Vítor Almeida Subhi

Majestic: Uma Ampliação de uma Linguagem de Programação para Robótica Educacional

Jataí-Goiás

2019

Vítor Almeida Subhi

Majestic: Uma Ampliação de uma Linguagem de Programação para Robótica Educacional

Monografia apresentada como requisito para a obtenção do diploma de Bacharel em Ciência da Computação pela Universidade Federal de Goiás - Regional Jataí

Universidade Federal de Goiás - Regional Jataí - UFG-REJ

Instituto de Ciências Exatas e Tecnológicas (ICET)

Bacharelado em Ciências da Computação

Orientador: Prof. Dr. Thiago Borges de Oliveira

Jataí-Goiás

2019

Vítor Almeida Subhi
Majestic: Uma Ampliação de uma Linguagem de Programação para Robótica Educacional/ Vítor Almeida Subhi. – Jataí-Goiás, 2019-
57 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Thiago Borges de Oliveira

Monografia (Graduação) – Universidade Federal de Goiás - Regional Jataí - UFG-REJ
Instituto de Ciências Exatas e Tecnológicas (ICET)
Bacharelado em Ciências da Computação, 2019.

1. Robótica Educacional. 2. Linguagem de Programação.

Vítor Almeida Subhi

Majestic: Uma Ampliação de uma Linguagem de Programação para Robótica Educacional

Monografia apresentada como requisito para a obtenção do diploma de Bacharel em Ciência da Computação pela Universidade Federal de Goiás - Regional Jataí

Trabalho aprovado. Jataí-Goiás, data da defesa:

Prof. Dr. Thiago Borges de Oliveira
Orientador

Prof. Dr. José Euripedes Ferreira de Jesus Filho
Avaliador

Prof^{a}. Dr^{a}. Ana Paula Freitas Vilela Boaventura
Avaliador

Jataí-Goiás
2019

Este trabalho é dedicado a Deus, o maior orientador da minha vida, sem Ele nada seria possível. E também ao meu avô Viltomar Almeida Cunha (in memorian), que não pôde estar ao meu lado neste momento tão importante, mas que sempre torceu muito por mim.

AGRADECIMENTOS

Agradeço a Deus por ter me dado forças e me mantido na trilha certa durante este projeto de pesquisa com saúde e forças para chegar até o final. Sou grato à minha família e amigos pelo apoio que sempre me deram durante toda a minha vida. Ao meu professor orientador por sempre estar presente para indicar a direção correta que o trabalho deveria tomar. Também agradeço a todos os meus colegas de curso, pela oportunidade do convívio e pela cooperação mútua durante esses anos.

"O sucesso é ir de fracasso em fracasso sem perder entusiasmo..
(Winston Churchill)

RESUMO

Uma das técnicas de motivação ao aprendizado que tem se destacado devido à facilidade de transmitir ideias complexas é o uso de robótica educacional. Porém, as principais linguagens de programação utilizadas na Robótica Educacional são de propósito geral dificultando o ensino-aprendizado devido à complexidade de domínio de sintaxe das mesmas. Uma alternativa a essas linguagens são as Linguagens Específicas de Domínio que são projetadas com a finalidade de auxiliar no processo de resolução de problemas em um domínio. Com base nisso, este trabalho ampliou a expressividade de uma linguagem de programação experimental, implementada sobre o *framework* Robcmp, desenvolvido no contexto do projeto Especificação e Construção de Protótipos Funcionais de Kits Robóticos de Baixo Custo para uso em Processos de Ensino Aprendizagem, na Regional Jataí da UFG. A linguagem resultante deste projeto chama-se *Majestic* e acrescentou três recursos principais sobre os já disponibilizados pelo *framework* Robcmp: funções com parâmetros, vetores e matrizes estáticas, ambos alocados em pilha.

Palavras-chaves: *Robótica Educacional; Linguagem de Programação.*

ABSTRACT

One of the motivating techniques in learning process that has contrast due to the ease way of conveying complex ideas is the use of educational robotics. But the main programming languages used in Educational Robotics are of general purpose programming languages making it difficult in the process of teaching-learning due to the complexity of their syntax mastery. An alternative to these languages are Domain Specific Languages, which are designed to assist in the process of solving problems in a certain domain. Based on this, this paper extend the expressivity of an experimental programming language, implemented over the framework Robcmp, developed in the context of a project Expecification and Construction of Functional Lower-Cost Robotics Kits Prototypes in the teaching-learning process, in Regional Jataí - UFG. The resultant language of this project calls Majestic and extended three main resources already disponibilized over framework Robcmp: Functions with parameters, static vectors and matrixes, both allocated in stack.

Key-words: *Educational Robotics; Programming Language.*

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de código desenvolvido utilizando Scratch 22
Figura 2 – Exemplo de código desenvolvido utilizando DB4K 22
Figura 3 – Exemplo de código desenvolvido utilizando NXT-G 22
Figura 4 – Fases do Front-End de um Compilador 24
Figura 5 – Exemplo de Ambiguidade 25

LISTA DE TABELAS

Tabela 1 – Trabalhos e Seus Respectivos Métodos 34
Tabela 2 – Símbolos dos Operadores suportados pela linguagem ML 36
Tabela 3 – Tipos de Dados suportados pela função 38
Tabela 4 – Comparação de tempo de compilação entre as Linguagem ML e C . . . 43
Tabela 5 – Comparação de tamanho de arquivo entre as Linguagem ML e C . . . 44

LISTA DE ABREVIATURAS E SIGLAS

DC	*Direct Current*
GUI	*Graphical User Interface*
VPL	*Visual Programming Language*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
OOL	*Object-Oriented Language*
ML	*Majestic Language*

SUMÁRIO

1 Introdução 14
INTRODUÇÃO 14
1.1 Motivação 14
1.2 Objetivo do Trabalho 17
1.3 Contribuição do Trabalho 17
1.4 Organização da Monografia 17
2 Referencial Teórico 18
2.1 Introdução 18
2.2 Robótica Educacional 18
2.2.1 Programação na Robótica Educacional 20
2.2.2 Programação Textual na Robótica Educacional 20
2.2.3 Programação Gráfica na Robótica Educacional 21
2.3 Compilador 23
2.3.1 Front-End 23
2.3.2 Analisador Léxico 24
2.3.3 Analisador Sintático 24
2.3.4 Analisador Semântico 25
2.3.5 Gerador de Código Intermediário 26
2.4 Hardwares utilizados na Robótica Educacional 26
2.4.1 Microcontroladores 26
2.4.2 Sensores e Atuadores 27
2.5 Considerações Finais 27
3 Trabalhos relacionados 29
3.1 Processo de busca 29
3.2 Trabalhos analisados 30
3.2.1 A Plataforma Universal Urbi para Robótica 30
3.2.2 *Rust*, uma linguagem de programação 31
3.2.3 Ensino com robôs em escolas secundárias: alguns problemas pedagógicos novos e não tão novos 32
3.3 Resumo Comparativo 33
4 Linguagem de Programação 35
4.1 Linguagem de Programação ML 35
4.1.1 Operadores 35
4.1.2 Tipos de dados em variáveis 35
4.1.3 Estruturas de Decisão 36

4.1.4 Vetores e Matrizes . . . 37
4.1.5 Funções . . . 38
4.2 Estruturas de repetição . . . 39
4.3 Portas de Entrada/Saída . . . 39
4.4 Compilador . . . 39
4.5 Construção do Compilador . . . 40
4.6 Tecnologias de Apoio . . . 41
5 Avaliação e Testes . . . 42
5.0.1 Experimentos com Tempo de Compilação . . . 42
5.0.2 Experimentos com Tamanho de Arquivo . . . 43
5.0.3 Considerações Finais . . . 44
6 Conclusões e Trabalhos Futuros . . . 45
6.1 Conclusões . . . 45
6.2 Trabalhos futuros . . . 46

Referências . . . 47

Apêndices 51
APÊNDICE A Algoritmos em *Majestic Language* . . . 52

1 INTRODUÇÃO

1.1 Motivação

Estudos apontam que conforme há o avanço da tecnologia, as formas tradicionais de educação em escolas devem ser acompanhadas de algo inovador, complexo e tecnológico (FILIPPOV et al., 2017). De acordo com Stoppa (2012) uma maneira de motivar o aprendizado de teorias tradicionais, como matemática, química, física, dentre outras, consideradas "difíceis" por parte dos estudantes é através da utilização de novas ferramentas tecnológicas. Uma das técnicas que vêm sendo empregadas com sucesso em países desenvolvidos, desde o início dos anos 2000, é o uso de metodologias de ensino baseadas em Robótica Educacional (BARKER, 2012). Robótica Educacional é um termo utilizado para associar a utilização de kits robóticos como um instrumento de apoio ao processo de ensino-aprendizado, inserindo conceitos considerados complexos e relacionados a um determinado assunto desde o início da formação dos estudantes. Entre os conceitos geralmente empregados estão a eletrônica, programação de computadores, mecânica aplicada e robótica básica (MAJOR; KYRIACOU; BRERETON, 2012). A Robótica Educacional também é considerada como uma ferramenta única que apresenta um ambiente de aprendizado atrativo, prático e divertido que estimula o interesse e a curiosidade dos estudantes (EGUCHI, 2010).

Apesar da grande maioria dos trabalhos disponíveis sobre robótica educacional terem foco no ensino de programação ou da própria robótica, outras disciplinas podem também ser beneficiadas pela sua utilização (NETO et al., 2015). Nessa perspectiva, a robótica tem se tornado uma ferramenta educacional popular em áreas da ciência e tecnologia em escolas de ensino primário, secundário e em diversas áreas da engenharia em universidades (LÓPEZ-RODRÍGUEZ; CUESTA, 2016), pelo fato de oferecer uma maneira lúdica e tangível para as crianças envolverem-se com conceitos de tecnologia e engenharia durante sua formação (SULLIVAN; ELKIN; BERS, 2015). Dessa forma, o interesse em robótica tem crescido nos últimos anos, por oferecer diversos benefícios importantes para a educação em todos os níveis (JOHNSON, 2003), sendo bastante utilizada como reforço nas aulas de matemática, física e química no ensino básico (CASADO et al., 2016; MARTINS; TEIXEIRA; AUGUSTO, 2016; MIRANDA; SAMPAIO; BORGES, 2011; RICARDO et al., 2018). Essa utilização é apoiada por (STOPPA, 2012) citando que "a manipulação de kits de robótica se apresenta como um atrativo recurso didático adicional", e é apontada como um recurso que além de ter um impacto potencial no aprendizado dos estudantes em diversos assuntos, também contribui no desenvolvimento pessoal, incluindo cognitivo, meta-cognitivo e social, como em: pesquisa, pensamento criativo, tomada de decisões, solução de problemas, comunicação e trabalho em equipe, habilidades necessárias no

ambiente de trabalho do século 21 (EGUCHI, 2010; FILIPPOV et al., 2017; DAGDILELIS; SARTATZEMI; KAGANI, 2005).

Contudo, as principais linguagens de programação utilizadas na Robótica Educacional são compostas por linguagens de programação de propósito geral se tornando difíceis de serem utilizadas devido à complexidade de domínio de sintaxe das mesmas, contribuindo no aumento de dificuldade de compreensão, por parte das crianças (RESNICK et al., 2009). Uma alternativa a essas linguagens são as Linguagens Específicas de Domínio (*Domain-Specific Languages* - DSL) que são linguagens de programação projetadas com o propósito de auxiliar no processo de resolução de problemas em um domínio (FREITAS et al., 2017; FOWLER, 2010; DEURSEN; KLINT; VISSER, 2000; DEURSEN; KLINT, 2002). Diferentemente de linguagens de propósito geral que são empregadas para um grande conjunto de tarefas em domínios variados, as DSLs são restritas a um conjunto limitado de tarefas (FREITAS et al., 2017). Sendo assim, as linguagens específicas de domínio em sua grande maioria mais expressivas na solução de problemas em seu domínio específico (MERNIK; HEERING; SLOANE, 2005; DEURSEN; KLINT; VISSER, 2000) se tornando mais fáceis de compreender, escrever e modificar (HUDAK, 1998).

Segundo Deursen, Klint e Visser (2000), as linguagens de propósito específicos:

- São concisas, auto-documentadas em grande medida, e podem ser reutilizadas para diferentes propósitos.
- Aumentam a produtividade, a confiabilidade, a manutenibilidade e a portabilidade.
- Incorporam conhecimento de domínio e, assim, permitem a conservação e reutilização desse conhecimento.
- Permitem validação e otimização no nível do domínio.
- Melhoram a testabilidade.

Porém as linguagens de propósito específico possuem as seguintes desvantagens:

- Altos custos para projetar, implementar e manter a linguagem.
- Os custos de educação pelos usuários são elevados.
- A disponibilidade limitada de linguagens de propósito específico..
- A dificuldade de encontrar o escopo adequado para a mesma.
- A dificuldade de equilibrar entre construções de linguagem de programação de especificidade de domínio e de propósito geral.
- A potencial perda de eficiência quando comparada com linguagens de propósito geral

Conforme os robôs são desenvolvidos eles são acompanhados com sua própria interface de linguagem de programação, na maioria das vezes forçando os pesquisadores a reaprender o que já sabem, devido a seus diferentes paradigmas e sintaxes. Além disso, algumas dessas interfaces são notavelmente difíceis de se dominar (BAILLIE, 2005), fazendo com que programadores fiquem relutantes a aprender uma nova linguagem de programação, justificando a necessidade de uma linguagem de programação inovativa e padronizada (BAILLIE et al., 2008).

De acordo com Bruyninckx (2002), a robótica na indústria é caracterizada pela incompatibilidade entre diferentes linguagens de programação para robótica de diferentes fornecedores. Além disso, essas linguagens são limitadas, e não muito inovadoras quando comparadas entre si. Dessa forma, esses robôs quando adquiridos de diferentes fornecedores possuem a sua integração difícil, como também dificulta a transição para um fornecedor diferente. Além da dificuldade da inserção de empresas voltadas para o desenvolvimento de robôs voltados para um nicho específico, devido ao alto custo do desenvolvimento de um controlador para o mesmo.

De acordo com Lopes (2017) esses fatores tornam viável a utilização de uma linguagem de programação de propósito específico na Robótica Educacional. Gomes et al. (2016) defende que se é necessário uma linguagem de programação padronizada para facilitar a utilização de protótipos robóticos na Robótica Educacional. Para López-Rodríguez e Cuesta (2016) as linguagens de programação utilizadas na Robótica Educacional, deve ser além de expressiva, possuir poucas limitações. Dessa forma torna-se viável a elaboração de uma linguagem de programação específica e homogênea voltada para a Robótica Educacional, que seja expressiva e de complexidade intermediária para os usuários.

No curso de Ciências da Computação da Universidade Federal de Jataí foi desenvolvida uma linguagem de programação para auxiliar no ensino de Compiladores. A disciplina de Compiladores é um componente curricular considerado extremamente teórico e complexo devido ao uso de diversos conceitos aprendidos durante o curso como programação de computadores, linguagens formais e autômatos, paradigmas de linguagem de programação, estruturas de dados, entre outros. No entanto, é apresentado um desempenho por parte dos alunos em relação ao aprendizado, que conseguem ao final da disciplina, apresentar protótipos de novas linguagens de programação e compiladores funcionais.

O resultado da disciplina foi compilado em um framework escrito em C e C++, chamado Robcmp[1] que implementa uma linguagem básica experimental para programação de microcontroladores AVR, presentes principalmente nas conhecidas placas de prototipação Arduino(ARDUINO, 2015), frequentemente empregadas em robótica. O Objetivo deste trabalho é expandir a expressividade desta linguagem básica, acrescentando recursos importantes para a programação na Robótica Educacional, conforme detalhado a seguir.

[1] github.com/thborges/robcmp

1.2 Objetivo do Trabalho

Este trabalho teve como objetivo,

- Acréscimo de funções e seus parâmetros à linguagem de programação;
- Acréscimo de vetores e matrizes à linguagem de programação;
- Construção de programas exemplos para a linguagem de programação resultante;
- Avaliação da adição proposta a linguagem de programação.

1.3 Contribuição do Trabalho

A contribuição deste trabalho está na criação de um modelo que contemple a geração de uma linguagem de programação específica do domínio de robótica educacional, que proporciona ao usuário a simplicidade presente em linguagens de propósito específico, além de permitir a expressividade e um desempenho próximo de uma linguagem de propósito geral. Nesse sentido foi desenvolvida uma linguagem nomeada *Majestic Language - ML*, e o seu respectivo compilador.

1.4 Organização da Monografia

O trabalho está dividido em sete capítulos, descritos resumidamente a seguir: No Capítulo 2 consta o referencial teórico que apresenta as fundamentações teóricas que orientaram o estudo. O Capítulo 3 elenca os principais trabalhos relacionados a este. O Capítulo 4 ilustra a arquitetura e implementação do sistema utilizado para avaliação dos experimentos. O Capítulo 5 discute as principais características referentes ao funcionamento da linguagem desenvolvida. O Capítulo 6 apresenta a metodologia e experimentos realizados para avaliar a proposta. E por fim, o Capítulo 7 apresenta os principais pontos discutidos no trabalho

2 REFERENCIAL TEÓRICO

2.1 Introdução

Para a compreensão dessa pesquisa são apresentadas neste capítulo as definições utilizadas ao decorrer do texto. A Seção 2.1 apresenta a definição de Robótica Educacional, suas características e utilizações. A Seção 2.2 aborda os tipos de linguagens de programação e paradigmas utilizados na robótica. A Seção 2.3 apresenta as características do hardware utilizado na robótica educacional. A Seção 2.4 apresenta as etapas de compilação de um compilador moderno. Por fim, a Seção 2.5 apresenta as considerações finais do capítulo.

2.2 Robótica Educacional

O termo robótica foi enunciado pela primeira vez no ano de 1942 pelo escritor Isaac Asimov, na obra "Runaround", referindo-se a tecnologia baseada em robôs. Um robô consiste em um mecanismo multifuncional projetado para realizar o movimento de objetos através de uma programação prévia (ULLRICH, 1987).

As primeiras pesquisas sobre a robótica educacional foram desenvolvidas por um grupo de pesquisadores do *Massachusetts Institute of Technology - MIT*, em um projeto liderado pelo professor Seymour Papert, ao final do ano de 1960 (MIRANDA, 1990; RESNICK, 1993).

A Robótica Educacional é fundamentada na teoria do construtivismo e do construcionismo. A teoria do construtivismo aborda o aluno como construtor do próprio conhecimento, em que através da associação de conhecimentos prévios e interações, se é construído o novo conhecimento do aluno. O construcionismo utilizasse a abordagem de que a educação é um processo contínuo e exploratório sendo fundamentado no "aprender fazendo". Sendo realizada a construção e desconstrução de ideias com o auxílio de questões e objetos do mundo real; sendo os alunos estimulados a descobrirem e propor soluções para problemas presentes no mundo real, com o auxílio do conhecimento que já possuem (PAPERT; HAREL, 1991; ZILLI et al., 2004).

Nesse contexto, a robótica educacional pode ser introduzida como um ambiente composto por computador, componentes eletrônicos, componentes mecânicos e *software*, permitindo ao aluno a integração desses elementos para a construção de protótipos e testes de conceitos provenientes de múltiplas áreas de ensino (SILVA et al., 2014), tendo como meta produzir o maior aprendizado a partir do mínimo de ensino (PAPERT; HAREL, 1991).

É apontado por Zilli et al. (2004) que os objetivos da robótica educacional são de contribuir com:

- raciocínio lógico;
- habilidades manuais e estéticas;
- relações interpessoais e intrapessoais;
- utilização de 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 erros e acertos;
- aplicação das teorias formuladas a atividades concretas;
- utilização da criatividade em diferentes situações;
- capacidade crítica.

A Holanda e Alemanha fazem o uso de robótica educacional em escolas públicas e apresentam melhorias no desempenho dos alunos em todos os níveis de ensino (SILVA et al., 2014). Em outros países como Espanha, França, Romênia, Grécia e Itália houve aumento de projetos de pesquisa relacionados a robótica educacional (FRANGOU et al., 2008). Apesar de no Brasil não existir estatísticas nacionais de quantas escolas fazem o uso da robótica no processo de ensino-aprendizado, é perceptível o aumento de projetos de robótica e quantidade de participantes em competições de robótica. Segundo (SILVA et al., 2014), há um perceptível aumento da busca e implementação de projetos de robótica por escolas públicas e particulares.

Para atuar em um determinado ambiente, a robótica faz o uso tanto de sistemas computacionais, quanto de dispositivos elétricos e mecânicos. Os sistemas computacionais constituem a parte lógica, denominados *Softwares* e os dispositivos elétricos e mecânicos constituem a parte física, denominada *Hardware*, elementos fundamentais de qualquer kit robótico disponível para aquisição (SILVA et al., 2014).

2.2.1 Programação na Robótica Educacional

Linguagem de programação é um conjunto de instruções padronizadas utilizadas para a interação entre o homem e o dispositivo, através de um conjunto de regras, sendo possível especificar instruções capazes de realizar a execução de algoritmos para a resolução de problemas (SILVA et al., 2014). Essas instruções são notações com a finalidade de descrever computações tanto para máquinas quanto para pessoas. Portanto, todo *Software* necessita de uma linguagem de programação para especificá-lo (SETHI; ULLMAN; LAM, 2008; SILVA et al., 2014), sendo que antes de ser executado precisa anteriormente ser traduzido para um formato que seja reconhecido pela máquina-alvo para que então possa executar. Os sistemas responsáveis por essa tradução são denominados Compiladores (SETHI; ULLMAN; LAM, 2008).

Na Robótica Educacional, o conjunto entre a linguagem de programação e as peças físicas possibilita a programação de ações ou rotinas a serem cumpridas pelo robô (FRANGOU et al., 2008). Nessa perspectiva, a linguagem de programação é imprescindível para a robótica educacional proporcionando ao usuário a possibilidade de programar ações para seus protótipos (SILVA et al., 2014).

As primeiras aplicações da robótica educacional ocorreram na década de 80, com o auxílio da linguagem LOGO. Posteriormente, surgiram linguagens de programação visuais com o mesmo propósito, como por exemplo, a NXT que é utilizada em kits robóticos da Lego (FRANGOU et al., 2008). Com a ascenção dos kits robóticos e de seu comércio, novas linguagens de programação foram surgindo (JÚNIOR; GUEDES, 2015).

Em geral as linguagens de programação podem ser divididas em duas categorias: linguagens de propósito geral e linguagens de propósito específico. Linguagens de propósito geral são linguagens com grande poder expressivo, capazes de fornecer soluções para problemas de diversos domínios, por exemplo linguagens como C, COBOl, Java. Linguagens de propósito específico são linguagens cujas propriedades são desenvolvidas voltadas para a solução de problemas de um domínio em específico, por exemplo linguagens como PIC, LEX, YACC, HTML (DEURSEN; KLINT; VISSER, 2000).

As linguagens de programação utilizadas pela robótica educacional podem ser classificadas em linguagens textuais e linguagens gráficas. A programação gráfica consiste na utilização de blocos visuais encaixáveis para a descrição das ações que o programa deve realizar. A programação textual consiste na utilização de códigos compostos por fluxos de caracteres (JÚNIOR; GUEDES, 2015).

2.2.2 Programação Textual na Robótica Educacional

Programação Textual refere-se a linguagens de programação que são produzidas através de um fluxo de caracteres para o desenvolvimento do programa fonte. De forma

geral, esse tipo de linguagem é o mais difundido, com o auxílio de bibliotecas bem definidas (JÚNIOR; GUEDES, 2015). Essas linguagens e bibliotecas são em grande maioria escritas em linguagens de propósito geral, principalmente C (GOMES et al., 2016).

As principais linguagens textuais utilizadas na Robótica Educacional são adaptações baseadas em linguagens de propósito geral. Sendo essas adaptações, apenas o acréscimo de especificações das portas nas quais os sensores e atuadores estão conectados em relação ao *hardware* (QUEIROZ; SAMPAIO, 2016).

2.2.3 Programação Gráfica na Robótica Educacional

A dificuldade de compreensão da sintaxe das linguagens de programação existentes até o final dos anos 70 fez com que surgissem diversas pesquisas e iniciativas com o propósito de desenvolver linguagens de programação com sintaxes mais simples. Dessa forma, uma das alternativas à Programação textual foi o surgimento das VLPs - *Visual Programming Languages* ou Linguagens de Programação Visual, que são linguagens em que seu modelo de sintaxe é composto por expressões visuais de diferentes formatos (QUEIROZ; SAMPAIO, 2016).

Diversas linguagens de programação surgiram com esse propósito, tais como:

- Scratch: O design original da linguagem Scratch foi motivado pela necessidade e interesse de jovens (entre 8 e 16 anos) (RESNICK; KAFAI; MAEDA, 2005; RESNICK et al., 2009). A Figura 1 apresenta um exemplo de código desenvolvido utilizando Scratch;
- DB4K: A linguagem DB4K consiste em blocos com expressões sintáticas em português. Essa linguagem foi construída para funcionar com o auxílio da placa de prototipação Arduino (QUEIROZ; SAMPAIO, 2016). A Figura 2 apresenta um exemplo de código desenvolvido utilizando DB4K; e
- NXT-G: A linguagem NXT-G acompanha o kit Lego MindStorms, propriedades da empresa LEGO. Essa linguagem utiliza-se o formato de blocos que são combinados para realizar ações (FORNAZA; WEBBER; VILLA-BOAS, 2015). A Figura 3 apresenta um exemplo de código desenvolvido utilizando NXT-G.

Contudo, VLPs apresentam uma baixa expressividade em relação às linguagens textuais pois são restritivas apenas a comando de blocos, além de limitar o usuário a um número restritos de atividades assim limitando as possíveis aplicações (BRAVO; GONZÁLEZ; GONZÁLEZ, 2017).

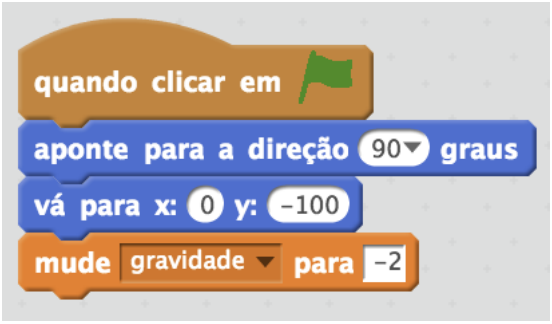

Figura 1 – Exemplo de código desenvolvido utilizando Scratch

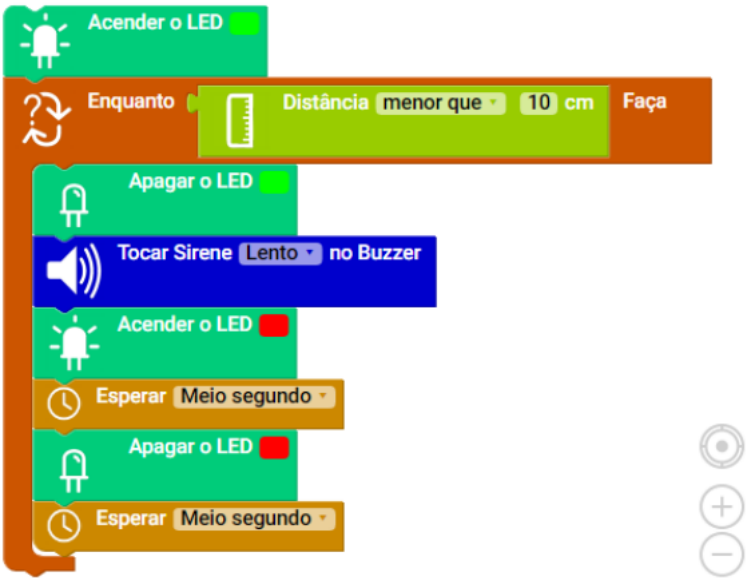

Figura 2 – Exemplo de código desenvolvido utilizando DB4K

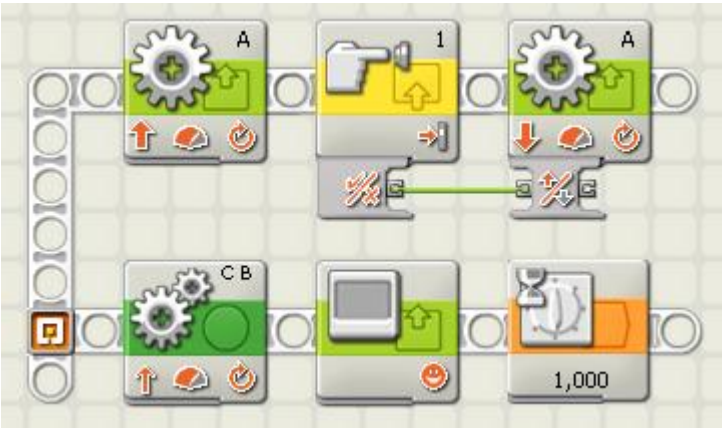

Figura 3 – Exemplo de código desenvolvido utilizando NXT-G

2.3 Compilador

Softwares em geral tem como objetivo facilitar a vida do usuário, através da otimização e automatização das atividades, além de solucionar problemas e auxiliar em atividades que demorariam um maior tempo sem o auxílio dos mesmos (PECORELLI, 2014). Esses *Softwares* por sua vez são sequências de instruções e operações representadas em uma linguagem de programação, que são abstraidas pelo *Hardware*. Contudo, as operações programadas pelo usuário são incompatíveis com as operações e linguagens utilizadas pelo *Hardware*. O *software* responsável por essa tradução é denominado Compilador(COOPER; TORCZON, 2003).

Os compiladores são sistemas de *software* que recebem como entrada um programa em uma denominada linguagem de programação (programa fonte) traduzindo-o em outra linguagem equivalente (programa objeto) que possa ser executado por um *hardware* (SETHI; ULLMAN; LAM, 2008). Para que essa tradução ocorra, o compilador deve compreender os aspectos da linguagem de origem, tais como, sintaxe, significados e formatos realizando assim o mapeamento da mesma para a linguagem alvo (COOPER; TORCZON, 2003). Durante esse processo o compilador deve detectar erros no código fonte, para que seja possível gerar um relatório para o usuário (PECORELLI, 2014).

O processo de compilação pode ser dividido em duas grandes partes, sendo elas o *front-end* e o *back-end*.

2.3.1 Front-End

O *Front-End* de um compilador tem como objetivo verificar se o código desenvolvido é compatível com a linguagem a ser utilizada, realizando a detecção de possíveis erros, preparando o código para a etapa do *Back-End* (SETHI; ULLMAN; LAM, 2008). Esse processo é dividido em quatro etapas principais, sendo elas análise léxica, análise sintática, análise semântica e gerador de código intermediário, conforme apresentado na Figura 4.

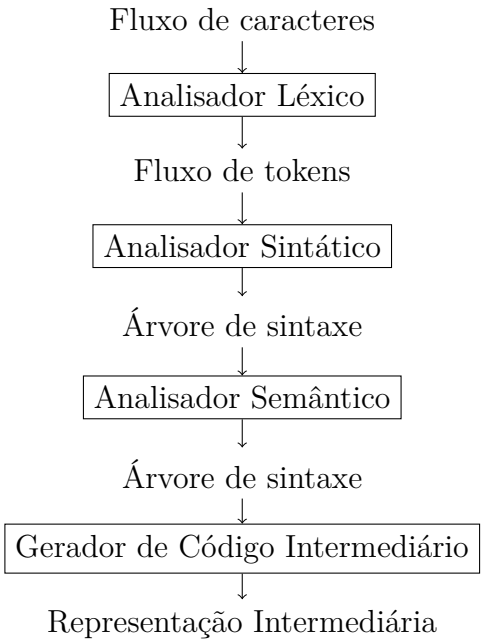

Figura 4 – Fases do Front-End de um Compilador (caixas), com sua entrada e saída respectiva. - Adaptado de (SETHI; ULLMAN; LAM, 2008).

2.3.2 Analisador Léxico

A primeira etapa de um compilador é chamada de análise léxica, sendo responsável por ler os caracteres do código fonte agrupando-os em unidades logicamente significativas, chamadas lexemas, e produz como saída tokens que representam esses lexemas (SETHI; ULLMAN; LAM, 2008). Um *lexema* é um fluxo de caracteres que são identificados com um padrão, que é identificado pelo analisador léxico. O processo da análise léxica é feito por um analisador léxico (PECORELLI, 2014).

Um Token é constituido por dois componentes, sendo eles *nome-token* e *valor-atributo* (SETHI; ULLMAN; LAM, 2008; FOLEISS et al., 2009). Os nomes de token serão posteriormente utilizados durante a análise sintática. processo de análise léxica também funciona como um verificador de alfabeto, em que verifica se o conjunto de caracteres contidos no código fonte existe ou não na gramática (SETHI; ULLMAN; LAM, 2008).

2.3.3 Analisador Sintático

A segunda etapa de um compilador é chamada de análise sintática. O processo da análise sintática é feita por um analisador sintático (PECORELLI, 2014). O analisador sintático utiliza-se dos tokens gerados pelo analisador léxico para realizar a construção de uma representação intermediária tipo árvore denominada árvore de sintaxe, que mostra a estrutura gramatical da sequência de tokens (SETHI; ULLMAN; LAM, 2008).

O analisador sintático possui duas formas distintas de trabalhar, sendo essas, *bottom-up* que faz a construção da árvore de derivação a partir dos nós folhas para o nó raiz e *top-down* que constrói a árvore começando pelo nó raiz em direção aos nós folhas (SETHI; ULLMAN; LAM, 2008).

Um dos desafios de verificar se uma cadeia está de acordo com a gramática desejada é o de garantir que a linguagem de programação não seja ambigua. Uma gramática é considerada ambigua, se for possível ter mais de uma árvore de derivação a partir de uma mesma cadeia. Como uma cadeia com mais de uma árvore de derivação possui mais de um significado, é necessário projetar gramáticas não ambíguas, ou usar regras para solucionar as ambiguidades (SETHI; ULLMAN; LAM, 2008). A Figura 5 apresenta um exemplo de gramática ambígua.

Figura 5 – Exemplo de ambiguidade para 9-5*2

2.3.4 Analisador Semântico

A terceira etapa de um compilador é chamada de análise semântica (SETHI; ULLMAN; LAM, 2008). Após a análise léxica e sintática o programa pode apresentar erros que impeçam a compilação do mesmo. O processo responsável por detectar esses erros é realizado por um analisador semântico. Para que se possa detectar esses erros, o compilador precisa analisar informações relacionadas ao contexto da linguagem e suas definições de dados específicos da linguagem projetada (COOPER; TORCZON, 2003).

Como informações comuns em compiladores temos a verificação de tipo, verificação de fluxo de controle e verificação de unicidade. Verificação de tipo consiste no compilador relatar erros caso o operador utilizado não seja compatível com o tipo declarado pelo programador (Exemplo: atribuição de uma string “texto” à uma variável do tipo inteiro). Caso uma estrutura ou instrução seja encerrada se é utilizada uma verificação de fluxo de controle, dessa forma o compilador deve saber retornar ao ponto anterior de execução para que possa continuar (Exemplo: Ao realizar uma chamada de função, após ser executada, deve-se retornar ao ponto de partida). Verificação de unicidade, consiste na verificação de declaração de identificadores distintos com o mesmo nome (Exemplo: Declaração de duas variáveis “variavelA”).

2.3.5 Gerador de Código Intermediário

O gerador de código intermediário é a última etada do *front-end* de um compilador, sendo responsável por gerar uma linguagem intermediária, mais próxima do código alvo e da tabela de símbolos (FOLEISS et al., 2009), porém ainda apresenta facilidade de manipulação (SETHI; ULLMAN; LAM, 2008). O LLVM [1] (*Low Level Virtual Machine*) possui uma biblioteca orientada a objetos para a geração de código intermediário. Esta linguagem gerada é independente de arquitetura, podendo ser utilizado para se obter o código alvo compatível para diferentes *Hardwares* (FOLEISS et al., 2009).

2.4 Hardwares utilizados na Robótica Educacional

A robótica é composta por sistemas computacionais e dispositivos elétricos e mecânicos que quando integrados podem obter informações a atuar em um determinado ambiente. Os dispositivos elétricos e mecânicos consistem na parte física, denominada *Hardware* (SILVA et al., 2014).

Alguns kits robóticos foram projetados para uma funcionalidade específica, tendo um conjunto limitado de componentes para a composição do *hardware* (FORNAZA; WEBBER; VILLA-BOAS, 2015).

A estrutura básica de um *hardware* para robótica é composta por:

- Microcontrolador: Responsável por fazer o gerenciamento lógico através da execução de instruções fornecidas pelo usuário;
- Estrutura: Consiste na parte externa do protótico, como o chassi e outras peças mecânicas;
- Componentes eletrônicos: Consiste em componentes sensores e atuadores que podem coletar ou reagir respectivamente em relação ao ambiente (JÚNIOR; GUEDES, 2015).

2.4.1 Microcontroladores

Microcontrolador é a parte do *hardware* responsável por executar as instruções recebidas, fazendo a interação de dispositivos de entrada e de saída, com o auxílio de uma central de processamento e uma memória interna (MARTINS; TEIXEIRA; AUGUSTO, 2016; SILVA et al., 2014).

Kits robóticos podem possuir diferentes microcontroladores, sendo os mais comuns:

[1] https://llvm.org/

- AVR: é um microcontrolador desenvolvido pela fabricante ATMEL. Normalmente comercializado de forma embutida em placas de prototipagem Arduíno (SILVA et al., 2014). Pode ser programado pelo Arduino IDE, com o auxilio de sua linguagem de programação baseada em C e C++, com o auxílio de bibliotecas adicionais (SILVA et al., 2014).

- PIC: é um microcontrolador desenvolvido pela fabricante Microchip (MARTINS, 2005).

- ARM: é uma arquitetura de processadores que operam com um conjunto de instruções reduzido. Permitindo um baixo consumo de energia (ADUSUMILLI, 2002).

2.4.2 Sensores e Atuadores

Sensores são dispositivos de entrada, cujo objetivo é coletar informações e convertê-las em sinais elétricos. Sensores podem ser classificados em dois tipos, sendo eles ativos e passivos. Sensores ativos são sensores que possuem sua própria fonte de radiação eletromagnética para coleta de dados, enquanto os sensores passivos atuam apenas com a radiação emitida pelo ambiente (SILVA et al., 2014).

Atuadores são dispositivos de saída, que permitem ao robô interagir diretamente com o ambiente. Os atuadores quando utilizados em conjunto com sensores, permitem ao robô reagir de forma diferente de acordo com as informações obtidas do ambiente e de sua programação (SILVA et al., 2014).

2.5 Considerações Finais

Neste capítulo foram descritos a programação na robótica educacional, linguagem de programação e seus paradigmas e etapas do processo de compilação (analisador léxico, sintático, semântico e gerador de código intermediário). Estes componentes são necessários para o projeto e a implementação da linguagem que foi desenvolvida nesse trabalho. Em resumo, são consideradas as seguintes características:

- Linguagem: A linguagem proposta é dinamicamente tipada, ou seja, não necessita de uma definição direta do tipo de variável. Essa atribuição ocorre de acordo com o valor passado para a variável durante o processo de análise semântica.

- Paradigma: A linguagem é estrutural, do tipo textual.

- Análise léxica: Foi utilizado o software *Flex* para essa tarefa;

- Análise sintática: Foi utilizado o software *Bison* para essa tarefa;

- Análise semântica: Foi utilizada a biblioteca IR do LLVM em conjunto com a linguagem C++ para realizar a análise semântica.

Estes componentes constituem o front-end do compilador. O *front-end* é integrado, através da linguagem intermediária IR do LLVM, ao *back-end* disponibilizado também pelo projeto LLVM, o qual traduz a linguagem intermediária em código alvo para processadores AVR e ARM.

3 TRABALHOS RELACIONADOS

Mapeamento sistemático é um processo que classifica e contabiliza as contribuições existentes em uma determinada área que possam dar uma visão geral da área de pesquisa (PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Para elaboração de um mapeamento sistemático é necessário definir o processo de busca, que envolve a definição dos argumentos a partir das questões de pesquisa e a seleção dos mecanismos de busca a serem utilizados, as questões de pesquisa, e o processo de seleção. Nesta última fase são definidos os critérios de inclusão e exclusão dos artigos encontrados. Uma vez que essa etapa termina os artigos selecionados têm seus dados extraídos para analisar e, posteriormente, responder as questões de pesquisa definidas.

3.1 Processo de busca

Para buscar por trabalhos relacionados, elaboramos uma lista de palavras chaves vinculadas ao objeto de estudo deste trabalho. As mesmas foram detalhadas em cinco grandes categorias de palavras- chaves: relacionadas a Programação (Programação, *Programming*), relacionadas a Linguagem de Programação (Linguagem de Programação, *Programming Language*), relacionadas à Compilador (Compilador, Compilador de Compilador,*compiler, compiler-compiler*), relacionado a Robótica (Robótica, *Robotics*), relacionadas à educação (educação, educacional, *education, learning, educational*) e relacionadas ao processo de avaliação (avaliação, estudo de caso, teste, validação, avaliação *improving, evaluate, case study, test, validation, assessment*)

Assim, foram combinadas as *strings* de busca e buscados artigos que atendessem o argumento de busca: para programação "(programação *OR programming*)* *AND* ((robótica *OR* robotics) AND (educação *OR* educacional *OR* education *OR* learning *OR* educational)) *AND* (avaliação *OR* teste *OR* validação *OR improving OR evaluate* OR case study OR test**)*", posteriormente fora replicada a *string* substituindo apenas a parte inicial que faz menção a programação pela referente a compilador.

Os trabalhos foram filtrados por meio do sistema de busca de periódicos da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), que possui bases referenciais importantes para este trabalho. As bases utilizadas foram *ACM Digital Library, ACM Computing Reviews*, *IEEE Xplore.*

3.2 Trabalhos analisados

Foram considerados trabalhos correlatos as publicações que compartilham dos conceitos principais desta pesquisa, isto é, investigamos trabalhos que descreviam linguagens de programação de propósito específico do domínio, textuais, o uso das mesmas em processos de ensino aprendizagem e observamos característica que facilitavam a programação para alunos iniciantes como tipagem dinâmica e automatizada. Foram analisados seis trabalhos e os três mais relevantes encontrados são descritos a seguir.

3.2.1 A Plataforma Universal Urbi para Robótica

Baillie (2005) em seu artigo apresenta a plataforma Urbi e demonstra como ela alcança seus objetivos. Urbi que é uma plataforma de software usada para desenvolver aplicativos portáteis para robótica e inteligência artificial, baseado em uma linguagem de*script* paralela e orientada a eventos e em uma arquitetura de componentes distribuídos. A plataforma foi projetada para ser usada tanto por crianças (seus usuários mais jovens conhecidos têm cerca de doze anos de idade) dispostas a personalizar seus robôs, até pesquisadores que desejam se concentrar em problemas científicos complexos em robótica

Um ponto critico relatado por Baillie (2005) é a falta de padrões em robótica que torna a plataforma Urbi particularmente sensível às diferenças de API (*Application Program Interfaces*) entre robôs, componentes e assim por diante. Os autores ainda relatam que vários desafios impedem a facilidade de uso, começando com a necessidade de lidar com uma ampla variedade de arquiteturas, embora a plataforma seja estruturada para lidar com a falta de padrões adequados de robótica além de ser simples, e poderosa.

Em relação das características de sintaxe da linguagem *UrbiScript* podemos destacar:

1. Pertence à família C: ele é inspirado em C, C++, e Java;
2. Dinamicamente tipada.
3. Funções são entidades de primeira classe: elas se comportam como valores comuns. Eles podem ser atribuídos a variáveis, passados como argumentos para funções e assim por diante.
4. Suporta fechamentos: as funções podem capturar referências de seu ambiente e depois usá-las para recuperar ou definir seu conteúdo.
5. Linguagem orientada a objetos (OOL): valores são objetos. Um objeto é composto por uma lista de protótipos (objetos pai) e uma lista de *slots*.

6. Baseado em protótipo ou sem classe. No OOL baseado em classe, as classes são modelos que descrevem o comportamento e os membros esperados de um objeto

Um ponto relevante que deve ser levado em consideração referente as linguagens de programação é que elas seguem um modelo de execução sequencial, ou seja, o código é executado uma instrução após a outra, levando ao resultado esperado. Os robôs são projetados para evoluir no mundo real: enquanto estão se movendo, eles precisam estar cientes de seu ambiente e de suas possíveis mudanças brutais, embora as linguagens de programação tradicionais normalmente ofereçam recursos assíncronos limitados por meio de sinais e retornos de chamada, Baillie (2005) relata que a plataforma Urbi os integra no coração do modelo de execução, e a linguagem *UrbiScript* fornece ao usuário construções curtas e intuitivas para o tratamento das intervenções.

A abordagem dos autores foi usada com sucesso para implantar alguns aplicativos complexos em cima de vários robôs diferentes, porém os mesmos relatam que ainda há muito a ser feito para alcançar o nível de padronização de que um mercado robótico maduro precisa finalmente ser capaz de cumprir suas promessas: um robô em todas as casas, ou quase.

3.2.2 *Rust*, uma linguagem de programação

Matsakis e II (2014) discorre sobre a eficiência e confiabilidade dos sistemas e nos apresenta a linguagem de programação denominada *Rust* que segundo os mesmos é uma nova linguagem de programação direcionada a aplicativos em nível de sistema para o desenvolvimento de sistemas confiáveis e eficientes. Esta linguagem é voltada para um maior controle da memória sem a necessidade de um coletor de lixo, essa estrategia facilita o desenvolvimento de *softwares*, e evita os frequentes erros de memória além de ser extremamente eficaz em sistemas embarcados. Projetada para oferecer suporte à simultaneidade e ao paralelismo na criação de aplicativos e bibliotecas que aproveitam ao máximo o hardware moderno, os autores ressaltam o quanto os sistema de tipo estático da *Rust* são seguros e expressivos, uma vez que os mesmos oferecem fortes garantias sobre isolamento, concorrência e segurança de memória.

Uma característica a ser analisada na *Rust* é a forma como ela oferece um modelo de desempenho claro, com intuito de facilitar a previsão e o raciocínio sobre a eficiência do programa. Para que isso seja possível ela permite um controle refinado sobre representações de memória, com suporte direto para alocação de pilha e armazenamento de registros contíguos. Matsakis e II (2014) relatam que o idioma equilibra esses controles com o requisito absoluto de segurança, o sistema e o tempo de execução do tipo*Rust* garantem a ausência de corridas de dados, estouros de *buffer*, estouros de pilha e acessos a memória não inicializada ou desalocada.

Apesar de *Rust* ser uma linguagem de programação que permite uma ampla compatibilidade com *Hardware*, por ser uma linguagem complexa, por ser voltada para usuários experientes requer um conhecimento prévio por parte do usuário, não sendo trivial para iniciantes.

Em relação das características de sintaxe da linguagem *Rust* podemos destacar:

1. Dinamicamente tipada.
2. Funções são entidades de primeira classe: elas se comportam como valores comuns. Eles podem ser atribuídos a variáveis, passados como argumentos para funções e assim por diante.

3.2.3 Ensino com robôs em escolas secundárias: alguns problemas pedagógicos novos e não tão novos

Dagdilelis, Sartatzemi e Kagani (2005) salientam que programadores iniciantes frequentemente enfrentam dificuldades significativas no entendimento dos conceitos de programação e na busca de soluções para problemas de programação mesmo elementares. Diversas estratégias são utilizadas para tentar combater essa problemática e segundo os autores uma das estratégias básicas adotadas para enfrentar esse problema foram as linguagens de programação e os ambientes criados com o objetivo específico de facilitar os processos de ensino e aprendizagem da programação. Desses tipos de ambientes se destaca uma categoria típica que consiste em sistemas nos quais o programador iniciante manipula entidades reais, como robôs e mecanismos automáticos.

Dado o exposto Dagdilelis, Sartatzemi e Kagani (2005) apresentam em sua pesquisa uma série de lições piloto para uma introdução à programação com a ajuda do *Lego Mindstorms*[1] e o ambiente de programação *visual ROBOLAB*. A pesquisa foi realizada com alunos do ensino médio grego nas 9ª e 10ª séries e apresentamos os resultados iniciais, que mostram claramente as vantagens desse ambiente em relação aos padrões convencionalmente utilizados. Dentre os pontos avaliados no trabalho se destacam a compreensão do mecanismo básico de funcionamento dos robôs e compreender o significado exato dos comandos, especialmente porque certos comandos expressam conceitos que não são comuns na vida cotidiana. Segundo os autores as aulas introdutórias tiveram duração de seis horas e ocorreram em duas turmas de 15 alunos e as lições aplicadas foram divididas em 3 lições, são elas: Lição 1: Introdução à programação com *LEGO*; Lição 2: Comando de Saída, Comando de Entrada (Sensores), Comando Aguardar, Modificadores; e por fim Lição 3: Estrutura de Repetição.

[1] *Lego Mindstorms* é um robô produzido em larga escala pela fabricante *lego*

Como o ponto de vista adotado pelos autores para a avaliação do *Lego Mindstorms* é o da pedagogia os aspectos avaliados do uso da ferramenta foram através de um exame de questões didáticas contemporâneas, ou seja, o interesse do trabalho e saber até que ponto os recursos do sistema melhoram ou não o processo de ensino e aprendizagem de programação. Dagdilelis, Sartatzemi e Kagani (2005) relatam que os experimentos permitiram que eles adquirissem um conhecimento técnico elementar sobre os usos educacionais dos sistemas robóticos e, especialmente, deste sistema em particular, o que foi de suma importância uma vez que muitos dos fatores que desempenham um papel no ensino eram desconhecidos para eles.

Como resultado dos trabalhos os autores destacam que o entendimento e o uso correto dos conceitos básicos de programação, que era o objetivo inicial na organização dos planos de aula, parece ser facilitado com o uso de sistemas robóticos, uma vez que certos conceitos e processos de programação são óbvios em um sistema que possui entidades reais, porém chamam a atenção no que diz respeito a certos conceitos como a criação e o uso de números aleatórios que parecem permanecer difíceis de entender para programadores iniciantes, para outros como estruturas de repetição os autores não puderam declarar nada com certeza.

No que diz respeito à linguagem de programação, o trabalho relata que a escolha da empresa de usar símbolos diferentes para cada tipo de atividade dificulta que os iniciantes memorizem os vários símbolos. No entanto, relatam que o ambiente gráfico *ROBOLAB*, usado nas atividades praticas pelos autores, facilitou muito o processo de criação e execução de programas para robôs. Os resultados mostram como o uso e a programação de entidades reais colocam os alunos em contato, não apenas com a tecnologia, mas também com as dificuldades técnicas que o uso de sistemas reais cria, fato que segundo Dagdilelis, Sartatzemi e Kagani (2005) tem um valor educacional muito alto.

Por fim os autores relatam que o uso do *Lego Mindstorms* tem, em geral, resultados positivos no nível em que estão interessados, bem como nos objetivos educacionais que foram estabelecidos. Portanto, continuaram as lições expandindo-as tanto em termos de tempo quanto de número e nível de alunos envolvidos.

3.3 Resumo Comparativo

Esta sessão tem por objetivo relacionar a utilização das características estudadas neste trabalho que serão detalhadas a seguir. A Tabela 1 apresenta a relação dos trabalhos aqui citados e em qual deles foram respectivamente utilizados, aos trabalhos que utilizam uma das características propostas será atribuído ”+” e para os que não utilizam será atribuído ”-”.

Referente as características avaliadas em cada trabalho relacionado e que fazem

parte do conjunto de características proposta nesse trabalho detalhamos as seguintes:

- C1 - Possibilidade de implementação para robôs com características distintas;
- C2 - Facilidade de implementação de vetores e matrizes;
- C3 - Permite implementação de algoritmos com o auxílio de funções e parâmetros de funções.
- C4 - A linguagem de programação, biblioteca ou compilador é voltada para a Robótica Educacional.
- C5 - A linguagem de programação ou biblioteca pode ser compilada diretamente, ou seja, não necessita de tradução

Tabela 1 – Trabalhos e Seus Respectivos Métodos

Trabalhos	C1	C2	C3	C4	C5
(BAILLIE, 2005)	+	+	+	-	-
(MATSAKIS; II, 2014)	+	+	+	-	+
(DAGDILELIS; SARTATZEMI; KAGANI, 2005)	-	-	+	-	+
PROPOSTA DESTA PESQUISA	+	+	+	+	+

É importante ressaltar que os estudos relacionados apresentam características positivas, entretanto nenhum deles explora a possibilidade do uso delas em conjunto, analisando e explorando o que cada uma tem de importante e que possa vir a contribuir em conjunto.

Isso reforça a ideia de que o assunto é relevante. Assim sendo, enxerga-se como a principal contribuição para a área, o fato deste trabalho propor em conjunto as características descritas.

4 LINGUAGEM DE PROGRAMAÇÃO

Neste capítulo é apresentado como a Linguagem de Programação (*Majestic Language - ML*) e o compilador da mesma foram desenvolvidos, demonstrando o funcionamento e a utilização da linguagem bem como as ferramentas e metodologias que foram aplicadas no desenvolvimento do compilador.

4.1 Linguagem de Programação ML

A ML é uma linguagem de programação textual, ou seja, o usuário pode programar ações ou comportamentos que devem ser tomados através da escrita de instruções que serão reconhecidas pela linguagem.

A linguagem ML é fracamente tipada, logo, o usuário não declara os tipos das variáveis, basta atribuir os valores e o compilador reconhece o tipo que se adéqua ao valor recebido, atribuindo-o automaticamente à variável. Quanto a alocação de memória da linguagem ML, é feita de maneira estática, ou seja, o tamanho de vetores e matrizes não podem ser alterados em tempos de execução.

A seguir, detalhamos outras funcionalidades e comportamentos da linguagem.

4.1.1 Operadores

Os operadores lógicos e aritméticos aceitos pela linguagem de programação ML, foram escolhidos com base nas linguagens de propósito geral, uma vez que é usado amplamente em diferentes casos, o que facilita o entendimento e a compreensão as mesmas são apresentados na Tabela 2 juntamente com os caracteres aceitos pela gramática.

4.1.2 Tipos de dados em variáveis

A proposta deste trabalho utilizou dos tipos de variáveis definidas no ROBCMP, uma vez que já foram realizados testes com o mesmo e obteve-se um resultado satisfatório. Porém foi acrescentado o tipo de variável Booleana com intuito de se obter o máximo proveito da linguagem de programação. Os tipos de variáveis utilizados são apresentados no Código 4.1.

Código 4.1 – Exemplo de atribuição de valores e tipos de dados

```
1 A = 3; //Atribui o valor inteiro (3) à variável de nome A.
2
3 B = 4.5; //Atribui o valor real (4.5) à variável de nome B.
```

Tabela 2 – Símbolos dos Operadores suportados pela linguagem ML

Símbolo	Significado
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
=	Atribuição
==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
AND	Operador lógico 'E'
OR	Operador lógico 'OU'

```
4
5 C = "String"; //Atribui o valor caractere (string) à variável de nome C.
6
7 D = true; //Atribui o valor booleano (true) à variável de nome D.
```

4.1.3 Estruturas de Decisão

Já para estruturas de decisão não ouve a necessidade de se alterar a forma proposta na linguagem usada como base. Sendo assim o usuário poderá definir tomadas de decisões a partir de uma expressão lógica, precedida pela palavra reservada 'if'. Opcionalmente, o usuário pode definir sequências de instruções a serem executadas caso a condição anterior não tenha sido satisfeita, através do uso da palavra reservada 'else' após a finalização do bloco de instruções da condição anterior.

Código 4.2 – Exemplo de uso de estrutura de condição

```
1 if (/*Operação Lógica*/){
2     //Conjunto de instruções a serem executadas caso a decisão seja verdadeira.
3 }
4 else{
5     //Conjunto de instruções a serem executadas caso a decisão seja false.
6 }
```

4.1.4 Vetores e Matrizes

Vetores e matrizes são uma parte crucial desta proposta uma vez que sua disponibilização para uso do programador não faz parte, ou seja, não foi implementada na linguagem de programação usada como referência. A escolha da utilização das mesmas facilita na implementação de códigos simples e complexos, podemos citar como casos específicos o mapeamento de ambientes, onde o robô poderá fazer uso dessas estruturas para facilitar o acesso das informações.

Os vetores e as matrizes são estrutura de dados capazes de associar um conjunto de valores a uma variável. Os valores associados a um vetor são do tipo primitivo, enquanto, uma matriz possui valores do tipo vetor, ou seja, uma matriz é definida por um vetor de vetores.

Com a finalidade de facilitar a atribuição de valores de um vetor na linguagem ML, optou-se por adicionar à característica na declaração de vetores, a possibilidade de atribuição de valores com repetições através do uso do carácter ':'. Assim como as variáveis, os vetores e as matrizes, também são fracamente tipados reconhecendo automaticamente o tipo mais adequado aos dados.

Código 4.3 – Exemplo de declaração de vetores e matrizes

```
1 A = { 3, 5, 2 }; //Vetor de inteiros.
2
3 B = { 3.0, 1, 27}; //Vetor de números reais.
4
5 C = { 0:30 }; //Vetor de 30 posições, preenchido com zeros.
6
7 D = {{ 3.0:22 }:3}; //Matriz de 22x3 de números reais, preenchido com 3,0 .
```

A leitura e manipulação de uma posição de um vetor é realizada através do uso de uma expressão entre '[' e ']', permitindo o uso de operações aritméticas, uso de variáveis e até mesmo posições de outros vetores e retornos de funções.

Código 4.4 – Exemplo de manipulações com vetores e matrizes

```
1 a = { 3, 2, 0 };
2 b = {{ 2, 7, 5 }:3};
3
4 print a[0]; //Exemplo de leitura de dado em um vetor.
5 a[0] = 25; //Exemplo de alteração de valor na primeira posição do vetor.
6 a[1] = a[3-1]; //Exemplo de alteração de valor com o auxílio de uma expressão.
7
8 print b[0][0]; //Exemplo de leitura de dado em uma matriz.
9 b[1][2] = 3; //Exemplo de alteração de valor em uma matriz.
```

4.1.5 Funções

As funções são uma das funcionalidades que facilitam na programação de robótica em geral, visto que auxilia no desenvolvimento a partir do conjunto de instruções a serem executados sem que haja a necessidade de repetição de código, além de permitir flexibilidade do reuso do mesmo. A linguagem proposta nesse trabalho suporta a criação de funções, desde que sejam declarados os tipos de retorno e dos seus parâmetros, visto que a proposta da linguagem é de que as funções não atuem como *Template*[1]. Os tipos de dados da função são definidos na Tabela 3.

Tabela 3 – Tipos de Dados suportados pela função

Termos	Tipo	Tamanho
void	Vazio	
bool	Booleano	1 bit
short int	Inteiro	8 bits
int	Inteiro	16 bits
long int	Inteiro	32 bits
long long int	Inteiro	64 bits
long long long int	Inteiro	128 bits
short float	Real	8 bits
float	Real	16 bits
double	Real	32 bits
double double	Real	64 bits

Os parâmetros de função têm por objetivo definir as atribuições de valores utilizadas pelas variáveis baseada na necessidade da mesma. Diante disso utilizaremos os tipos de dados listados na Tabela 3, com exceção do tipo apresentado na primeira linha da tabela, uma vez que não será possível declarar uma variável do tipo "vazio".

Outra parte importante a ser lembrada é retorno da função, que tem por objetivo devolver o resultado obtido apos as operações realizadas para a função requisitante, os tipos de dados suportados pelo retorno da função são os mesmos listados na 3.

Código 4.5 – Exemplo de declaração de função com parâmetros

```
1 float multiply (int variavel, float variavel_2){
2     return variavel*variavel_2;
3 }
```

[1] Template: Permite a função e seu tipo de retorno possuir um tipo genérico

4.2 Estruturas de repetição

Como estruturas de repetição são consideradas importantes em uma linguagem de programação, e por elas já serem utilizadas no ROBCMP, optou-se por mantê-las na proposta deste trabalho, sendo assim para a utilização de estrutura de repetição na linguagem ML, o usuário deverá especificar uma condição que, enquanto a mesma for verdadeira, executa-se um conjunto de rotinas.

Código 4.6 – Exemplo de estrutura de repeticao

```
1 while(/*Condição*/){
2     //Instruções a serem repetidas até que a verificação da condição seja falsa.
3 }
```

4.3 Portas de Entrada/Saída

O uso de portas de entrada e saída é o método utilizado para integração entre o *software* e as portas presentes no *harware*, o que possibilita a manipulação do comportamento de um robô.

Com isso, a linguagem de programação ML permitirá a manipulação de portas de entrada e de saída de um microcontrolador. O valor passado, faz manipulação da voltagem emitida para a porta, de modo que esse sinal seja reconhecido pelos sensores e atuadores. O uso de controle de entrada e saída, é feito através do uso das palavras reservadas '*IN*' e '*OUT*' respectivamente. Sendo seguidas pelo número da porta em que será manipulada, e o valor que será passado.

Código 4.7 – Exemplo de estrutura de repeticao

```
1 if (IN8 == 0){ //Ao não ser reconhecido sinal positivo na porta 8
2     OUT2 = 255; //É realizada a passagem de valor para a porta 2.
3 }
```

4.4 Compilador

Após definidas as características da linguagem de programação ML, foram geradas as expressões regulares referentes ao procedimento de análise léxica da linguagem, utilizando o *software Flex*. Em seguida, foram geradas a gramática referente ao procedimento de análise sintática, com o auxilio do *software Bison*. Com o uso da linguagem C++ concluiu-se o desenvolvimento do *frontend* do compilador. Posteriormente o *frontend* gerado foi integrado ao *backend* para AVR, contido no LLVM.

4.5 Construção do Compilador

Após definidas as características e funcionalidades da linguagem ML, foi necessário realizar modificações no protótipo ROBCMP, com a finalidade de que o mesmo pudesse reconhecer a gramática proposta.

Para o desenvolvimento do compilador, inicialmente foram necessárias as instalações das tecnologias de apoio, apresentadas na Seção 4.6. Posteriormente foram realizadas as seguintes modificações sobre os arquivos presentes no compilador ROBCMP, sendo essas:

Foram modificadas as expressões regulares presentes no arquivo rob.l, de maneira a se adequarem as modificações proposta pela linguagem ML, adicionando assim os comandos necessários aos novos recursos.

No arquivo rob.y, foram adicionadas sintaxes referentes à gramática proposta pela linguagem ML. Dentre as novas sintaxes adicionadas encontram-se declaração de vetor, matriz, função, e seus respectivos parâmetros.

O arquivo node.h, foi refatorado de acordo com as novas funcionalidades presentes na linguagem ML. Sendo então criadas novas classes, sendo elas:

- ArrayElements: Responsável por fazer o gerenciamento de elementos de um vetor e suas repetições.
- Int (1 bit, 32 bits, 64 bits, 128 bits): Tem por finalidade a instância de elementos do tipo Inteiro.
- Float (8 bits, 64 bits), Double: Tem por finalidade a instância de elementos do tipo Real.
- Vector: Atribuição e Instancia de Vetores, baseado na quantidade de repetições utilizadas pela classe ArrayElements.
- LoadVector: Permite obter os valores dos elementos de um vetor.
- UpdateVector: Responsável pela modificação de um vetor.
- MatrixElements: Responsável por fazer o gerenciamento de elementos e suas repetições.
- Matrix: Atribuição e Instancia de um Vetor de Vetores, utilizando como base a quantidade de repetições fornecidas pela classe MatrixElements.
- LoadMatrix: Permite obter os valores dos elementos de uma matriz.
- UpdateMatrix: Responsável pela modificação de uma matriz.

- ParamsCall: Classe que gerência os tipos de chamadas de uma função, para que se possa ser feito a recuperação de dados dos parâmetros.

Foram atreladas as sintaxes declaradas no arquivo Language.y às suas respectivas classes, responsáveis por tais funcionalidades.

4.6 Tecnologias de Apoio

Para a construção do compilador da linguagem ML, foi utilizado como base o *framework* Robcmp[2], disponibilizado na plataforma de desenvolvimento de *software* GitHub. O Robcmp é utilizado para a construção de *Front-End* de compiladores, utilizado como auxílio nas aulas de compiladores ministradas para o curso Ciências da Computação na Universidade Federal de Jataí. Para o desenvolvimento desse projeto contaremos também com o auxílio das tecnologias C++ e LLVM.

[2] Repositório do *framework* ROBCMP: http://github.com/thborges/robcmp

5 AVALIAÇÃO E TESTES

Neste capítulo será apresentado os experimentos realizados com o compilador e com a linguagem de programação. Os experimentos foram realizados com o propósito de validar o compilador e a linguagem proposta, sendo realizados os testes em um Sistema Operacional Debian 10 - Linux 4.9.0-11, Processador i7-8565U, 16 GB Memória RAM. Para os procedimentos realizados com o compilador foram analisados os seguintes aspectos.

- Tempo de compilação
- Tamanho do arquivo de código alvo gerado.

Para a realização dos experimentos, foram desenvolvidos 9 algoritmos, com características distintas, tanto na linguagem proposta quanto em C, sendo cada algoritmo equivalente em cada linguagem. Cada algoritmo foi compilado cinco vezes utilizando os mesmos parâmetros e as mesmas *flags* de otimização '-Os'.

5.0.1 Experimentos com Tempo de Compilação

O processo de compilação e seu tempo é um atrativo para programadores de todos os níveis.

Para realizar a avaliação do tempo de compilação do compilador proposto, foi utilizado o auxílio do comando '*time*', o qual é nativo do sistema operacional, uma vez que esse comando exibe o tempo utilizado para a execução do processo.

Os algoritmos utilizados para os experimentos foram levados em consideração as seguintes características e estão disponíveis no Apêndice A:

- 1 programa de *firmware* básico para um braço robótico;
- 2 Algoritmos de Ordenação de Vetores (*Bubble Sort* e *Insertion Sort*);
- 1 Algoritmo de Programação Dinâmica (Caminho Mínimo, Bellman-Ford);
- 1 Algoritmo de Multiplicação de Matriz;
- 3 Algoritmos Recursivos (Fibonacci, Fatorial, Potência);
- 1 Algoritmo de busca de elemento em Vetor;

Tabela 4 – Comparação de tempo de compilação entre as Linguagem ML e C

Algoritmo	ML	C (gcc)
Insertion Sort	0.080	0.0816
Bellman-Ford	0.053	0.0738
Algoritmo de Recursividade (Fatorial)	0.0634	0.0732
Bubble Sort	0.0926	0.0913
Multiplicação de Matrizes	0.0682	0.074
Recursividade (Potência)	0.0822	0.0762
Recursividade (Fibonacci)	0.0923	0.0892
Busca Elemento em Vetor	0.0739	0.0794
Braço Robótico	0.0865	0.0823

A Tabela 4 apresenta a média dos tempos de compilação, em segundos, obtidos pelos experimentos realizados com os algoritmos no compilador ML e a média do tempo em segundos dos algoritmos equivalentes em linguagem C.

Durante a execução dos procedimentos, apenas o processo de compilação estava sendo executado na máquina em conjunto com o Sistema Operacional, sendo auxiliado pelo comando '*time*', fornecido pelo próprio Sistema Operacional.

A partir dos resultados é possível observar que o compilador ML compilou aproximadamente 14.6% mais rápido do que o compilador gcc ao gerar um código equivalente na linguagem C.

5.0.2 Experimentos com Tamanho de Arquivo

No aspecto de tamanho do arquivo de código alvo, devido à baixa capacidade de memória disponíveis nos microcontroladores para o armazenamento dos *softwares* e de seu próprio sistema, surge a necessidade de um código funcional.

Durante o procedimento de experimentos de tamanho de arquivo, foram utilizados os mesmos 9 algorítmos desenvolvidos para o experimento de tempo. Como o tamanho do arquivo é determinístico, a geração do arquivo .o foi feita apenas uma vez.

A Tabela 5 apresenta a média dos tempos de compilação, em segundos, obtidos pelos experimentos realizados com os algoritmos no compilador ML e em linguagem C utilizando o compilado gcc.

A partir dos resultados é possível observar que o compilador ML gerou um arquivo

Tabela 5 – Comparação de tamanho de arquivo entre as Linguagem ML e C

Algoritmo	ML	C (gcc)
Insertion Sort	1432	1480
Bellman-Ford	1445	1463
Algoritmo de Recursividade (Fatorial)	1441	1437
Bubble Sort	1447	1453
Multiplicação de Matrizes	1452	1431
Recursividade (Potência)	1444	1421
Recursividade (Fibonacci)	1523	1495
Busca Elemento em Vetor	1339	1494
Braço Robótico	1265	1282

final aproximadamente 12.3% menor do que o gerado pelo compilador gcc em um código equivalente na linguagem C.

5.0.3 Considerações Finais

Foi verificado que em tanto no experimento de tempo, quanto de tamanho final de arquivo o compilador ML apresentou um ganho em relação ao compilador C.

Durante a fase de experimentos foi utilizado o GCC para arquiteturas X86_64 devido ao fato do *backend* da arquitetura avr para o LLVM ainda ser experimental. Experimentos devem ser realizados no futuro com o avr-gcc juntamente com o Robcmp para avr.

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo serão apresentadas as conclusões e contribuições obtidos na pesquisa, através do desenvolvimento da linguagem ML e de seu compilador, além de propor temas para futuras pesquisas relacionadas à esta.

6.1 Conclusões

Considerando a necessidade de uma linguagem de programação voltada para a Robótica Educacional, que seja simples e de baixa complexidade e ao mesmo tempo pouco limitada, essa pesquisa propôs a linguagem de programação ML.

Esta linguagem faz uso do paradigma estruturado, representado textualmente, que implementa as principais características de uma linguagem de programação de propósito geral, além de possuir propriedades de uma linguagem de propósito específico de domínio, como simplicidade na declaração de vetores, matrizes e funções, os quais são fundamentais para a evolução da programação de protótipos de robôs.

A linguagem de programação C++, frequentemente utilizada para a programação de protótipos é compatível com a utilização de diversos tipos de atuadores e sensores. Como o propósito deste trabalho é permitir que o programador tenha acesso a uma linguagem que seja simples e ao mesmo tempo não seja limitada, a linguagem ML foi desenvolvida com o propósito de ser compatível com a utilização de diversos tipos de sensores ou atuadores, assim, sendo capaz de gerar códigos para tarefas tanto simples, quanto complexas.

Para que se fosse possível desenvolver *softwares* com o uso da linguagem ML, foi desenvolvido um *frontend* de um compilador capaz de realizar a análise léxica, sintática e semântica, verificando se o *software* fornecido pelo usuário é compatível com a linguagem, e em seguida fazendo a integração com o *backend* LLVM.

Ao final dessa pesquisa obteve-se uma linguagem de programação de propósito específico do domínio de robótica educacional, capaz de facilitar e auxiliar na programação de kits robóticos por usuários que tenham alguma experiência na programação de códigos estruturados. A linguagem e o programador da linguagem resultante é de domínio público e de código-aberto.

Através dos experimentos foi possível concluir que a linguagem ML apresentou um menor tempo durante o processo de compilação, além de um tamanho de arquivo inferior à linguagem C++.

6.2 Trabalhos futuros

Como possíveis melhoras para a linguagem de programação ML, considera-se como trabalhos futuros a adição das seguintes funcionalidades:

- Experimento AVR: Fazer experimentos comparativos com o avr-gcc e o *backend* AVR do LLVM.
- Arquivo externo: o arquivo externo permitiria ao desenvolvedor interligar o código desenvolvido na linguagem de programação ML, com códigos desenvolvidos em C++, para que possa ser utilizadas instruções que ainda não foram implementadas nativamente pela linguagem.
- Implementação de funções unárias: característica importante para operações lógicas.
- Ponteiro: a implementação de ponteiros permitiria, ao usuário da linguagem ML, o uso de ponteiros, o que permitiria o aumento da gama de possibilidades do uso de funções e outras funcionalidades da linguagem.
- Estruturas de dados: a implementação de estruturas de dados permitiria, ao usuário da linguagem ML, a otimização do armazenamento de dados, além do acesso mais eficiente, aproveitando melhor os recursos limitados disponibilizados pelo microcontrolador das plataformas de prototipação

REFERÊNCIAS

ADUSUMILLI, S. *System and method to reduce power consumption in advanced RISC machine (ARM) based systems*. [S.l.]: Google Patents, 2002. US Patent 6,438,700. Citado na página 27.

ARDUINO, S. A. Arduino. *Arduino LLC*, 2015. Citado na página 16.

BAILLIE, J.-C. Urbi: Towards a universal robotic low-level programming language. In: IEEE. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.], 2005. p. 820–825. Citado 4 vezes nas páginas 16, 30, 31 e 34.

BAILLIE, J.-C. et al. The urbi universal platform for robotics. In: *First International Workshop on Standards and Common Platform for Robotics*. [S.l.: s.n.], 2008. Citado na página 16.

BARKER, B. S. *Robots in K-12 education: A new technology for learning: A new technology for learning*. [S.l.]: IGI Global, 2012. Citado na página 14.

BRAVO, F. A.; GONZÁLEZ, A. M.; GONZÁLEZ, E. A review of intuitive robot programming environments for educational purposes. In: IEEE. *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*. [S.l.], 2017. p. 1–6. Citado na página 21.

BRUYNINCKX, H. Orocos: design and implementation of a robot control software framework. In: CITESEER. *Proceedings of IEEE International Conference on Robotics and Automation*. [S.l.], 2002. Citado na página 16.

CASADO, C. et al. Robótica educacional: Estímulo à aprendizagem de conceitos científicos por alunos da educação básica. *Anais do Salão Internacional de Ensino, Pesquisa e Extensão*, v. 7, n. 3, 2016. Citado na página 14.

COOPER, K.; TORCZON, L. *Engineering a compiler second edition*. [S.l.]: San Francisco, 2003. Citado 2 vezes nas páginas 23 e 25.

DAGDILELIS, V.; SARTATZEMI, M.; KAGANI, K. Teaching (with) robots in secondary schools: some new and not-so-new pedagogical problems. In: IEEE. *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*. [S.l.], 2005. p. 757–761. Citado 4 vezes nas páginas 15, 32, 33 e 34.

DEURSEN, A. V.; KLINT, P. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology*, SRCE-Sveučilišni računski centar, v. 10, n. 1, p. 1–17, 2002. Citado na página 15.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, ACM, v. 35, n. 6, p. 26–36, 2000. Citado 2 vezes nas páginas 15 e 20.

EGUCHI, A. What is educational robotics? theories behind it and practical implementation. In: ASSOCIATION FOR THE ADVANCEMENT OF COMPUTING IN EDUCATION (AACE). *Society for information technology & teacher education international conference*. [S.l.], 2010. p. 4006–4014. Citado 2 vezes nas páginas 14 e 15.

FILIPPOV, S. et al. Robotics education in saint petersburg secondary school. In: SPRINGER. *International Conference on Robotics and Education RiE 2017*. [S.l.], 2017. p. 38–49. Citado 2 vezes nas páginas 14 e 15.

FOLEISS, J. H. et al. Scc: Um compilador c como ferramenta de ensino de compiladores. In: *WEAC2009-Workshop Educação em Arquitetura de Computadores*. [S.l.: s.n.], 2009. p. 15–22. Citado 2 vezes nas páginas 24 e 26.

FORNAZA, R.; WEBBER, C.; VILLA-BOAS, V. Kits educacionais de robótica: opções para o ensino de ciências. *Scientia cum Industria*, v. 3, n. 3, p. 142–147, 2015. Citado 2 vezes nas páginas 21 e 26.

FOWLER, M. *Domain-specific languages*. [S.l.]: Pearson Education, 2010. Citado na página 15.

FRANGOU, S. et al. Representative examples of implementing educational robotics in school based on the constructivist approach. In: *Workshop Proceedings of simpar*. [S.l.: s.n.], 2008. p. 54–65. Citado 2 vezes nas páginas 19 e 20.

FREITAS, L. A. et al. Programação de espaços inteligentes utilizando modelos em tempo de execução. Universidade Federal de Goiás, 2017. Citado na página 15.

GOMES, J. et al. Guaráteca: Uma poderosa biblioteca de funções para robôs baseados em arduino. *Mostra Nacional de Robótica*, p. 1–6, 2016. Citado 2 vezes nas páginas 16 e 21.

HUDAK, P. Modular domain specific languages and tools. In: IEEE. *Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203)*. [S.l.], 1998. p. 134–142. Citado na página 15.

JOHNSON, J. Children, robotics, and education. *Artificial Life and Robotics*, Springer, v. 7, n. 1-2, p. 16–21, 2003. Citado na página 14.

JÚNIOR, A. d. O. C.; GUEDES, E. B. Uma análise comparativa de kits para a robótica educacional. 2015. Citado 3 vezes nas páginas 20, 21 e 26.

LOPES, M. S. d. S. Ambiente colaborativo para ensino aprendizagem de programação integrando laboratório remoto de robótica. Escola Politécnica, 2017. Citado na página 16.

LÓPEZ-RODRÍGUEZ, F. M.; CUESTA, F. Andruino-a1: Low-cost educational mobile robot based on android and arduino. *Journal of Intelligent & Robotic Systems*, Springer, v. 81, n. 1, p. 63–76, 2016. Citado 2 vezes nas páginas 14 e 16.

MAJOR, L.; KYRIACOU, T.; BRERETON, O. P. Systematic literature review: teaching novices programming using robots. *IET software*, IET, v. 6, n. 6, p. 502–513, 2012. Citado na página 14.

MARTINS, A.; TEIXEIRA, A. C.; AUGUSTO, F. V. O desenvolvimento da criatividade através da robótica educacional. *Medi@ ções*, v. 4, n. 1, p. 4–18, 2016. Citado 2 vezes nas páginas 14 e 26.

MARTINS, N. A. Sistemas microcontrolados. *São Paulo: Novatec Editora*, 2005. Citado na página 27.

MATSAKIS, N. D.; II, F. S. K. The rust language. In: ACM. *ACM SIGAda Ada Letters*. [S.l.], 2014. v. 34, n. 3, p. 103–104. Citado 2 vezes nas páginas 31 e 34.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, ACM, v. 37, n. 4, p. 316–344, 2005. Citado na página 15.

MIRANDA, G. L. Linguagem logo. *Análise Psicológica*, Instituto Superior de Psicologia Aplicada, v. 8, p. 117–120, 1990. Citado na página 18.

MIRANDA, L. C. D.; SAMPAIO, F. F.; BORGES, J. A. dos S. Robofácil: Especificação e implementação de um kit de robótica para a realidade educacional brasileira. *Brazilian Journal of Computers in Education*, v. 18, n. 03, p. 46, 2011. Citado na página 14.

NETO, R. P. B. et al. Robótica na educação: uma revisão sistemática dos últimos 10 anos. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2015. v. 26, n. 1, p. 386. Citado na página 14.

PAPERT, S.; HAREL, I. Situating constructionism. *Constructionism*, v. 36, n. 2, p. 1–11, 1991. Citado na página 18.

PECORELLI, J. M. F. *Simulador de linguagem em texto estruturado para autómato TSX3721*. Tese (Doutorado), 2014. Citado 2 vezes nas páginas 23 e 24.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, Elsevier, v. 64, p. 1–18, 2015. Citado na página 29.

QUEIROZ, R. L.; SAMPAIO, F. F. Duinoblocks4kids: Um ambiente de programação em blocos para o ensino de conceitos básicos de programação a crianças do ensino fundamental i por meio da robótica educacional. *Anais do CSBC*, p. 2086–2095, 2016. Citado na página 21.

RESNICK, M. Behavior construction kits. *Communications of the ACM*, v. 36, n. 7, p. 64–71, 1993. Citado na página 18.

RESNICK, M.; KAFAI, Y.; MAEDA, J. A networked, media-rich programming environment to enhance technological fluency at after-school centers in economically-disadvantaged communities. 2005. Citado na página 21.

RESNICK, M. et al. Scratch: Programming for all. *Commun. Acm*, v. 52, n. 11, p. 60–67, 2009. Citado 2 vezes nas páginas 15 e 21.

RICARDO, J. et al. Ensino de matemática e a robótica educacional: uma proposta de investigação tecnológica na educação básica. *Rematec*, v. 1, n. 26, 2018. Citado na página 14.

SETHI, R.; ULLMAN, J. D.; LAM monica S. *Compiladores: princípios, técnicas e ferramentas*. [S.l.]: Pearson Addison Wesley, 2008. Citado 5 vezes nas páginas 20, 23, 24, 25 e 26.

SILVA, A. J. et al. Um modelo de baixo custo para aulas de robã³tica educativa usando a interface arduino. Universidade Federal de Alagoas, 2014. Citado 5 vezes nas páginas 18, 19, 20, 26 e 27.

STOPPA, M. H. A robótica educacional em experimentos elementares de física. *Instrumento-Revista de Estudo e Pesquisa em Educação*, v. 14, n. 1, 2012. Citado na página 14.

SULLIVAN, A.; ELKIN, M.; BERS, M. U. Kibo robot demo: engaging young children in programming and engineering. In: ACM. *Proceedings of the 14th international conference on interaction design and children*. [S.l.], 2015. p. 418–421. Citado na página 14.

ULLRICH, R. *Robótica–uma introdução. o porquê dos robôs e seu papel no trabalho*. [S.l.]: Editora Campus, 1987. Citado na página 18.

ZILLI, S. d. R. et al. A robótica educacional no ensino fundamental: perspectivas e prática. Florianópolis, SC, 2004. Citado 2 vezes nas páginas 18 e 19.

Apêndices

APÊNDICE A – ALGORITMOS EM *MAJESTIC LANGUAGE*

Código A.1 – Algoritmo de Ordenação de Vetor (*Insertion Sort*) em *Majestic Language*

```
a = { 34, −5, 6, 0, 12, 100, 56, 22, 44, −3, −9, 12, 17, 22, 6, 11 };
int sort (int n)
{
    t = 0;
    s = 0;
    temp = 0;
    while (t < n − 1)
    {
        s = t + 1;
        while (s < n)
        {
            if ( a[t] > a[s] ) {
                temp = a[t];
                a[t] = a[s];
                a[s] = temp;
            }
            s++;
        }
        t++;
    }
    return 0;
}

i = 0;
while (i < 16){
    print a[i];
    i++;
}

print "\n";
sort (16);
i = 0;
while(i < 16){
    print a[i];
    i++;
}
```

Código A.2 – Algoritmo de Programação Dinâmica (Bellman-Ford) em *Majestic Language*

```
INT_MAX = 9999;

graph = // edges (u,v,w)
 {{0, 1, -1}, {0, 2,  4}, {1, 2, 3},
  {1, 3,  2}, {1, 4,  2}, {3, 2, 5},
  {3, 1,  1}, {4, 3, -3}};

int bellmanford(int V, int E, int src) {
    dis = {INT_MAX : 5}; // Initialize distance of all vertices as 0.

     dis[src] = 0; // initialize distance of source as 0

    i = 0;                          // Relax all edges |V| - 1 times. A simple
    while(i < (V-1)) {              // shortest path from src to any other
          j = 0;                    // vertex can have at-most |V| - 1 edges
        while(j < E) {
            d = dis[graph[j][0]] + graph[j][2];
                if (d < dis[graph[j][1]])
                    dis[graph[j][1]] = dis[graph[j][0]] + graph[j][2];
            j++;
        }
        i++;
    }

    i = 0;                              // check for negative-weight cycles.
    while(i < E) {                      //The above step guarantees shortest
          x = graph[i][0];              //distances if graph doesn't contain
          y = graph[i][1];              //negative weight cycle. If we get a
          weight = graph[i][2];         //shorter path, then there is a cycle.
          if (dis[x] != INT_MAX and dis[x] + weight < dis[y])
              print "Graph contains negative weight cycle\n";
        i++;
    }

     print "Vertex Distance from Source\n";
     i = 0;
    while (i < V) {
          print i;
        print dis[i];
        print "\n";
        i++;
    }
    return 0;
}

bellmanford(5, 8, 0);
```

Código A.3 – Algoritmo de Recursividade (Fatorial) em *Majestic Language*

```
int fatorial (int n){
    if (n == 0)
        return 1;
    else
        return n * fatorial (n-1);
    return 1;
}
print fatorial (4);
```

Código A.4 – Algoritmo de Ordenação de Vetor (*Bubble Sort*) em *Majestic Language*

```
array = {92 , 24, 81:3, 2, 13, 832, 0};

int BubbleSort(int n){
    if (n < 1){
        return 0;
    }

    i = 0;
    while (i <= 8){
        if (array [i] > array [i+1]){
            aux = array[i];
            array[i] = array[i+1];
            array[i+1] = aux;
        }
        i++;
    }
    BubbleSort(n-1);
}

i = 0;
while (i <= 8)
{
    print array[i];
    i++;
}
print "\n";
BubbleSort(9);
i = 0;
while (i <= 9)
{
    print array[i];
    i++;
}
```

Código A.5 – Algoritmo de Multiplicação de Matrizes em *Majestic Language*

```
A = {{ 1.4, 32.3,  24.2}:3};
B = {{ 4.2, 31,  25}:3};
C = {{ 0:3}:3};

i  = 0;
j  = 0;
k = 0;
while(k<3)
{
    j  = 0;
    while(j<3)
    {
        temp = 0.0;
        i  = 0;
        while(i<3)
        {
            temp = temp + A[k][i]*B[i][j];
            i++;
        }
        C[k][j]  = temp;
        j++;
    }
    k++;
}

j  = 0;
k = 0;
while(k<3)
{
    j  = 0;
    while(j<3)
    {
        if (j!=2)
        {
            print  (C[i][j]);
        }
          else
          {
           print  (C[i][j]);
          }
        j++;
     }
    k++;
}
```

Código A.6 – Algoritmo de Recursividade (Potência) em *Majestic Language*

```
float pow (float num, int exp){
    if (exp == 1)
        return num;
    else
        return num * pow(num, exp-1);
    return 0;
}

print pow(2.2, 2);
```

Código A.7 – Algoritmo de Recursividade (Fibonacci) em *Majestic Language*

```
int fibonacci (int n){
    if (n == 1)
        return 1;
    else
        if (n == 2)
            return 1;
        else
            return fibonacci(n - 1) + fibonacci(n - 2);
    return 1;
}

i = 1;
while (i <= 5)
{
    print (fibonacci(i));
    i++;
}
```

Código A.8 – Algoritmo de Busca Elemento em Vetor em *Majestic Language*

```
vetor = {3, 2, 1, 5, 8, 4, 2, 6};

int posicao_vetor(int numero){
    k = 8;
    while (k >= 0 and vetor[k] != numero)
        k -= 1;
    if (k < 0)
        return -1;
    else
        return k;
}

print posicao_vetor(4);
```

Código A.9 – Algoritmo de Busca Elemento em Vetor em *Majestic Language*

```
void OPEN_GRIP() {
    Z = SERVO_POS_INC();
}

void CLOSE_GRIP() {
    Y = SERVO_POS_DEC();
}

void blink(float time){
    out13 = 255.0;
    delay time;
    out13 = 0.0;
    delay time;
}

blink(500.0);
blink(500.0);

while (true == true) {
    F = READ_IR();
    if (F == 3041546415.0) { // OPEN GRIP
        OPEN_GRIP();
    }
    else if (F == 3041579055.0) { // CLOSE GRIP
        CLOSE_GRIP();
    }
}
```