Y
Avião	31	2
Galho	14	33
Tsunami	29	14
Ambulância	33	22
Base de Isolamento	53	30
Família	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 31 2 0 0 14 33 0 0 29 14 0 0 33 22 0 0 62 31 0 0 53 30 ".
```

Figura 28 – *String* De Pontos gerada para o Cenário de Teste 4

5.5 Cenário de teste 5

O quinto Cenário de Teste, conforme a Tabela 18 e a Figura 29, também obteve sucesso ao gerar o subconjunto ótimo. Quanto aos pontos iniciais e finais, a Tabela 19 e a Figura 30 demostram que o cenário obteve sucesso.

Tabela 18 – Gabarito do Cenário de Teste 5

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Avião(13)	0	30	30	30	30	30	30	30	30	30	30
{30} Tsunami(15)	0	30	60	60	60	60	60	60	60	60	60
{25} Elevar a Casa(15)	0	30	60	85	85	85	85	85	85	85	85
{30} Base de Isolamento (26)	0	30	60	85	90	115	115	115	115	115	115
{33} Família (26)	0	30	60	85	93	118	123	148	148	148	148
{26} Segurança (26)	0	30	60	85	96	121	129	154	159	184	184
{20} Realocação (37)	0	30	60	85	96	121	129	154	159	184	184
{31} Obstáculos (40)	0	30	60	85	96	121	129	154	160	185	190

```
Subconjunto = [missao(30, 13, aviao), missao(30, 15, tsunami), missao(30, 26, baseIsolamento),
 missao(31, 40, obstaculos), missao(33, 26, familia), missao(36, 26, seguranca)].
```

Figura 29 – Retorno do Cenário de Teste 5 aplicado no *algoritmoDeDecisao*

Tabela 19 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 5

	Ponto Final	
Missão	X	Y
Avião	31	2
Tsunami	29	14
Base de Isolamento	53	30
Obstáculos	75	8
Família	75	12
Segurança	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 31 2 0 0 29 14 0 0 53 30 0 0 75 8 0 0 75 12 0 0 75 12 ".
```

Figura 30 – *String* De Pontos gerada para o Cenário de Teste 5

5.6 Cenário de teste 6

O sexto Cenário de Teste obteve sucesso nos dois pontos analisados: tanto na geração do subconjunto ótimo quanto na geração da *string* de pontos. A Tabela 20 e a Figura 31 demonstram o sucesso no primeiro ponto analisado, enquanto a Tabela 21 e a Figura 32 demonstram o sucesso do segundo ponto.

Tabela 20 – Gabarito do Cenário de Teste 6

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Avião(13)	0	30	30	30	30	30	30	30	30	30	30
{30} Galho(15)	0	30	60	60	60	60	60	60	60	60	60
{25} Elevar a Casa(15)	0	30	60	85	85	85	85	85	85	85	85
{30} Sinal de Evacuação (22)	0	30	60	85	90	115	115	115	115	115	115
{33} Família (26)	0	30	60	85	93	118	123	148	148	148	148
{26} Segurança (26)	0	30	60	85	96	121	129	154	184	184	184
{20} Caminhão (28)	0	30	60	85	96	121	129	154	184	184	204
{31} Obstáculos (40)	0	30	60	85	96	121	129	154	184	185	184

```
Subconjunto = [missao(20, 28, caminhao), missao(30, 22, sinalEvacuacao), missao(30, 13, aviao), missao(30, 15, galho),
 missao(25, 15, elevarCasa), missao(33, 26, familia), missao(36, 26, seguranca)].
```

Figura 31 – Retorno do Cenário de Teste 6 aplicado no *algoritmoDeDecisao*

Tabela 21 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 6

	Ponto Final	
Missão	X	Y
Caminhão	12	21
Sinal de Evacuação	65	18
Avião	31	2
Galho	14	33
Elevar a Casa	34	26
Família	75	12
Segurança	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 12 21 0 0 65 18 0 0 31 2 0 0 14 33 0 0 34 26 0 0 75 12 0 0 75 12 ".
```

Figura 32 – *String* De Pontos gerada para o Cenário de Teste 6

5.7 Cenário de teste 7

O sétimo Cenário de Uso obteve sucesso na geração do subconjunto ótimo, como demostra a Tabela 22 e a Figura 33. Obteve sucesso também na geração da *string* de pontos, conforme Tabela 23 e Figura 34.

Tabela 22 – Gabarito do Cenário de Teste 7

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Tsunami(15)	0	30	30	30	30	30	30	30	30	30	30
{30} Base de Isolamento (26)	0	30	30	60	60	60	60	60	60	60	60
{33} Família (26)	0	30	33	63	63	93	93	93	93	93	93
{26} Segurança (26)	0	30	36	66	69	99	99	129	129	129	129
{25} Ambulância (30)	0	30	36	66	69	99	99	129	129	154	154
{20} Realocação (37)	0	30	36	66	69	99	99	129	129	154	154

```
Subconjunto = [missao(30, 15, tsunami), missao(25, 30, ambulancia), missao(30, 26, baseIsolamento),
 missao(33, 26, familia), missao(36, 26, seguranca)].
```

Figura 33 – Retorno do Cenário de Teste 7 aplicado no *algoritmoDeDecisao*

Tabela 23 – Pontos das Missões do Subconjun|to Ótimo do Cenário de Teste 7

	Ponto Final	
Missão	X	Y
Tsunami	29	14
Ambulância	33	22
Base de Isolamento	53	30
Família	75	12
Segurança	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 29 14 0 0 33 22 0 0 53 30 0 0 75 12 0 0 75 12 ".
```

Figura 34 – *String* De Pontos gerada para o Cenário de Teste 7

5.8 Cenário de teste 8

Segundo a Tabela 24 e a Figura 35, o quinto Cenário de Teste obteve sucesso no quesito geração do subconjunto ótimo.

Tabela 24 – Gabarito do Cenário de Teste 8

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Tsunami(15)	0	30	30	30	30	30	30	30	30	30	30
{30} Sinal de Evacuação (22)	0	30	30	60	60	60	60	60	60	60	60
{33} Família (26)	0	30	33	63	63	93	93	93	93	93	93
{26} Segurança (26)	0	30	36	66	66	96	129	129	129	129	129
{25} Ambulância (30)	0	30	36	66	66	96	129	129	154	154	154
{20} Realocação (37)	0	30	36	66	66	96	129	129	154	154	154

```
Subconjunto = [missao(30, 22, sinalEvacuacao), missao(30, 15, tsunami), missao(25, 30, ambulancia),
missao(33, 26, familia), missao(36, 26, seguranca)].
```

Figura 35 – Retorno do Cenário de Teste 8 aplicado no *algoritmoDeDecisao*

Conforme consta na Tabela 25 e na Figura 36, o oitavo Cenário de Teste também obteve sucesso no quesito geração da string de pontos.

Tabela 25 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 8

	Ponto Final	
Missão	X	Y
Sinal de Evacuação	65	18
Tsunami	29	14
Ambulância	33	22
Família	75	12
Segurança	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 29 14 0 0 33 22 0 0 62 31 0 0 75 12 0 0 75 12 ".
```

Figura 36 – *String* De Pontos gerada para o Cenário de Teste 8

5.9 Cenário de teste 9

O último cenário de teste também obteve sucesso nos dois quesitos: geração do subconjunto ótimo de geração da *string* de pontos. A Tabela 26 e a Figura 37 comprovam o sucesso do nono Cenário de Teste no primeiro quesito. E, comprovando o sucesso no segundo quesito, estão os resultados apresentados na Tabela 27 e na Figura 38.

Tabela 26 – Gabarito do Cenário de Teste 9

	Tempo Restante										
{Valor} Missão(Tempo)	**0**	**15**	**30**	**45**	**60**	**75**	**90**	**105**	**120**	**135**	**150**
0	0	0	0	0	0	0	0	0	0	0	0
{30} Galho(15)	0	30	30	30	30	30	30	30	30	30	30
{30} Tsunami(15)	0	30	60	60	60	60	60	60	60	60	60
{25} Elevar a Casa(15)	0	30	60	80	85	85	85	85	85	85	85
{30} Sinal de Evacuação (22)	0	30	60	85	90	115	115	115	115	115	115
{30} Base de Isolamento (26)	0	30	60	85	90	115	120	145	145	145	145
{33} Família (26)	0	30	60	85	93	118	123	148	178	178	178
{26} Segurança (26)	0	30	60	85	96	121	129	154	159	184	214
{20} Caminhão (28)	0	30	60	85	96	121	129	154	159	184	214
{25} Ambulância (30)	0	30	60	85	96	121	129	154	159	184	214
{20} Realocação (37)	0	30	60	85	96	121	129	154	159	184	214
{31} Obstáculos (40)	0	30	60	85	96	121	129	154	159	184	214

```
Subconjunto = [missao(30, 22, sinalEvacuacao), missao(30, 13, aviao), missao(30, 15, galho), missao(30, 15, tsunami),
missao(30, 26, baseIsolamento), missao(33, 26, familia), missao(36, 26, seguranca)].
```

Figura 37 – Retorno do Cenário de Teste 9 aplicado no *algoritmoDeDecisao*

Tabela 27 – Pontos das Missões do Subconjunto Ótimo do Cenário de Teste 9

	Ponto Final	
Missão	X	Y
Sinal de Evacuação	65	18
Avião	31	2
Galho	14	33
Tsunami	29	14
Base de Isolamento	53	30
Família	75	12
Segurança	75	12

```
stringDePontos(Pontos):- Pontos = "0 0 65 18 0 0 31 2 0 0 14 33 0 0 29 14 0 0 53 30 0 0 75 12 0 0 75 12 ".
```

Figura 38 – *String* De Pontos gerada para o Cenário de Teste 9

5.10 Considerações parciais

Os cenários de teste foram criados propositalmente com números variáveis de missões, para que fosse comprovada a eficiência da máquina com diversos tamanhos de entradas. Cada cenário de teste seguiu a regra de que o somatório dos tempos de execução das missões excedesse o tempo estipulado de 150 segundos, pois assim ao menos uma

missão seria excluída pela máquina de raciocínio. Desta forma, todos os cenários de uso obtiveram sucesso nos quesitos analisados.

6 Conclusão

Nesse trabalho foram investigados alguns tipos de algoritmos de otimização, como o algoritmo guloso e a programação dinâmica, e como estes algoritmos poderiam ser adaptados ao problema em questão. Foi estudado também o paradigma lógico, para avaliar a viabilidade de utilizá-lo na solução. A decisão de utilizar o kit Lego Mindstorms NXT foi baseada no estudo sobre o funcionamento do mesmo e na comparação com outras soluções robóticas, como exposto no Capítulo 3.

A condução da pesquisa foi baseado na metodologia definida no Capítulo 1, orientada às boas práticas da Engenharia de Software.

Com base nessa pesquisa bibliográfica, foi desenvolvida uma máquina de raciocínio que, considerando a programação dinâmica como algoritmo para tomada de decisões, é capaz de selecionar missões, compondo um roteiro, de forma que a pontuação é maximizada em relação ao tempo disponível.

A metodologia utilizada definiu que para a análise dos resultados seria utilizado o procedimento técnico de cenário de uso e técnica de coleta de dados da observação sistemática. Todos os resultados e descrição dos cenários de teste estão contidos no Capítulo 5. Como pôde-se observar, todos os cenários de teste obtiveram sucesso nos quesitos analisados, e a máquina de racicínio se mostrou satisfátoria.

Utilizando a observação sistemática em cada cenário de teste, a máquina de raciocínio foi evoluída de forma ficar o mais modularizado possível, obedecendo às boas práticas de programação. A máquina de racicínio também foi evoluída para integrar com o *framework* Traveller.

6.1 Sugestão de trabalhos futuros

Tendo em vista a continuidade do trabalho, com foco na melhoria da máquina de racicínio, algumas sugestões são indicadas aos interessados:

1. evoluir a máquina para que ela tenha conhecimento da localização atual, desta forma as missões poderão iniciar em qualquer local do tapete, e não só no ponto (0,0);

2. com o conhecimento da localização atual e levando em consideração que a máquina de raciocínio leva milisegundos para devolver a resposta, evoluir o algoritmo para que ele retorne a próxima missão a ser executada, e não um roteiro de missões, assim a máquina de raciocínio é utilizada em tempo de execução das missões, e não antes de começar a competição, e

3. acrescentar à maquina uma função que, em tempo de execução, retire a missão que foi executada anteriormente, atualizando assim, a base de conhecimento.

Referências

ANDRÉ, M. E. D. A. d. *Estudo de caso em pesquisa e avaliação educacional*. [S.l.]: Líber Livro, 2008. Citado na página 27.

BARRIENTOS, A. et al. Fundamentos de robótica. *Ed. Mc Graw Hill,*, 1997. Citado 2 vezes nas páginas 31 e 32.

BENITTI, F. B. V. et al. Experimentação com robótica educativa no ensino médio: ambiente, atividades e resultados. In: *Anais do XXIX Congresso da Sociedade Brasileira de Computação, Bento Gonçalves/RS*. [S.l.: s.n.], 2009. p. 1811–1820. Citado na página 36.

CASTALONGA, J. P.; LEMOS, J. M. S. et al. Cobertura e empacotamento por circuitos através de um elemento em matróides. Universidade Federal de Pernambuco, 2007. Citado na página 43.

CLOCKSIN, W.; MELLISH, C. S. *Programming in PROLOG*. [S.l.]: Springer Science & Business Media, 2003. Citado na página 45.

CORMEN, T. H. *Introduction to algorithms*. [S.l.]: MIT press, 2009. Citado na página 44.

CURCIO, C. P. d. C. Instituto de tecnologia para o desenvolvimento (lactec) instituto de engenharia do paraná (iep) programa de pós-graduação em desenvolvimento de tecnologia (prodetec). 2008. Citado na página 32.

GRASSI, V. Arquitetura híbrida para robôs móveis baseada em funções de navegação com interação humana. *Escola Politécnica, USP.*, 2006. Citado na página 40.

GROOVER, M. P. *Robótica: tecnologia e programação*. [S.l.]: McGraw-Hill, 1989. Citado na página 32.

KAMEN, D.; KRISTIANSEN, K. K. First lego league. 2010. Citado na página 25.

LEAGUE, F. L. *Nature's Fury Challenge*. 2013. Citado na página 58.

LOBATO, F. S. *Otimização Multi-objetivo para o Projeto de Sistemas de Engenharia*. Tese (Doutorado), 2008. Citado 2 vezes nas páginas 47 e 48.

MAISONNETTE, R. A utilização dos recursos informatizados a partir de uma relação inventiva com a máquina: a robótica educativa. *PROINFO-Programa Nacional de Informática na Educação, Curitiba-PR*, 2002. Citado 2 vezes nas páginas 35 e 36.

MORESI, E. et al. Metodologia da pesquisa. *Universidade Católica de Brasília*, 2003. Citado na página 27.

MURPHY, R. *Introduction to AI robotics*. [S.l.]: MIT press, 2000. Citado na página 23.

NOF, S. Y. *Handbook of industrial robotics*. [S.l.]: John Wiley & Sons, 1999. v. 1. Citado na página 31.

PIO, J. L. de S.; CASTRO, T. H. C. de; JÚNIOR, A. N. de C. A robótica móvel como instrumento de apoio à aprendizagem de computação. In: *Anais do Simpósio Brasileiro de Informática na Educação*. [S.l.: s.n.], 2006. v. 1, n. 1, p. 497–506. Citado na página 32.

PRAKASAM, A.; SAVARIMUTHU, N. Metaheuristic algorithms and polynomial turing reductions: A case study based on ant colony optimization. *Procedia Computer Science*, Elsevier, v. 46, p. 388–395, 2015. Citado na página 48.

REDEL, R.; HOUNSELL, M. d. S. Implementação de simuladores de robôs com o uso da tecnologia de realidade virtual. In: *IV Congresso Brasileiro de Computação, Itajaí–SC. IV CBCOMP*. [S.l.: s.n.], 2004. v. 1, p. 398–401. Citado na página 32.

RINCON, R. L. Framework de definição de trajetória para robôs móveis. 2014. Citado 5 vezes nas páginas 13, 38, 51, 56 e 57.

ROBOMIND Academy - Learn To Code. <http://www.robomindacademy.com>. Citado na página 49.

ROCHA, A. Algoritmos gulosos: definições e aplicacções. Campinas, SP, 2004. Citado 2 vezes nas páginas 42 e 43.

RODRIGUES, T. G. *Sobre os Fundamentos da Programação Lógica Paraconsistente*. Tese (Doutorado) — Universidade Estadual de Campinas, 2010. Citado na página 45.

RUSSEL, S.; NORVIG, P. Inteligência artificial. *Editora Campus*, 2004. Citado na página 24.

SANTOS, C. C. d. Robótica na construção: uma aplicação prática. Universidade de Coimbra, 2002. Citado 2 vezes nas páginas 31 e 32.

SANTOS, K. *SISTEMA DE NAVEGAÇÃO AUTÔNOMA PARA ROBÔS MÓVEIS BASEADO EM ARQUITETURA HÍBRIDA: TEORIA E APLICAÇÃO*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DE ITAJUBÁ, 2009. Citado 2 vezes nas páginas 40 e 41.

SILVA, A. F. D. *RoboEduc: Uma metodologia de aprendizado com Robótica Educacional*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE, 2009. Citado 2 vezes nas páginas 23 e 24.

SILVA, A. F. de et al. Utilização da teoria de vygotsky em robótica educativa, em ‘. In: IX CONGRESO IBEROAMERICANO DE INFORMATICA EDUCATIVA RIBIE. [S.l.], 2008. Citado na página 32.

TUCKER, A.; NOONAN, R. *Linguagens de Programação: Princípios e Paradigmas*. [S.l.]: Grupo A Educação, 2009. Citado na página 25.

VIEIRA, F. C. *Controle dinâmico de robôs móveis com acionamento diferencial*. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, 2005. Citado 4 vezes nas páginas 13, 38, 56 e 57.

WOLF, D. F. et al. Robótica móvel inteligente: Da simulação às aplicações no mundo real. In: *Mini-Curso: Jornada de Atualização em Informática (JAI), Congresso da SBC*. [S.l.: s.n.], 2009. Citado 5 vezes nas páginas 13, 38, 39, 40 e 41.

ZILLI, S. d. R. et al. A robótica educacional no ensino fundamental: perspectivas e prática. Florianópolis, SC, 2004. Citado 3 vezes nas páginas 33, 35 e 36.

Apêndices

APÊNDICE A – Primeiro Apêndice

O código fonte do projeto encontra-se no seguinte endereço de repositório:

https://github.com/Prolego/prolego

Colaborando com a filosofia *OpenSource*, a máquina de raciocínio pode ser utilizada por terceiros, sendo apenas requisitada a menção desse Trabalho como fonte de pesquisa. Para isso, segue a referência ao Trabalho: Ramalho, Carolina Barros. "Máquina de Raciocínio Lógico para Tomada de Decisões Estratégicas em Robótica Educacional". Trabalho de Conclusão de Curso. Universidade de Brasília (UnB), Faculdade do Gama (FGA), Curso de Engenharia de Software, Milene Serrano (profa. orientadora) e Maurício Serrano (prof. coorientador). Janeiro de 2015 a Julho de 2016.