

Danilo Martins Rodrigues

Elaboração de uma Linguagem de Programação Específica para Robótica Educacional

Jataí-Goiás

2018

Danilo Martins Rodrigues

Elaboração de uma Linguagem de Programação Específica para Robótica Educacional

Monografia apresentada ao curso de Ciências da Computação da Universidade Federal de Goiás - Regional Jataí, como requisito parcial para obtenção do título de BACHAREL em Ciência da Computação.

Universidade Federal de Jataí - UFJ
Instituto de Ciências Exatas e Tecnológicas (ICET)
Bacharelado em Ciências da Computação

Orientador: Prof. Dr. Thiago Borges de Oliveira

Jataí-Goiás

2018

Danilo Martins Rodrigues

Elaboração de uma Linguagem de Programação Específica para Robótica Educacional/ Danilo Martins Rodrigues. – Jataí-Goiás, 2018-
64 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Thiago Borges de Oliveira

Monografia (Graduação) – Universidade Federal de Jataí - UFJ
Instituto de Ciências Exatas e Tecnológicas (ICET)
Bacharelado em Ciências da Computação, 2018.

1. LRE 2. Linguagem de Programação 3. Compilador 4. Robótica Educacional

Danilo Martins Rodrigues

Elaboração de uma Linguagem de Programação Específica para Robótica Educacional

Monografia apresentada ao curso de Ciências da Computação da Universidade Federal de Goiás - Regional Jataí, como requisito parcial para obtenção do título de BACHAREL em Ciência da Computação.

Trabalho aprovado. Jataí-Goiás, data da defesa:

Prof. Dr. Thiago Borges de Oliveira
Orientador

**Prof^a. Dra. Joslaine Cristina Jeske de
Freitas**
Avaliador

Prof^a. Ma. Franciny Medeiros Barreto
Avaliador

Jataí-Goiás
2018

Este trabalho é dedicado à Laura Verginassi, por me ensinar que é possível encontrar a felicidade mesmo nas horas mais sombrias, basta se lembrar de procurar pela luz.

AGRADECIMENTOS

Agradeço ao Professor Doutor Thiago de Oliveira Borges pelas orientações inspiradoras, por compartilhar ideias criativas e conhecimento para o desenvolvimento deste trabalho. A Professora Mestra Franciny Medeiros Barreto pelo apoio durante o estágio e incentivo a continuar buscando novos projetos. A Professora Doutora Ana Carolina Gondim Inocêncio por permitir minha participação em projetos tão gratificante quanto o TIC e o Winfo. A minha família por acreditar em mim e incentivar a não desistir. E principalmente para Laura Verginassi por estar sempre ao meu lado, ajudando e confortando nos momentos mais difíceis.

“Hoje a programação é uma corrida entre os engenheiros de software para tentar construir maiores e melhores programas à prova de idiotas, e o Universo tentando produzir maiores e melhores idiotas. Até agora, o Universo está ganhando.”
(Rich Cook)

RESUMO

Com o avanço da tecnologia e, como consequência, a facilidade do acesso a ela, a robótica vem ganhando espaço entre as novas metodologias de ensino atuais. Quando aplicada ao processo de ensino aprendizagem, a robótica passa a ser definida pelo termo Robótica Educacional, que consiste na aplicação de kits robóticos para o ensino. Contudo existem problemas com relação a escolha do kit robótico apropriado. Estes kits podem ser divididos em kits robóticos proprietários e kits robóticos *open-source*. Kits proprietários possuem sua própria linguagem de programação, porém, como são kits produzidos para o ensino possuem alto valor de aquisição e não permitem a livre alteração dos componentes limitado apenas aos componentes que fazem parte do próprio kit, dessa forma, também limitando sua linguagem de programação a estes. Kits *open-source* são baratos e abertos para a modificação, mas utilizam de linguagens de programação de propósito geral, comumente complexa para ser utilizada por usuários inexperientes para a programar seus protótipos. Nesse sentido esse trabalho propõe a linguagem de programação LRE (Linguagem para Robótica Educacional), voltada para kits *open-source*, estruturada em português, qual contem componentes das linguagens de programação específicas para a robótica, tais como eventos, e componentes das linguagens de programação de propósito geral, tais como funções, comandos para repetição de instruções e comparação de valores. A linguagem de programação LRE conta com o compilador LRE, dedicado para realizar a transformação do código desenvolvido pelo usuário da linguagem para código alvo reconhecido pela plataforma de prototipação alvo. O compilador LRE consiste em um *frontend* baseado no LLVM integrado ao *backend* GCC experimental para microcontroladores AVR. O compilador LRE é mais rápido que ambiente de desenvolvimento Arduino IDE, gerando códigos equivalentes em até 7 vezes menos tempo. O código alvo gerado pelo compilador LRE é 149% maior que um código equivalente gerado pelo Arduino IDE, valor considerado aceitável devido a utilização do *backend* LLVM experimental, ao contrário da já consolidada plataforma GCC usada na Arduino IDE.

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

ABSTRACT

Considering the advancement of technology and, as a consequence, the facility in the access to it, robotics has been gaining ground among the new teaching methodologies nowadays. When applied to the teaching methodology process, robotics is defined by the term Educational Robotics, which consists of the application of robotic kits in the educational process. However there are problems in choosing the appropriate robotic kit. Those kits can be divided into proprietary robotic kits and *open-source* robotic kits. Proprietary kits have their own programming language, however, as they are manufactured kits made for teaching, they have a high acquisition value and do not allow the components to be changed, so their programming language is limited to its own components. *Open-source* robotic kits are less expensive and open for modification, but use general-purpose programming language, considered complex to be used by inexperienced users to program their own prototypes. In this sense, this research proposes the LRE (Linguagem para Robótica Educacional) programming language for *open-source* kits, structured in portuguese and contains components of programming languages specific to robotics, such as event programming, and components of general purpose programming languages, such as functions and commands for comparing values or repeating instructions. The developed programming language relies on a dedicated compiler, named LRE, to perform the transformation of user-developed language code into target code recognized by the prototyping platform. The LRE compiler consists in a *frontend* based in LLVM, integrated to a experimental GCC *backend* for AVR microcontrollers. The LRE compiler is faster than the development environment Arduino IDE, generating equivalent codes up to 17 times faster. The target code generated by the LRE compiler is up to 149% bigger than a similar code generated by Arduino IDE, this value is considerable acceptable, since the LRE uses a LLVM experimental *backend*, instead of the consolidated GCC platform used by Arduino IDE.

Key-words: *LRE; Programming Language; Compiler; Educacional Robotic.*

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de programa desenvolvido em DB4K	21
Figura 2 – Exemplo de programa desenvolvido em NXT	21
Figura 3 – Programação para Arduino	22
Figura 4 – Kit de robótica educacional Lego MindStorms	25
Figura 5 – Kit de robótica <i>open-source</i>	25
Figura 6 – Placa de prototipação Arduino Uno	26
Figura 7 – Microcontrolador PIC	27
Figura 8 – Sensor Passivo para Arduino	28
Figura 9 – Sensor Ativo para Arduino	28
Figura 10 – Atuador	28
Figura 11 – Fluxograma apresentando as etapas do <i>frontend</i>	30
Figura 12 – Processo de Compilação	45
Figura 13 – Esquema do Circuito Arduino	47

LISTA DE TABELAS

Tabela 1 – Comparativo entre trabalhos	38
Tabela 2 – Operadores aceitos pela linguagem LRE	40
Tabela 3 – Médias de Tempo de Compilação	52
Tabela 4 – Médias de Tamanho do Código Alvo	53

SUMÁRIO

1	Introdução	14
	INTRODUÇÃO	14
2	Referencial Teórico	18
2.1	Robótica Educacional	18
2.2	Programação na Robótica Educacional	19
2.2.1	Programação Gráfica	20
2.2.2	Programação Textual	21
2.2.3	IDEs	22
2.2.4	Principais Paradigmas	23
2.2.5	Linguagens Orientadas a Eventos	23
2.2.6	Linguagens Tipadas	23
2.3	Hardware Utilizado na Robótica Educacional	24
2.3.1	Microcontroladores	24
2.3.1.1	AVR	25
2.3.1.2	PIC	26
2.3.2	Sensores e Atuadores	27
2.4	Compilador	28
2.4.1	Frontend	29
2.4.1.1	Analisador Léxico	29
2.4.1.2	Analisador Sintático	31
2.4.1.3	Analisador Semântico	32
2.4.1.4	Gerador de Código Intermediário	32
2.4.2	Backend	32
2.5	Considerações Finais	33
3	Trabalhos relacionados	34
3.1	Introdução	34
3.2	Critérios de Busca	34
3.3	Trabalhos Analisados	34
3.3.1	Using visual programming kit and Lego Mindstorms: An introduction to embedded system (T1)	35
3.3.2	Guarátca: uma poderosa biblioteca de funções para os robôs baseados em Arduino (T2)	35
3.3.3	Brasilino: biblioteca para Arduino que permite o desenvolvimento de projetos em português do Brasil (T3)	36

3.3.4	DuinoBlocks for Kids: 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 (T4)	36
3.4	Resumo Comparativo	37
3.5	Considerações Finais	37
4	Linguagem de Programação LRE	39
4.1	Introdução	39
4.2	Linguagem de Programação LRE	39
4.2.1	Operadores	40
4.2.2	Comando de Exibição	40
4.2.3	Tipos de Dados	41
4.2.4	Estruturas de Decisão	41
4.2.5	Estruturas de Repetição	41
4.2.6	Funções	42
4.2.7	Utilização de Portas	42
4.2.8	Eventos	42
4.3	Compilador LRE	44
4.3.1	Construção do Compilador	44
4.3.2	Processo de Compilação	44
4.4	Testes	46
4.4.1	Materiais Utilizados	46
4.4.2	Teste com Interrupção Externa	47
4.4.3	Teste com Interrupção de Tempo	48
4.5	Considerações Finais	49
5	Experimentos Realizados	51
5.1	Introdução	51
5.2	Experimentos com Tempo de Compilação	51
5.3	Experimentos com Tamanho de Arquivo	52
5.4	Considerações finais	53
6	Conclusões e Trabalhos Futuros	55
6.1	Introdução	55
6.2	Conclusões	55
6.3	Trabalhos futuros	56
	Referências	57
	Anexos	59
	ANEXO A Instalação do LLVM no sistema operacional Ubuntu	60

ANEXO B	Códigos de Interrupção Externa	61
ANEXO C	Códigos de Interrupção de Tempo	63

1 INTRODUÇÃO

A robótica está se tornando uma ferramenta educacional muito popular nas áreas de ciência e tecnologia em escolas de ensino fundamental e médio e em diversas áreas da engenharia em universidades (RODRIGUEZ; CUESTA, 2016). O termo Robótica Educacional é frequentemente associado a utilização de kits robóticos no processo de ensino-aprendizagem, introduzindo conceitos complexos já no início da formação dos estudantes, como robótica básica, eletrônica, mecânica aplicada e programação de computadores (MAJOR; KYRIACOU; BRERETON, 2012). Os kits robóticos utilizados na Robótica Educacional podem ser divididos em kits proprietários ou kits *open-source*, esses kits são compostos por peças estruturais, como chassis e engrenagens, atuadores, sensores, microcontrolador, baterias e um conjunto de *software* composto por um ambiente de desenvolvimento e linguagem de programação para a programação dos microcontroladores (FORNAZA; WEBBER; VILLASBOAS, 2015).

Os kits proprietários de robótica são produzidos por diversas empresas e possuem o foco no ensino-aprendizagem de conceitos relacionados a tecnologia, engenharia e matemática (FORNAZA; WEBBER; VILLASBOAS, 2015). Para a programação, estes kits, possuem um ambiente de desenvolvimento e uma linguagem de programação proprietária com o objetivo facilitar a programação das ações do robô (SILVA, 2014). Alguns kits robóticos educacionais disponíveis no mercado são: Lego Mindstorms, Edison robot, Scribbler2, E-puck robot, Wskits, entre outros (MONDADA; BONANI; RIEDO, 2017).

O kit de robótica Lego Mindstorms¹ é frequentemente utilizado em pesquisas relacionadas a Robótica Educacional (RODRIGUEZ; CUESTA, 2016), sendo esse, uma linha do brinquedo LegoTM lançada comercialmente em 1998 e voltada para a educação tecnológica. Atualmente a versão Mindstorms EV3 é versátil e suas peças possibilitam a montagem de robôs diferentes (MONDADA; BONANI; RIEDO, 2017). Porém, esse kit apresenta um preço muito elevado, principalmente para ser adquirido em grande escala por instituições públicas e sua aplicabilidade é de certa forma limitada quando é usada a sua linguagem de programação de blocos própria (RODRIGUEZ; CUESTA, 2016).

Um exemplo deste problema: a linguagem utilizada para a programação dos robôs Lego não contempla a utilização de estruturas de escolha tais como *IF* da Linguagem C e cálculos complexos. Isso reduz as possibilidades de aplicação desse robô para áreas do conhecimento externas a engenharia e programação, diminuindo assim suas aplicações para Robótica Educacional. O robô também permite a utilização de linguagens de programação alternativas, porém, quando utilizada linguagens como C para a Robótica

¹ <https://www.lego.com/en-us/mindstorms>

Educacional diminui o interesse dos usuários, principalmente crianças, devido à dificuldade de compreensão sintática (QUEIROZ; SAMPAIO; SANTOS, 2016).

Devido ao preço acessível e diversas possibilidades de acrescentar componentes externos, kits robóticos *open-source* tem ganhado espaço na Robótica Educacional (RODRIGUEZ; CUESTA, 2016). Para estes kits é possível a integração de componentes externos, tais como sensores, atuadores, peças e microcontrolador. Ao contrário dos kits educacionais de robótica que podem ser adquiridos apenas em conjuntos completos e suas especificações são restritas aos desenvolvedores, os kits *open-source* possuem especificações abertas e portanto permitem ao usuário modelar e programar livremente o protótipo robótico (FERDOUSH; LI, 2014).

Contudo, a linguagens de programação utilizadas pelos usuários em kits *open-source* consistem em linguagens de propósito geral adaptadas para Robótica Educacional (QUEIROZ; SAMPAIO; SANTOS, 2016). Dessa forma, a linguagem utilizada não possui meios para abstrair a complexidade sintática presente em uma linguagem de baixo nível tornando a Robótica Educacional menos atrativa (QUEIROZ; SAMPAIO; SANTOS, 2016; LOPES, 2017).

Para reduzir a complexidade presente na programação de kits *open-source*, foram projetados ambientes de desenvolvimento. Esses ambientes têm como objetivo fornecer suporte para o usuário desenvolver seus programas e compilá-los para uma linguagem de baixo nível compatível com o microcontrolador (JUNIOR; GUEDES, 2015).

A maioria dos kits robóticos educacionais possui um microcontrolador, o qual é responsável pela integração e ativação dos componentes, como por exemplo, processamento dos sinais dos sensores, ativação dos atuadores, dentre outros. Os microcontroladores mais comuns são PIC da Microchips, AVR da Atmel e Advantech PCM (SILVA, 2014). Alguns microcontroladores são integrados em plataformas para facilitar sua utilização, alguns exemplos de plataformas são o Arduino e o Raspberry PI (FERDOUSH; LI, 2014).

O Arduino² é uma plataforma que integra o microcontrolador AVR da Atmel com conexões macho ou fêmea que facilitam o acesso a portas de entrada e saída (SILVA, 2014). Arduino é um projeto de *hardware open-source* desenvolvido por um grupo de 5 pesquisadores no Interaction Design Institute Ivrea³, que tem por objetivo a criação de uma plataforma de prototipagem eletrônica de código aberto, simples de ser utilizada. O seu baixo custo, qualidade e flexibilidade, têm feito desta plataforma uma excelente opção para o desenvolvimento de projetos na área de Robótica Educacional (MAHMUD, 2017). Porém, as aplicações desenvolvidas para a plataforma Arduino, assim como demais kits *open-source*, utilizam de linguagens de programação de propósito geral, geralmente a linguagem C, adaptada para robótica.

² <https://www.arduino.cc/>

³ <http://www.interaction-ivrea.com/>

Um exemplo desse problema pode ser descrito como: imagine a programação de um protótipo de braço robótico que utiliza Arduino. Para movimentar o braço é necessário movimentar os motores de acordo com o ângulo que deseja girá-lo. Para definir o ângulo utilizando a linguagem de programação C, é necessário ligar o motor responsável pelo movimento por determinado tempo para que seja possível a movimentação em sentido horário e anti-horário. Outros movimentos como mover o braço na vertical necessitariam novas funções semelhantes, assim, aumentando a complexidade e quantidade de linhas de código. Ao utilizar uma linguagem própria para a Robótica Educacional como, por exemplo a NXT, o usuário apenas utilizaria blocos de movimento do braço, os quais necessitam apenas da especificação do ângulo e motor.

De acordo com [Lopes \(2017\)](#), esses fatores tornam viável a utilização de uma linguagem de programação de propósito específico na Robótica Educacional que reduza a diferença entre a linguagem natural do usuário e a linguagem usada para programar o robô. [Gomes et al. \(2016\)](#) defendem que para facilitar a utilização de protótipos robóticos na Robótica Educacional é necessário uma linguagem de programação padronizada e em português. Para [Rodriguez e Cuesta \(2016\)](#), a linguagem de programação utilizada na Robótica Educacional deve ser expressiva possuindo poucas limitações. Dessa forma torna-se viável a elaboração de uma linguagem de programação para a Robótica Educacional que seja expressiva e de complexidade intermediária para os usuários.

Visando este cenário, este trabalho propôs uma linguagem de programação textual, escrita em português e especificada para a Robótica Educacional, contendo elementos provenientes de linguagens de propósito geral e elementos próprios para a programação de protótipos robóticos. Para o desenvolvimento da linguagem de programação proposta, foi necessário a realização dos seguintes objetivos: elencar os principais elementos presentes em linguagens de programação, o desenvolvimento de um *frontend* compilador para a linguagem e a integração a um *backend* existente.

A contribuição principal deste trabalho é uma linguagem de programação chamada LRE (Linguagem para Robótica Educacional), que consiste em uma linguagem de programação textual, em português e conta com elementos presentes em linguagens de programação de propósito geral e linguagens específicas para a robótica. Outra contribuição deste trabalho consiste no compilador LRE qual compila mais rápido que o ambiente de desenvolvimento Arduino IDE, gerando códigos equivalentes em tamanho aceitável.

O presente trabalho está organizado da seguinte maneira: o Capítulo 2, Referencial Teórico, apresenta conceitos importantes para o entendimento da pesquisa. O Capítulo 3, Trabalhos Relacionados, apresentam trabalhos selecionados que são a área de pesquisa deste trabalho. O Capítulo 4, Linguagem de Programação LRE, apresenta a metodologia utilizada para a construção da linguagem de programação LRE e compilador LRE, juntamente com os resultados esperados dos experimentos. O Capítulo 5, Experimentos, detalha

os experimentos realizados com a linguagem e compilador propostos em comparação com a linguagem de programação C++. Por fim, o Capítulo 6, Conclusões, apresenta as conclusões sobre o presente trabalho, juntamente com propostas para futuros trabalhos relacionados a este.

2 REFERENCIAL TEÓRICO

Para a compreensão desse projeto de pesquisa, são apresentados 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 deste capítulo.

2.1 Robótica Educacional

Seymour Papert, matemático que lecionava no Massachusetts Institute of Technology - MIT, é considerado o precursor da Robótica Educacional por meio da teoria do construcionismo (JUNIOR; GUEDES, 2015). O construcionismo é associado a construção de objetos físicos para a representação do conhecimento. Segundo ele, a educação deve ser exploratória e oferecer aos alunos a oportunidade de experimentar aplicações e atividades relacionadas aos assuntos (JUNIOR; GUEDES, 2015). Nesse contexto, a Robótica Educacional pode ser introduzida como um ambiente formado pelo computador, componentes eletrônicos, mecânicos e *software*, onde o aluno pode integrar esses elementos para construir protótipos e testar conceitos provenientes de outras áreas de ensino (SILVA, 2014).

Em meados dos anos 80 era utilizado uma tartaruga gráfica, um robô pronto para responder aos comandos do usuário que poderia ser programada através da linguagem de programação LOGO, desenvolvido para realizar as atividades da Robótica Educacional. Em conjunto com o MIT, Papert propôs a construção de engrenagens, motores, peças, polias, sensores, para que fosse possível ceder movimento e programar comportamentos dos protótipos desenvolvidos pelos estudantes (FRANGO et al., 2008). Com o avanço da tecnologia e facilidade de acesso, a Robótica Educacional começou a ser difundida em diversos países (JUNIOR; GUEDES, 2015).

Holanda e Alemanha utilizam a robótica como ferramenta em todas as escolas públicas e apresentam melhorias no desempenho dos alunos em todos os níveis de ensino (SILVA, 2014). Outros países como Espanha, França, Romênia, Grécia e Itália mostram aumento significativo da quantidade de projetos de pesquisa relacionados a Robótica Educacional visando a melhoria do processo de ensino-aprendizagem (FRANGO et al., 2008). No Brasil não existem estatísticas nacionais de quantas escolas possuem aulas de robótica, mas é perceptível o aumento do número de projetos de robótica e quantidade de

participantes em competições de robótica. Segundo [Silva \(2014\)](#), há um perceptível aumento da busca e implementação de projetos de robótica por escolas públicas e particulares.

Para facilitar a utilização da robótica educacional são comercializados kits de robótica ([JUNIOR; GUEDES, 2015](#)). Esses kits são compostos por peças estruturais, como chassis e engrenagens, atuadores, sensores, microcontrolador, baterias atuadores, sensores, microcontrolador, baterias e um conjunto de *software* composto por um ambiente de desenvolvimento e linguagem de programação para a programação dos protótipos ([FORNAZA; WEBBER; VILLASBOAS, 2015](#)).

2.2 Programação na Robótica Educacional

Linguagem de programação é um método padronizado de instruções utilizadas para a comunicação do homem com o computador, através de um conjunto de regras. Por meio das linguagens de programação é possível especificar instruções para o armazenamento de informações e execução de algoritmos que representem operações ou resolvam problemas. Portanto, todo *software* necessita de uma linguagem de programação para especificá-lo ([SILVA, 2014](#)).

É característico, portanto, na Robótica Educacional, a junção da linguagem de programação com as peças físicas que possibilitem a programação de ações ou rotinas ([FRANGOU et al., 2008](#)). Nesse sentido, a linguagem de programação é imprescindível para a Robótica Educacional, pois proporciona ao usuário a possibilidade de programar ações para seus protótipos ([SILVA, 2014](#)).

De forma geral, é possível dividir as linguagens de programação em duas classes: linguagens de propósito específico e linguagens de propósito geral. Linguagens de propósito específico são linguagens de programação que oferecem, por meio de comandos e funções restritos, expressividade focadas em um domínio específico. Linguagens de propósito geral são linguagens com grande poder expressivo, porém fornecem soluções aceitáveis para qualquer domínio ([DEURSEN; KLINT; VISSER, 2000](#)). Em kits de robótica proprietários é comum a utilização de linguagens de propósito específico, enquanto em kits *open-source* é comum a utilização de linguagens de propósito geral ([RODRIGUEZ; CUESTA, 2016](#)).

As primeiras aplicações da robótica educacional datam da década de 80, onde tartarugas robóticas eram programadas utilizando a linguagem LOGO. Posteriormente, as primeiras linguagens de programação visuais foram surgindo como, por exemplo, a NXT utilizada em kits de robótica proprietários da Lego ([FRANGOU et al., 2008](#)). Com o crescente número de kits robóticos sendo comercializados, diversas novas linguagens de programação foram surgindo ou sendo adaptadas de linguagens de propósito geral ([JUNIOR; GUEDES, 2015](#)).

As linguagens de programação utilizadas na Robótica Educacional podem ser classificadas em dois tipos: linguagem de programação gráfica, geralmente de propósito específico e linguagem de programação textual, de propósito geral. A programação gráfica consiste em utilizar blocos desenhados encaixáveis para programar ações, enquanto a programação textual consiste em códigos digitados assim como as linguagens de programação C e Java (JUNIOR; GUEDES, 2015).

Para desenvolver aplicações para os protótipos é possível a utilização de ambientes de desenvolvimento - IDE, que fornecem suporte para o usuário escrever ou diagramar e compilar a aplicação em determinada linguagem (JUNIOR; GUEDES, 2015). A IDE, geralmente, é desenvolvida em uma linguagem de programação diferente da qual ela fornece suporte.

2.2.1 Programação Gráfica

Devido a dificuldade de compreensão da sintaxe das linguagens de programação existentes no final dos anos 70, começaram a surgir diversas iniciativas para desenvolver linguagens de programação alternativas com sintaxes mais simples. Uma das alternativas encontradas foi a utilização das VLP - *Visual Programming Language* ou linguagem de programação visual. As linguagens desenvolvidas nesse modelo apresentavam sintaxes contidas em expressões visuais de diferentes formatos (QUEIROZ; SAMPAIO; SANTOS, 2016).

Posteriormente foram iniciados projetos implementando VLPs na Robótica Educacional. Os resultados dessa implementação demonstravam que diversos usuários poderiam desenvolver rotinas simplesmente arrastando e soltando componentes na IDE (MAHMUD, 2017). Diversas linguagens de programação surgiram com esse propósito, tais como:

- DB4K: consiste em blocos com expressões sintáticas em português que são montados para realizar determinada ação. Essa linguagem é feita para funcionar na placa de prototipação Arduino (QUEIROZ; SAMPAIO; SANTOS, 2016). Um exemplo dessa linguagem pode ser visto na [Figura 1](#).
- NXT-G: a linguagem NXT-G é proprietária da empresa LEGO e acompanha o kit Lego MindStorms. Essa linguagem apresenta blocos contendo desenhos ou cálculos que são montados para realizar ações (FORNAZA; WEBBER; VILLASBOAS, 2015). Um exemplo de programação em NXT-G pode ser visto na [Figura 2](#).

Contudo, VLPs apresentam baixa expressividade, pois são restritivas apenas a comando de blocos e necessitam que usuários tenham conhecimento sobre estruturas, de forma que, não é possível realizar determinadas atividades e seja complexa para usuários pouco expe-



Figura 1 – Exemplo de programa desenvolvido na linguagem gráfica DB4K, no ambiente DB4k onde as diferentes cores, na divisória esquerda, representam diferentes tipos de ações. À direita o usuário arrasta e insere os blocos em sequência ou sobre outro bloco, como parâmetro, para desenvolver o algoritmo. Retirado de (QUEIROZ; SAMPAIO; SANTOS, 2016).

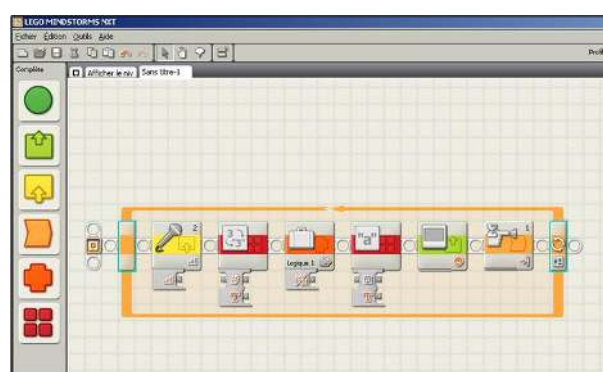


Figura 2 – Exemplo de programa desenvolvido em NXT. À esquerda são disponibilizados blocos coloridos, os quais o usuário deve arrastar e soltar no painel quadriculado à direita em sequência para formar um algoritmo (FORNAZA; WEBBER; VILLASBOAS, 2015).

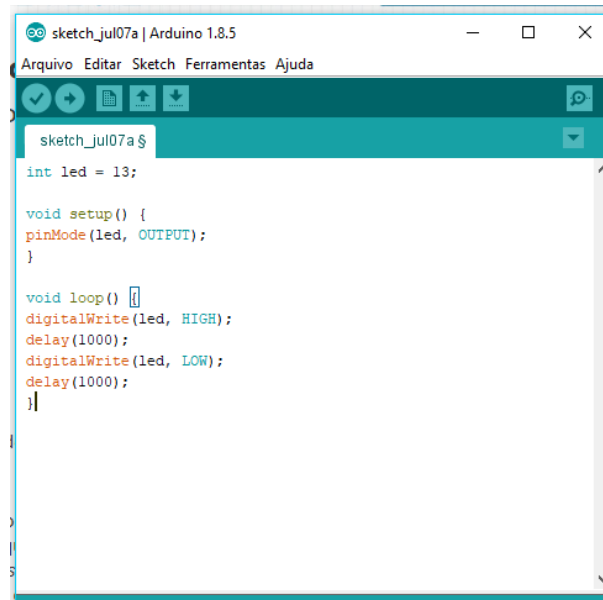
rientes limitando assim as possíveis aplicações (RODRIGUEZ; CUESTA, 2016; BRAVO; GONZÁLEZ; GONZÁLEZ, 2017).

2.2.2 Programação Textual

Programação textual refere-se a linguagens de programação de forma escrita para o desenvolvimento de funções que serão realizadas pelo protótipo. De forma geral, esse tipo de linguagem é o mais comum e amplamente difundido, com bibliotecas bem definidas, tais como C ou Java (JUNIOR; GUEDES, 2015) demonstrados na Figura 3. Essas linguagens e suas bibliotecas, em grande maioria, são escritas em C (GOMES et al., 2016).

As linguagens textuais utilizadas na Robótica Educacional são adaptações das linguagens de propósito geral. Essa adaptação conserva os comandos da linguagem de

propósito geral, acrescentando apenas especificações das portas nas quais o *hardware* será conectado. Esse tipo de programação apresenta sintaxes complexas, deixando projetos de robótica menos atrativos, principalmente para crianças (QUEIROZ; SAMPAIO; SANTOS, 2016).

A screenshot of the Arduino IDE interface. The window title is "sketch_jul07a | Arduino 1.8.5". The menu bar includes "Arquivo", "Editar", "Sketch", "Ferramentas", and "Ajuda". The toolbar shows icons for file operations and execution. The main editor area contains the following C code:

```
sketch_jul07a $
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Figura 3 – Exemplo de programa desenvolvido em C no Ambiente de Desenvolvimento Arduino IDE

2.2.3 IDEs

IDE - *Integrated Development Environment* ou ambiente de desenvolvimento integrado é um *software* que possibilita o desenvolvimento e execução de programas em uma determinada linguagem (JUNIOR; GUEDES, 2015). As IDEs são desenvolvidas para atenderem as necessidades de programadores profissionais, possuindo diversas características e elementos integrados (LOPES, 2017).

Algumas IDEs disponíveis atualmente podem depurar o programa desenvolvido pelo usuário em tempo de execução, de forma que seja possível a identificação imediata de erros por meio de mensagens de compilação. Outras funcionalidades como a execução de linhas individuais e a adição de *breakpoints* facilitam o processo de entendimento e correção de erros dos programas desenvolvidos (LOPES, 2017). Existem IDEs de programação gráfica e programação textual disponíveis para kits robóticos independentemente de serem *open-source* ou não (JUNIOR; GUEDES, 2015).

Apesar de sua relevância, neste projeto não será considerada a construção de uma IDE para a linguagem proposta, devido a redução de escopo. Este ponto deve ser considerado em trabalhos futuros.

2.2.4 Principais Paradigmas

Dentre as linguagens de contexto geral adaptados para a Robótica Educacional, destacam-se os paradigmas estruturado e, menos utilizado, o paradigma orientado a objeto (BRAVO; GONZÁLEZ; GONZÁLEZ, 2017).

O paradigma estruturado recomenda que todos os programas podem ser reduzidos a estruturas de sequência, decisão e interação. Também orienta ao desenvolvimento de estruturas simples nos programas utilizando funções e rotinas (JUNGTHON; GOULART, 2016). Esse paradigma é predominante na Robótica Educacional, contemplando as linguagens gráficas e a grande maioria das linguagens textuais (BRAVO; GONZÁLEZ; GONZÁLEZ, 2017).

O paradigma orientado a objeto é baseado na interação de unidades de *software* denominados objetos. O funcionamento das linguagens orientadas a objetos consiste na troca de mensagens entre esses. Os objetos são classes que possuem métodos e atributos (JUNGTHON; GOULART, 2016). Na Robótica Educacional não é comum a utilização de linguagens orientadas a objeto (BRAVO; GONZÁLEZ; GONZÁLEZ, 2017).

2.2.5 Linguagens Orientadas a Eventos

Um evento é uma condição detectada que dispara uma notificação, essa notificação é enviada e processada em tempo de execução para que haja uma ação. Os eventos podem ser classificados em três formas: evento de tempo, o qual ocorre ao transcorrer do tempo, evento físico que ocorre por ocorrências no ambiente externo e mudança de estado o qual ocorre quando um valor de variável é alterada (XAVIER, 2014).

Na Robótica Educacional a linguagem orientada a evento, ou linguagem dirigida a evento, é utilizada em todos os paradigmas e classificações de linguagens de programação pois, por meio dela, é possível programar rotinas em que o robô tenha um conjunto de ações para executá-las quando houver interações do usuário, ambiente ou do próprio programa (XAVIER, 2014).

2.2.6 Linguagens Tipadas

Linguagens tipadas ou tipificadas são linguagens de programação que permitem a atribuição de um tipo, seja inteiro, real ou caractere, a uma variável com intuito de alocar memória para essa e, posteriormente, realizar operações entre elas.

- Linguagens fortemente tipada: consiste em linguagens que exigem a declaração do tipo de variável pelo usuário antes da execução do programa. A variável permanece no mesmo tipo declarado até o final da execução.

- Linguagens fracamente tipadas ou dinamicamente tipadas: consiste em linguagens que atribuem o tipo de variável em tempo de execução do programa, essa atribuição é realizada pelo compilador. O tipo da variável pode ser alterado durante a execução.
- Linguagens não tipadas: consiste em linguagens que não possuem tipos de dados.

2.3 Hardware Utilizado na Robótica Educacional

A robótica faz o uso integrado de sistemas computacionais e dispositivos elétricos e mecânicos para obter informações e atuar em um determinado ambiente. Os dispositivos elétricos e mecânicos consistem na parte física, o *hardware* do protótipo, um elemento fundamental de qualquer tipo de kit robótico disponível para aquisição (SILVA, 2014).

Kits proprietários de robótica, Figura 4, são comercializados em conjunto completos de forma que a inserção de novas peças de *hardware* é limitada, porém, estes apresentam diversas funcionalidades em apenas um kit (FORNAZA; WEBBER; VILLAS-BOAS, 2015). Em comparação, kits *open-source*, Figura 5, possibilitam a utilização de estruturas de materiais reciclados e aquisição de componentes de *hardware* independente de conjunto, proporcionando a possibilidade de modificação da estrutura e inserção de novas funcionalidades (MONDADA; BONANI; RIEDO, 2017).

Ambos os kits de robótica utilizam a mesma estrutura básica, composta por:

- Microcontrolador: responsável por receber o programa do usuário e executar as instruções.
- Estrutura: consiste na parte externa do protótipo, tais como chassi, rodas, entre outras peças mecânicas.
- Eletrônicos: permitem o protótipo coletar informações do ambiente por meio de sensores e realizar ações por meio de atuadores (JUNIOR; GUEDES, 2015).

2.3.1 Microcontroladores

Microcontrolador é a parte do *hardware* responsável por executar as instruções recebidas. Para isso microcontroladores contam com uma central de processamento e memória interna, onde o programa desenvolvido pelo usuário é armazenado para ser executado posteriormente (MARTINS, 2005). Para realizar ações em um protótipo, o microcontrolador é embutido a placas de prototipação que possuem dispositivos de entrada e meios de saída (SILVA, 2014).

O microcontrolador necessita de uma interface externa para que possa ser programado. Essa interface permite que o usuário desenvolva programas em um computador



Figura 4 – Ilustração do kit de robótica proprietário Lego Mindstorms EV3. Retirado de (JUNIOR; GUEDES, 2015).



Figura 5 – Kit de robótica *open-source* com placa de prototipação Arduino UNO. Retirado de (ROBOCORE, 2018).

externo e os transfira diretamente para o microcontrolador. Nesse sentido, kits de robótica, independente de tipo, necessitam de uma interface não embarcada para receber instruções de computadores externos por meio de cabos ou conexões sem fio (SILVA, 2014).

A comercialização dos microcontroladores varia de acordo com o kit adquirido. Em kits de robótica proprietário o microcontrolador faz parte do conjunto de peças referentes ao seu kit, não sendo vendido separadamente. Kits robóticos *open-source* podem possuir diferentes microcontroladores, sendo os mais comuns o AVR e o PIC (MONDADA; BONANI; RIEDO, 2017; SILVA, 2014).

O modelo de processador é importante na concepção de uma linguagem porque o conjunto básico de instruções do mesmo muda de acordo com o fabricante ou modelo. Nas sub-seções a seguir serão destacadas as particularidades destes dois modelos principais, que serão considerados no desenvolvimento deste projeto.

2.3.1.1 AVR

O AVR é um microcontrolador desenvolvido pela fabricante ATMEL. Sua arquitetura separa a memória de dados da memória de programa. Essa separação fornece maior velocidade de processamento de dados do programa. O AVR é comercializado de forma embutida em placas de prototipagem Arduino (SILVA, 2014).

Arduino, demonstrado na Figura 6, é uma plataforma de prototipagem eletrônica de *hardware open-source*, projetada com o microcontrolador AVR em conjunto com conexões macho ou fêmea que facilitam o acesso às portas de entrada e saída do microprocessa-

dor (FORNAZA; WEBBER; VILLASBOAS, 2015). Existem diversos modelos de Arduino disponíveis no mercado. De forma geral, é comum entre eles o microcontrolador, linhas de entrada e saída digital e analógica e uma interface serial ou USB. Para programar nessa plataforma é possível utilizar o Arduino IDE, uma interface de desenvolvimento escrita em Java em conjunto com sua linguagem de programação C e C++ e bibliotecas prontas escritas em C e C++ (SILVA, 2014).

O microcontrolador AVR possui quatro grupos de instruções, sendo essas instruções aritméticas ou lógicas, instruções de transferência de dados, instruções de desvio de programa, e instruções de controle interno do microcontrolador. Inicialmente todos os dados são transferidos para o núcleo do microcontrolador para serem processados e posteriormente armazenados na memória. Microcontroladores AVR são fabricados para atender aplicações específicas e, portanto, são amplamente utilizadas em plataformas de prototipação.

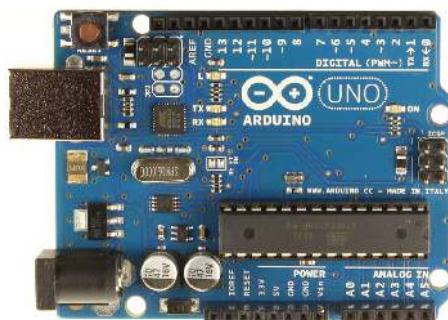


Figura 6 – Placa de prototipação Arduino Uno com microcontrolador ATmega328, chip comprido na parte inferior a direita, 32KB de memória Flash e 16MHz de velocidade de clock. Entrada USB, quadrado superior a esquerda, conexões no canto inferior a direita. Retirado de (JUNIOR; GUEDES, 2015).

2.3.1.2 PIC

Microcontroladores PIC são fabricados pela empresa Microchip. Assim como a arquitetura do microcontrolador AVR, o PIC separa a memória de dados da memória de programa (MARTINS, 2005).

O microcontrolador PIC, devido sua arquitetura, pode executar instruções de busca e execução em apenas um ciclo de máquina. Os barramentos do microcontrolador são otimizados para palavras de comprimento único, caso o barramento seja de 8 bits e haja duas palavras, sendo duas de 4 bits serão necessários dois ciclos de máquina para o processamento, o que pode ser considerado uma desvantagem em relação ao microcontrolador AVR (MARTINS, 2005).

As instruções do microcontrolador PIC podem ser divididas em três grupos, sendo elas: orientadas a *bit*, em que operam com *bits* de um determinado registrador, orientadas

a *byte*, operam com a memória completa de um registrador, e controle que consistem em instruções que usam números e endereços (MARTINS, 2005).



Figura 7 – Microcontrolador PIC 16F628A com 18 pinos de entrada/saída, 20 MHz de velocidade de clock e 14KB de memória. Retirado de (ROBOCORE, 2018).

2.3.2 Sensores e Atuadores

Sensores são dispositivos que têm como objetivo coletar e converter informações captadas no ambiente em sinais elétricos que podem ser interpretados. Sensores podem ser classificados de acordo com sua atuação em dois tipos: ativos e passivos. Sensores ativos são sensores que possuem sua própria fonte de radiação eletromagnética, tais como radares, *lasers*, entre outros (SILVA, 2014). Sensores passivos apenas medem a radiação emitida pelo ambiente, alguns exemplos são: sensores de luz e câmeras (SILVA, 2014).

A Figura 8 apresenta um modelo de sensor passivo, que capta a luminosidade do ambiente alterando a resistência de seus terminais de acordo com a luminosidade. A Figura 9 apresenta um modelo de sensor ativo, que emite ondas ultrassônicas que refletem em objetos posicionados a sua frente, ao retornar a onda, o sensor interpreta a informação fornecendo a sua distância ao objeto.

Atuadores são componentes que realizam a conversão de energia proveniente de baterias, tomadas e líquidos em energia mecânica. Essa energia é transmitida pelos atuadores para que peças móveis produzam movimento (SILVA, 2014). A Figura 10 mostra um exemplo de atuador (JUNIOR; GUEDES, 2015).

Sensores e atuadores permitem ao robô a interação direta com o ambiente, sendo capazes de lidar com diversas situações distintas em tempo real. Para que isso seja possível, a linguagem de programação utilizada deve ser capaz lidar com dados provenientes de um ou mais sensores, ativando ou desativando atuadores de forma simultânea quando necessário para garantir o funcionamento do robô (SILVA, 2014).



Figura 8 – Sensor passivo LDR que capta a luminosidade no local. Retirado de ([ARDUINOLANDIA, 2018](#)).



Figura 9 – Sensor ativo HCSR04 medidor de distância. Calcula a distância a outros objetos por meio do envio de ondas ultrassônicas. Retirado de ([ROBOCORE, 2018](#)).



Figura 10 – Motor DC utilizado para rotação de rodas, comumente utilizado em protótipos robóticos de carrinhos. Retirado de ([ARDUINOLANDIA, 2018](#)).

2.4 Compilador

Softwares, em geral, possuem como objetivo facilitar a vida do usuário, permitindo a otimização e automatização de diversas tarefas de forma que apenas um indivíduo possa realizar o trabalho de diversos ([PECORELLI, 2014](#)). Esses *softwares* são simplesmente sequências de operações abstratas escritas em uma linguagem de programação que extraem a complexidade das operações realizadas pelo *hardware*. Contudo, as operações programadas pelo usuário na linguagem de programação são incompatíveis com as operações e linguagem aceitas pelo *hardware*, de forma que é necessário uma ferramenta para realizar a tradução da linguagem utilizada pelo usuário para a linguagem aceita pelo *hardware*. O *software* responsável por essa tradução é chamado de compilador ([COOPER; TORCZON, 2012](#)).

O compilador é um *software* que traduz um código escrito em uma linguagem de programação, código fonte, em código de outra linguagem de programação, código alvo, de forma que a prepare para a execução. Para que essa tradução ocorra o compilador deve entender os aspectos da linguagem, tais como, sintaxe, significados e formatos para

que possa mapeá-la em uma linguagem alvo (COOPER; TORCZON, 2012). Durante esse processo o compilador pode detectar erros presentes no código fonte e gerar um relatório para o usuário (PECORELLI, 2014).

O processo de compilação pode ser dividido em duas grandes partes, a análise, ou *frontend* e a síntese ou *backend*. O *frontend* possui o foco no entendimento e construção do código fonte, o dividindo em partes para gerar uma representação intermediária que será utilizado pelo *backend*. O *backend* mapeia a representação intermediária, alocando e otimizando os recursos disponíveis em *hardware* para gerar o código alvo otimizado (AHO; LAM; SETHI, 2007; COOPER; TORCZON, 2012).

Neste trabalho foi desenvolvido um *frontend* de compilador, qual será responsável pela análise da linguagem proposta. A saída do *frontend* será em formato de linguagem intermediária compatível com o *backend* LLVM - *Low Level Virtual Machine*. A partir desta linguagem intermediária é possível aproveitar a infraestrutura de otimização de código e os *backends* existentes específicos para processadores AVR e PIC.

Devido sua relevância para este projeto, será definido detalhadamente o *frontend* e *backend* de um compilador nas subseções a seguir.

2.4.1 Frontend

O *frontend* de um compilador possui o foco no entendimento do código fonte, o fragmentando o código em trechos menores e impondo estruturas gramaticais especificadas pelo projetista para realizar a detecção de possíveis erros e preparar o código para etapa do *backend*. O *frontend* armazena as instruções do código na árvore sintática e os símbolos identificáveis, tais como variáveis, em uma tabela de símbolos para que possa gerar a representação intermediária do código fonte (AHO; LAM; SETHI, 2007; COOPER; TORCZON, 2012).

O processo realizado pelo *frontend* pode ser dividido em análise léxica, análise sintática, análise semântica e gerador de código intermediário. O código intermediário constitui a entrada do *backend*, conforme apresentado na Figura 11.

2.4.1.1 Analisador Léxico

A análise léxica é a primeira fase do processo de compilação, sendo responsável por ler o código fonte criado pelo usuário e gerar uma sequência de identificadores, chamados de *tokens*, que serão utilizados na análise sintática (FOLEISS et al., 2009). O processo de análise léxica também funciona como um verificador de alfabeto, que verifica se o caractere contido no código fonte existe ou não no alfabeto, eliminando espaços em branco, quebras de linha e comentários (AHO; LAM; SETHI, 2007). O processo de análise léxica é feito por um analisador léxico (PECORELLI, 2014).

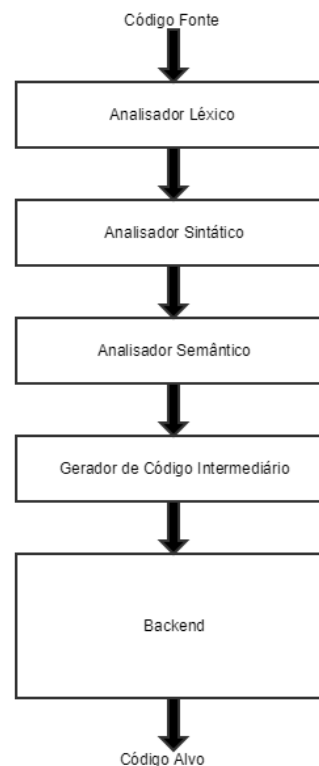


Figura 11 – Fluxograma apresentando as etapas de execução do *frontend* e a integração ao *backend*. O *frontend* aceita como entrada o código fonte e ao final do processo de compilação o resultado será o código alvo. Adaptado de (AHO; LAM; SETHI, 2007).

O *Flex*⁴ (*Fast Lexical Analyzer Generator*) é uma ferramenta que gera o analisador léxico, ou *scanner*. O analisador léxico gerado pela ferramenta *Flex* é escrito, por padrão, em linguagem de programação C ou C++.

O analisador léxico é responsável por reconhecer padrões a partir de uma entrada, código fonte, e as especificações feitas pelo usuário (AHO; LAM; SETHI, 2007). Essas especificações são regras constituídas por expressões regulares, ou *lexemas*, construídas utilizando a linguagem de ações baseadas em padrões chamada *Lex* em conjunto com código C (PECORELLI, 2014).

Após definidas as regras a partir das expressões regulares, o *Flex* divide o código fonte em partes e retorna um *token* para cada expressão encontrada. Quando o analisador léxico encontra identificadores no código fonte, estes são colocados em uma tabela de símbolos (PECORELLI, 2014). Um exemplo de análise léxica pode ser definido como:

Programa em LRE:

⁴ <https://www.gnu.org/software/flex/>

```
exiba(5);
```

Expressões regulares:

```
[+-]?[0..9]+ retorna TOK_INTEIRO;
```

```
"exiba" retorna TOK_EXIBA;
```

Resultado obtido a partir da análise léxica aplicada ao programa em LRE:

```
TOK_EXIBA TOK_ABREPARENTESSES TOK_INTEIRO TOK_FECHAPARENTESSES
```

2.4.1.2 Analisador Sintático

A segunda fase do processo de compilação é a análise sintática, ou *parsing*. O analisador sintático ou *parser*, utiliza os *tokens* produzidos pelo analisador léxico para criar a árvore sintática (PECORELLI, 2014). A árvore sintática ou AST é uma estrutura de dados dinâmica na qual são armazenadas as instruções do código alvo para posteriormente gerar a representação intermediária. A estrutura da árvore AST não é uniforme, ao contrário de árvores binárias ou ternárias, pois depende da sintaxe específica da linguagem (FOLEISS et al., 2009).

O analisador sintático pode trabalhar de duas formas diferentes, sendo essas, *bottom-up* que constrói a árvore de derivação começando pelas folhas e *top-down* que constrói a árvore a partir da raiz para as folhas. Em ambos os casos a entrada é varrida da esquerda para a direita, um símbolo de cada vez (AHO; LAM; SETHI, 2007).

A derivação *top-down* tenta encontrar uma derivação adequada para a entrada de acordo com a especificação da gramática. Esse método tenta prever a próxima entrada e, caso o estado atual não possa derivar a próxima entrada, o analisador retorna a entrada anterior derivando por outro caminho. Esse método é eficiente para gramáticas limitadas. A derivação *bottom-up* trabalha com a acumulação de entradas, quando a derivação se torna visível o analisador a realiza. Na prática apenas poucas gramáticas podem gerar analisadores *bottom-up* eficientes, porém, essas gramáticas contemplam a maior parte das necessidades de uma linguagem de programação (COOPER; TORCZON, 2012). Portanto, para essa pesquisa será desenvolvido um analisador sintático *bottom-up*.

O *software Bison*⁵, versão melhorada do *software Yacc* (*Yet Another Compiler Compiler*) pode ser usado para se construir um analisador sintático *bottom-up*, agrupando os *tokens* e validando-os segundo a gramática especificada no arquivo `.yacc` com as definições

⁵ <https://www.gnu.org/software/bison/>

sintáticas da linguagem, além de apontar erros de sintaxe (AHO; LAM; SETHI, 2007; PECORELLI, 2014; FOLEISS et al., 2009).

2.4.1.3 Analisador Semântico

Após a análise léxica e sintática o programa ainda pode apresentar erros que impedem o processo de compilação. Para detectar esses erros, o compilador necessita verificar informações sobre o contexto da linguagem e suas definições de dados pois são erros específicos da linguagem projetada. Essa etapa é realizada pelo analisador semântico (COOPER; TORCZON, 2012).

Para realizar a análise semântica com eficiência o compilador necessita de informações detalhadas da execução do programa. Alguns exemplos de informações necessárias são (AHO; LAM; SETHI, 2007):

- Verificação de tipo: o compilador necessita relatar erro caso um operador for aplicado a um operando incompatível.
- Verificação do fluxo de controle: caso uma estrutura ou instrução seja encerrada, o compilador deve saber em que ponto a execução deve continuar.
- Verificação de unicidade: o compilador deve verificar se existem elementos, seja variável ou função com o mesmo nome.

2.4.1.4 Gerador de Código Intermediário

O gerador de código intermediário é responsável por gerar uma linguagem intermediária próxima ao código alvo a partir da AST e da tabela de símbolos (FOLEISS et al., 2009). A linguagem intermediária está situada mais próxima para o código alvo do que o código fonte, porém ainda apresentando facilidade de manipulação (AHO; LAM; SETHI, 2007). O LLVM possui uma biblioteca orientada a objetos para a geração da linguagem intermediária. Esta linguagem é independente de arquitetura e por isso, pode ser utilizado para se obter um compilador que gera código alvo compatível para diferentes *hardwares* e microcontroladores (FOLEISS et al., 2009).

Neste projeto, implementaremos a geração de código usando a biblioteca de representação intermediária (IR) do LLVM. Isto fará com que a linguagem proposta seja compatível com os *backends* existentes.

2.4.2 Backend

O compilador construído por este trabalho utiliza o *backend* AVR-GCC, desenvolvido por Dylan McKay e posteriormente anexado aos pacote de compiladores LLVM.

Contudo esse *backend* ainda se mantém como versão experimental, assim possuindo alguns problemas, e recebe suporte de uma comunidade voltada para a melhoria do AVR-GCC (MCKAY, 2017).

Para este trabalho foi desenvolvido um *frontend* capaz de suportar características de outras linguagens de programação com adição de componentes próprios para a Robótica Educacional. Esse *frontend* passa a ser integrado ao *backend* GCC-AVR, assim gerando o compilador proposto.

2.5 Considerações Finais

Neste capítulo descrevemos sobre linguagens fracamente tipadas, o paradigma da linguagem, o analisador léxico, o analisador semântico, o gerador de código intermediário. Estes componentes são necessários para o projeto e implementação da linguagem que foi desenvolvida neste trabalho. Em resumo, as seguintes características são consideradas:

- Linguagem: a linguagem proposta é fracamente tipada, ou seja, não necessitam de definição direta do tipo de variável, essa tarefa será realizada pelo analisador semântico.
- Paradigma: a linguagem é estrutural, do tipo textual sendo orientada a eventos.
- Análise léxica: Foi utilizado o software *Flex* para essa tarefa.
- Análise sintática: Foi utilizado o software *Bison* para essa tarefa.
- Análise semântica: Foi desenvolvido programas na linguagem de programação C++ para realizar a análise semântica.

3 TRABALHOS RELACIONADOS

3.1 Introdução

Neste capítulo são apresentadas publicações relacionadas a linguagens de programação desenvolvidas para a utilização em robôs com foco educacional. Para a realização desta pesquisa foram utilizados as bases de busca e os critérios apresentados na Subseção 3.2 e, por fim, selecionados os trabalhos apresentados na Subseção 3.3.

3.2 Critérios de Busca

Como este trabalho propõe uma linguagem de programação específica para a Robótica Educacional, inicialmente foram buscados pesquisas que apresentassem informações relacionadas a aplicação de linguagens de programação como ferramenta educacional. A partir destes trabalhos, passaram a ser selecionados apenas trabalhos que estivessem relacionados diretamente a robótica, seja a proposta de uma nova biblioteca para linguagens de propósito geral adaptadas para a robótica, a proposta de uma linguagem específica para a robótica educacional ou o desenvolvimento de um compilador.

Para esta pesquisa foram utilizados as bases de dados Google Scholar, IEEE Xplore Digital Library, ACM Digital Library e o Portal de Periódicos da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) utilizando a *string* de busca: 'compilador *and* linguagem de programação *and* Robótica Educacional' e uma equivalente em inglês.

Desse modo, os trabalhos elencados neste capítulo englobam critérios relacionados a linguagem de programação para a educação, ou linguagens de programação Robótica Educacional ou desenvolvimento de compiladores

3.3 Trabalhos Analisados

Com base nas especificações apresentadas acima, os seguintes trabalhos relacionados foram selecionados:

3.3.1 Using visual programming kit and Lego Mindstorms: An introduction to embedded system (T1)

O trabalho intitulado 'Using visual programming kit and Lego Mindstorms: An introduction to embedded system' dos autores (KIM; JEON, 2008) apresenta uma pesquisa sobre a linguagem de programação LEGO NXT, qual foi desenvolvida exclusivamente para a plataforma Lego Mindstorms pela empresa LEGO. A linguagem NXT é linguagem de programação visual, que consiste na montagem de blocos sequencia para a programação de um robô LEGO. A Figura 2 apresentada no Capítulo 2 demonstra um exemplo de programa desenvolvido em NXT.

A linguagem de programação conta um ambiente de desenvolvimento chamado de Lsoft (LEGO *software*), qual possui diversos menus para a seleção dos blocos. É possível realizar a modificação de aspectos dos blocos, como por exemplo, alternar rotações de um motor em sentido horário ou anti-horário. Também é possível modificar aspectos relacionados a blocos de sensores, tais como temperatura ou distâncias.

Para os autores a utilização do Lsoft é melhor quando comparado com outros ambientes de desenvolvimento para linguagem de programação visual pois apresenta um sistema simplificado de arrastar um bloco e os ligar para criar um código. Outro fator citado pelos autores consiste no fato que cada bloco utilizado pelo NTX é referente a um componente, seja um sensor ou motor do robô e, dessa forma, é mais fácil entender um programa escrito em NTX quando comparados a outras linguagens visuais.

3.3.2 Guaráteca: uma poderosa biblioteca de funções para os robôs baseados em Arduino (T2)

Os autores (GOMES et al., 2016) apresentam, por meio do trabalho intitulado 'Guaráteca: uma poderosa biblioteca de funções para os robôs baseados em Arduino', a proposta de uma biblioteca para o ambiente de desenvolvimento Arduino IDE, visando facilitar a programação de protótipos robôs para competições de robótica que utilizam Arduino.

Para o desenvolvimento dessa biblioteca foi utilizado a linguagem de programação C++ e a biblioteca padrão do Arduino IDE *AF_Motor.h*, na qual foi implementado controladores para motores DC, servos, sensores ultrassônicos, sensores de refletância, sensores de cor, sensores giroscópios, sensores de condução de energia e lâmpadas LED. Como resultado, os autores obtiveram os arquivos *GuaraTeca.h* e *GuaraTeca.cpp*, sendo que estes devem ser posteriormente adicionados ao ambiente de desenvolvimento.

Visando validar o projeto proposto pelos autores, estes utilizaram de entrevistas com usuários que possuíam conhecimento básico sobre programação de robôs, alunos

participantes de competições de robótica. Com o resultado da entrevista foi possível afirmar que houve melhoria na facilidade de programação de robôs que utilizavam algum dos componentes qual a biblioteca englobava. Outro fator de validação foi a quantidade de linhas de economia de código obtido pela utilização da biblioteca, onde foi possível observar que, com a utilização da biblioteca, foi possível economizar linhas de códigos em robôs selecionados pelos autores.

3.3.3 Brasilino: biblioteca para Arduino que permite o desenvolvimento de projetos em português do Brasil (T3)

Os autores (NETO; MARTINS; ARAÚJO, 2016) apresentam, por meio do trabalho 'Brasilino: biblioteca para Arduino que permite o desenvolvimento de projetos em português do Brasil', uma biblioteca para ser integrada no ambiente de desenvolvimento Arduino IDE. O objetivo dessa biblioteca é facilitar a programação de códigos, sendo que estes podem ser escrito em português brasileiro, para a plataforma de prototipação Arduino.

A biblioteca desenvolvida utiliza o comando *define* da linguagem de programação C++ para realizar a tradução de trechos de código em C++ para o idioma português, além da adição multi traduções de instrução com intuito de facilitar a sintaxe da programação. O comando *define* pré-processa constantes declaradas, ou seja, a biblioteca desenvolvida define os trechos de código de código na linguagem de programação C++ atrelados a palavras em português, sendo que, múltiplas palavras em português podem se referir ao mesmo trecho de código.

Para validar o trabalho, os autores utilizam entrevistas com alunos do ensino técnico, do IFPE Campus Recife, que não haviam tido contato com linguagens de programação anteriormente. Os alunos compararam dois códigos, um desenvolvido utilizando C++ e outro desenvolvido utilizando a biblioteca, e dissertaram sobre estes. Ao final, todos os alunos entrevistados afirmaram que a biblioteca melhorava o entendimento sobre o trecho de código apresentado e dessa forma validando a pesquisa.

3.3.4 DuinoBlocks for Kids: 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 (T4)

Os autores (QUEIROZ; SAMPAIO; SANTOS, 2016) por meio do trabalho 'DuinoBlocks for Kids: 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' apresentam um ambiente de programação integrado a uma linguagem de programação visual voltado para o ensino de Robótica Educacional à crianças, onde o usuário pode mon-

tar um código, ou sequência de instruções, por meio de blocos desenhados que representam funções em C++.

Para o desenvolvimento dessa pesquisa os autores projetaram um ambiente de desenvolvimento (IDE) tendo como base a biblioteca Ardublockly, onde foram feitas as traduções dos comandos embutidos nos blocos desenhados do inglês para o português e adicionados novas funções. A IDE projetada é utilizada de forma *online* e realiza a tradução, transforma a sequência de instruções em um código C++ que posteriormente será compilado e transferido para a plataforma de prototipação Arduino.

Os autores deste projeto propõe que o uso desta biblioteca não seja exclusivo para crianças e que este seja melhorado em versões posteriores.

3.4 Resumo Comparativo

Por meio dos trabalhos selecionados é possível notar que todos estes apresentam um meio, seja a proposta de uma biblioteca para a integração em ambientes de desenvolvimento existentes ou a proposta um novo ambiente de desenvolvimento, para facilitar a utilização que kits robóticos no ensino, sendo esse o foco principal da Robótica Educacional.

Contudo, é possível observar que os trabalhos selecionados se diferem em alguns critérios do trabalho proposto (TR), como pode ser observado na Tabela 1. Os critérios de diferenciação são:

- C1: A linguagem de programação, biblioteca ou compilador é *open-source*.
- C2: Os códigos gerados nessa biblioteca ou linguagem de programação podem ser compilados diretamente, ou seja, não necessita de tradução.
- C3: É possível desenvolver códigos em português brasileiro nessa linguagem de programação ou biblioteca.
- C4: A linguagem de programação, biblioteca ou compilador é voltada para a Robótica Educacional.
- C5: A linguagem de programação facilita a programação eventos.

3.5 Considerações Finais

De acordo com as pesquisas apresentadas neste capítulo, é possível notar que estas apresentam propostas para a necessidade de uma linguagem específica para a Robótica Educacional. Contudo, os seguintes trabalhos levantados deixam transpassar aspectos importantes no quesito do desenvolvimento de uma linguagem.

Tabela 1 – Comparativo entre trabalhos

	C1	C2	C3	C4	C5
T1	Não	Não	Não	Sim	Sim
T2	Sim	Sim	Não	Não	Não
T3	Sim	Sim	Sim	Sim	Não
T4	Sim	Não	Sim	Sim	Sim
TR	Sim	Sim	Sim	Sim	Sim

Os trabalhos apresentados por [Gomes et al. \(2016\)](#) e [Neto, Martins e Araújo \(2016\)](#), apresentam bibliotecas para facilitar a utilização da linguagem de programação C++ para o Arduino por meio da Arduino IDE. Nesse sentido, a biblioteca entra como um apoio a linguagem, transformando os comandos de inglês para português brasileiro mas mantendo a complexidade da programação em C++ e a limitação dos comandos contidos na biblioteca.

O autor [Queiroz, Sampaio e Santos \(2016\)](#) apresenta uma IDE de programação visual para Arduino, onde a rotina criada pela ligação de blocos é traduzida em linguagem de programação C++ para ser posteriormente ser compilada para a plataforma de prototipação. Contudo, de acordo com ([RODRIGUEZ; CUESTA, 2016](#)), linguagens de programação visuais são limitadas aos blocos desenvolvidos.

No mesmo sentido entra a linguagem de programação proprietária NXT projetada especificamente para a plataforma LEGO. Os autores ([KIM; JEON, 2008](#)), apresenta detalhes da linguagem em sua pesquisa.

Contudo, nenhuma das pesquisas citadas apresentam uma linguagem de programação em conjunto com um compilador específico para a Robótica Educacional que possa abstrair parte da dificuldade da programação das rotinas, fugindo da rigurosidade presente na programação em C++, e que ao mesmo tempo seja pouco limitada. A linguagem desenvolvida também facilita a programação de eventos, característica presente apenas nas linguagens de programação visual dos trabalhos relacionados.

4 LINGUAGEM DE PROGRAMAÇÃO LRE

4.1 Introdução

Nesta seção será feita a descrição da forma como o compilador e a linguagem de programação LRE (Linguagem para Robótica Educacional) foram desenvolvidos, demonstrando o funcionamento e utilização da linguagem bem como as ferramentas e metodologias aplicadas no desenvolvimento do compilador.

Para a construção da compilador para a linguagem LRE proposta por esta pesquisa, foi utilizado como base o protótipo de compilador ROBCMP, desenvolvido pelo professor doutor Thiago de Oliveira Borges e disponibilizado em seu perfil na plataforma de desenvolvimento de *software* GitHub⁶. O ROBCMP é um protótipo de compilador utilizado para auxiliar as aulas de compiladores ministradas para o curso Ciências da Computação na Universidade Federal de Jataí .

4.2 Linguagem de Programação LRE

Com base em (GOMES et al., 2016), existe a necessidade de uma linguagem de programação para a Robótica Educacional em português. Nesse sentido, a linguagem LRE é proposta com comandos em português. A nomenclatura dos comandos de interrupção é baseada no ambiente de desenvolvimento Duino Blocks for Kids, apresentado por (QUEIROZ; SAMPAIO; SANTOS, 2016). O restante dos comandos possui a nomenclatura baseada na biblioteca brasilino, apresentado por (NETO; MARTINS; ARAÚJO, 2016).

A LRE é uma linguagem de programação textual, onde o usuário, por meio de um editor de texto, pode programar de forma escrita, as ações ou comportamentos que deseja representar através da linguagem. A vantagem das linguagens textuais consiste na quantidade de comportamentos possíveis de serem utilizados, não se limitando a uma gama específica como os blocos presentes em linguagens visuais.

A linguagem proposta é fracamente tipada, ou seja, o usuário que utilizar a linguagem não precisa declarar tipos de variáveis, basta atribuir valores a estas e o compilador irá atribuir o tipo automaticamente, seja inteiro, real ou caractere.

A linguagem LRE é orientada a eventos, ou seja, quando uma determinada condição é detectada, uma determinada execução ocorre. Os eventos disponíveis na linguagem são baseados nas interrupções externas disponibilizados pelo *hardware* das plataformas de

⁶ <https://github.com/thborges/robcmp>

prototipação Arduino e portanto pode ser utilizados em apenas algumas portas específicas. As portas que permitem a interrupção variam de acordo com o modelo das plataformas. Os detalhes relacionados as interrupções serão apresentados na subseção 4.2.8.

As Subseções 4.2.1 à 4.2.8 detalham a semântica, sintaxe e funcionalidades da linguagem LRE.

4.2.1 Operadores

Os operadores lógicos e aritméticos aceitos pela linguagem de programação são apresentados na Tabela 2 juntamente com sua representação em caracteres aceitos pela linguagem LRE.

Tabela 2 – Operadores aceitos pela linguagem LRE

<i>Utilização</i>	<i>Significado</i>
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
=	Atribuição
==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
&&	<i>E</i> lógico
	<i>OU</i> lógico

4.2.2 Comando de Exibição

Os comandos de exibição, por sua vez, tem como objetivo exibir um parâmetro contido dentro de parenteses esse parâmetro pode ser escrito por meio de uma operação matemática, uma variável ou apenas uma frase, desde que, colocada entre aspas.

A exibição é feita caso o *hardware* do protótipo possua um *display* LCD. A utilização deste comando é apresentada no exemplo abaixo:

```
exibir('parâmetro') //irá exibir o texto parâmetro
```

```
exibir(parâmetro) //será exibido o valor contido na variável parâmetro
```

```
exibir(5) //exibe o valor 5
```

4.2.3 Tipos de Dados

Na linguagem de programação LRE foram mantidos os tipos de dados inteiro, real e caractere. A atribuição de valor em variáveis ocorre de forma dinâmica pelo analisador semântico, ou seja, basta atribuir um valor a uma variável e a mesma será definida de acordo com o tipo do valor.

```
var a = 10.5 // Variável será atribuída como real pelo analisador semântico
```

```
var b = 7 // Variável será atribuída como inteiro pelo analisador semântico
```

4.2.4 Estruturas de Decisão

A linguagem de programação LRE possui um comando para realizar atividades relacionadas às decisões. Ao utilizá-la, o usuário deverá definir uma operação lógica a ser realizada, esta deve estar precedida do comando 'se' e a condição colocada entre parênteses.

Opcionalmente o usuário poderá colocar um trecho de código que será executado caso a condição do anterior não seja válida. Essa segunda condição será válida caso o usuário coloque o comando 'senao' após a chave que encerra o bloco de comandos 'se'. Um exemplo de utilização pode ser visto abaixo.

```
se (operação lógica){ // primeiro comando de decisão

// específica as instruções para executar caso verdadeiro

} senao{ // segundo comando de decisão (opcional)

// específica as instruções para executar caso seja falso

}
```

4.2.5 Estruturas de Repetição

Para codificar rotinas de repetição utilizando a linguagem de programação LRE, o usuário deve especificar uma condição que, enquanto verdadeira, executa uma sequência de código. Um exemplo de utilização da utilização de estruturas de repetição:

```
enquanto (operação lógica){
```

```
// instruções a serem repetidas  
  
}
```

4.2.6 Funções

A linguagem de programação LRE suporta a programação de funções, desde que as mesmas sejam declaradas em tipo *void*. A linguagem de programação LRE não aceita o retorno de valores por meio de funções ou a passagem de parâmetro, sendo estas utilizadas para a organização de trechos de código.

A sequência de instruções contidas em uma função será executada quando a mesma for chamada em algum momento da execução do código principal. Um exemplo de declaração de função e chamada de função é apresentado abaixo.

Declaração de função

```
funcao Incremental(){  
    ...rotina a ser executada ...  
}
```

Chamada de Função

```
Incremental();
```

4.2.7 Utilização de Portas

Por meio da linguagem de programação LRE é possível passar valores para as portas do microcontrolador da plataforma de prototipação. O valor passado pode alterar a voltagem de uma porta ativando ou desativando essa.

A passagem de voltagem é feito por meio do comando 'saidaN = M', onde N representa a porta qual vai receber a voltagem e M consiste em um número de 0 à 255, onde 0 representa voltagem baixa 0 volts e 255 representa voltagem alta, aproximadamente 5 volts. Alguns exemplos de passagem de voltagem são:

```
saida8 = 0; \\Passa a voltagem 0 para a porta 8  
saida8 = 255; \\ Passa a voltagem para a porta 8
```

4.2.8 Eventos

Eventos são trechos de código que serão executados assim que um determinado gatilho for acionado. A linguagem de programação LRE aceita eventos ocasionados por meio

de interrupções externas, ou seja, interrupções geradas por um componente, frequentemente um sensor, em uma determinada porta.

As interrupções externas podem ocorrer em quatro diferentes estados, sendo esses:

- Baixo: ocorre a interrupção quando a porta, que recebe a interrupção, detecta uma voltagem baixa de energia, aproximadamente 0v.
- Alternado: ocorre a interrupção quando existe uma variação na voltagem de energia, seja aumentando ou diminuindo.
- Crescente: ocorre a interrupção quando a voltagem de energia começa a subir.
- Caindo: ocorre a interrupção quando a voltagem de energia começa a cair.

Para especificar a porta dentro do comando de interrupção, é necessário utilizar o comando 'portaN', onde N é o número da porta qual pode ocorrerá a interrupção. Um exemplo de comando de evento pode ser visto abaixo:

```
quando porta_que_interrompe esta tipo_de_interrupção{
    ...rotina a ser executada ...
}
```

No campo 'porta_que_interrompe' deve ser passado a porta qual vai gerar a interrupção, sendo essa representada por alores do tipo constante e variável desde que sejam inteiros. As portas elegíveis a serem interrompidas são especificadas pela documentação do *hardware*.

No campo 'tipo_de_interrupção' deve ser passado o tipo de interrupção que será gerado pela porta. Esse valor deve ser uma constante, limitado a uma das seguintes opções: baixo, alternado, crescente e caindo.

Também é possível gerar uma evento de temporização, onde um determinado trecho de código será executado em intervalos de tempo pré-definidos pelo usuário. Um exemplo de evento de temporização pode ser visto abaixo:

```
a cada tempo_em_segundos segundos{
    ...rotina a ser executada ...
}
```

Devem ser passados valores do tipo inteiro ou real no campo 'tempo_em_segundos'.

4.3 Compilador LRE

Após a definição das características da linguagem de programação LRE, as mesmas foram adicionadas ao *frontend* do compilador LRE. Posteriormente o *frontend* gerado é integrado a um *backend* para AVR, contido no LLVM, que se encontra em estado experimental. Nesta pesquisa, utilizou-se do *backend* LLVM que consiste em um conjunto de ferramentas e tecnologias de compiladores.

A Subseção 4.3.1 detalha a construção do *frontend* do compilador LRE proposto por este trabalho e a Subseção 4.3.2 detalha o funcionamento do compilador construído.

4.3.1 Construção do Compilador

Após definida as características e componentes da linguagem de programação, foram necessárias realizar modificações no compilador ROBCMP, para que este seja capaz de gerar código alvo equivalente ao código escrito pelo usuário da linguagem LRE.

Para o desenvolvimento do compilador LRE, inicialmente foi necessário a instalação do LLVM e outros componentes necessários (o Anexo A apresenta o passo-a-passo para instalar o LLVM no sistema operacional Ubuntu). Posteriormente foram realizadas as seguintes modificações sobre os arquivos presentes no compilador ROBCMP, sendo essas:

No arquivo `rob.l`, foram feitas as modificações referentes aos conjuntos de caracteres aceitas pela linguagem, adicionando novos comandos a esse e atualizando os existentes para português brasileiro.

No arquivo `rob.y`, foram adicionadas as sintaxes referentes aos novos comandos propostos pela linguagem de programação LRE. Dentre as novas sintaxes adicionadas, encontram-se as interrupções externas.

No arquivo `node.h`, foram atreladas as sintaxes declaradas no arquivo `rob.y` às bibliotecas do Arduino responsáveis por tal funcionalidade.

Os componentes do *frontend* são anexados ao *backend* LLVM, assim gerando o compilador LRE. O compilador LRE gera código intermediário para o LLVM, esse código é posteriormente compilado para AVR por meio do compilador LLC.

4.3.2 Processo de Compilação

Para compilar um arquivo escrito na linguagem de programação LRE, o usuário deve inserir o código alvo em um arquivo `.lre` e executar os seguintes comandos na ordem apresentada:

1. `make out/nome_do_arquivo`: 'esse comando inicia o processo de compilação, chamando o compilador LRE e passando o programa com nome arquivo para ser compilado'

2. `./upload nome_do_arquivo/caminho_da_USB`: 'esse comando é responsável por realizar a transferência do código alvo gerado pelo compilador para a plataforma de prototipação. Também deve ser passado o endereço da USB onde a plataforma está conectada '

Inicialmente o compilador verifica o código presente no arquivo.lre gerado pelo usuário, processo que verifica a sintaxe e a semântica do código. Caso o código apresente algum erro relacionado a uma destas características, o processo de compilação é abortado. Caso o código esteja escrito de forma correta e que esteja semanticamente correto, o código passará por 6 etapas, assim como apresentado na Figura 12.

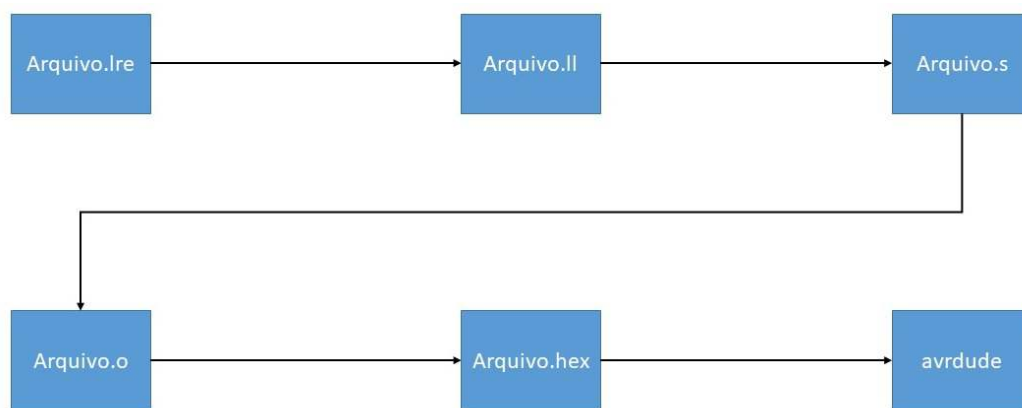


Figura 12 – Sequência de etapas para a compilação de um código escrito na linguagem de programação LRE

Quando o compilador é chamado, primeiramente são passadas as *flags*. Uma *flag* é um mecanismo que permite ou bloqueia determinadas sub-rotinas de serem executadas, nesse contexto, as *flags* passadas permitem a execução de uma sub-rotina que otimiza o código do usuário para o microcontrolador alvo. Durante esse processo, é gerado o arquivo.ll, qual contém o código intermediário para LLVM.

O arquivo.ll é transformado em arquivo.s por meio do compilador LLVM, qual gera código para o microcontrolador AVR. Em sequência, o arquivo.s é transformado otimizado e transformado em arquivo.o pelo avr-gcc.

O arquivo.o é transformado em arquivo.hex por meio do *avr-objcopy*. Por meio do comando *upload*, o programa *avrdude*, parte do *backend* LLVM, é chamado para realizar a transferência do arquivo.hex para o microcontrolador por meio da entrada USB do computador qual está conectado a placa de prototipação.

4.4 Testes

Com o intuito de validar a linguagem de programação e o compilador proposto, foram realizados testes de caixa preta com as interrupções externas e interrupções de tempo implementadas ao compilador.

Foi escolhido a utilização de testes de caixa preta para verificar validação do compilador. Os testes realizados são iniciais, e, no futuro, poderão ser realizados testes unitários para garantir a validade do compilador.

Os testes realizados utilizaram o circuito apresentado na Subseção 4.4.1, e são detalhados nas Subseções 4.4.2 e 4.4.3.

4.4.1 Materiais Utilizados

Para realizar testes com o compilador e a linguagem de compilação LRE, foi utilizado um circuito composto por:

- Três LEDs.
- Um botão.
- Um resistor de 1000 Ohms.
- Cabos *Jumper*.
- Uma *protoboard*
- Plataforma de prototipação Arduino Uno R3 com microcontrolador Atmega 328P.

Na plataforma de prototipação Arduino Uno, apenas as portas 2 e 3 podem receber interrupções externas, sendo essa definição documentado pelo seu *datasheet*⁶, documento qual contém as especificações do *hardware*.

No presente circuito, o botão é conectado a um cabo *Jumper* responsável pelo fornecimento de energia. Um resistor é conectado a saída de energia, qual posteriormente é conectado a porta 2 do Arduino Uno por meio de um *Jumper*. Neste circuito a porta 2 será responsável pela detecção das interrupções geradas pelo botão.

As LEDs são ligadas a portas aleatórias. O esquema do circuito utilizado pode ser visto na Figura 13.

⁶ <https://www.farnell.com/datasheets/1682209.pdf>

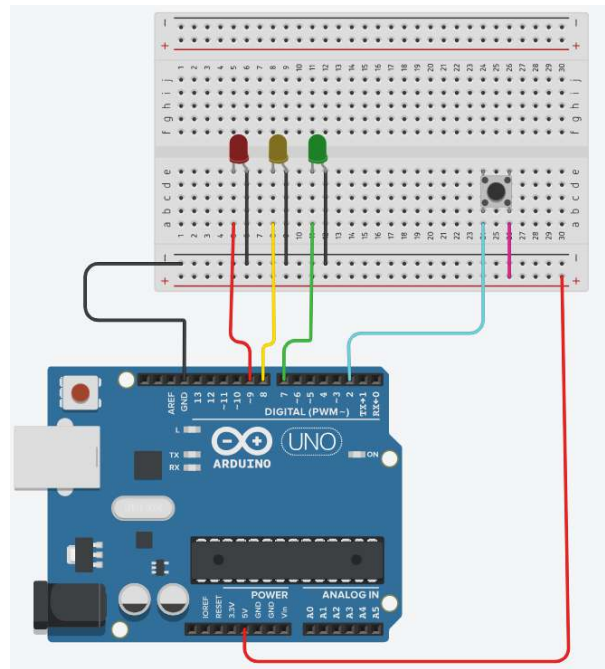


Figura 13 – Esquema do circuito utilizado para a realização de testes com códigos desenvolvidos na linguagem de programação LRE e códigos equivalentes desenvolvidos na linguagem de programação C++

4.4.2 Teste com Interrupção Externa

Os testes de caixa preta relacionados a interrupções externas consiste em um trecho de código que, quando o botão do circuito especificado na Seção anterior, é pressionado, gera uma interrupção que altera o estado das LEDs. Para foi utilizado o código de interrupção externa escrito em LRE apresentado no Anexo B.

Para averiguação, um código equivalente foi construído na linguagem de programação C++ e compilado através da plataforma Arduino IDE, onde observou-se um comportamento idêntico ao código construído na linguagem LRE e compilado pelo compilador LRE. O código escrito em C++ é apresentado no Anexo B.

Os códigos desenvolvidos foram utilizados para comprovar a independência da interrupção, ou seja, ocorrem fora do *loop* principal do código. Estes executam os seguintes passos:

1. Inicialmente o código instancia uma variável com valor 0 e liga as portas 8 e 9;
2. O evento de interrupção é gerado, definido a porta 3 como entrada no estado crescente e atribuindo o código a ser executado.
3. Quando o evento ocorrer, o valor de x se altera, quando x for 1, será alternado para 0 e vice-versa.

4. Um laço de repetição infinito é criado, onde dentro deste a porta 7 vai alternar entre ligado e desligado em intervalos de meio segundo.
5. Ainda dentro do laço existe uma estrutura de condição que verifica o valor da variável x . Caso o valor de x seja 0 as portas 8 e 9 serão ativadas, caso x seja 1, as portas 8 e 9 serão desativadas.

Foi observado que ao pressionar o botão por um curto período de tempo a porta detecta o crescimento da voltagem e LEDs, que inicialmente estavam ligadas, desligam. Ao soltar o botão o comportamento de queda de voltagem é detectado, mas não ativa a interrupção. Caso o botão seja pressionado novamente por um curto período de tempo a porta detectará o crescimento novamente e as LEDs irão ligar. O comportamento se repete todas as vezes que o botão for pressionado por um curto período de tempo e a porta detectar a alteração da voltagem.

Para ambas as linguagens de programação, o mesmo trecho de código foi utilizado, alternando o tipo da interrupção. Foram testados todos os tipos de interrupções e os comportamentos observados foram os seguintes:

- Alternado: as LEDs inicialmente estão ligadas, quando pressionado o botão, a porta detecta a voltagem alta e o estado das LEDs é modificado para desligado, Caso o usuário solte o botão a voltagem baixa é detectada e o estado das LEDs será alterado novamente para ligado e assim ocorrendo a cada alteração detectada pela porta.
- Caindo: as LEDs inicialmente estão ligadas, quando pressionado o botão, a porta detecta a voltagem alta mas a interrupção não é disparada. Caso o usuário solte o botão, as LEDs irão apagar pois a porta detecta que a voltagem está baixa e ativa a interrupção programada. A interrupção ocorrerá apenas quando o usuário soltar o botão que foi anteriormente pressionado.
- Baixo: as LEDs inicialmente estão ligadas, quando pressionado o botão, a porta detecta a voltagem alta mas a interrupção não é disparada. Instantaneamente quando o usuário soltar o botão, a interrupção de caindo será ativada e o evento será executado. Assim como na interrupção baixa, o estado das LEDs será alterado apenas quando o usuário soltar o botão.

4.4.3 Teste com Interrupção de Tempo

Para programar eventos de tempo, é necessário a utilização de interrupções de tempo. Uma interrupção de tempo ocorre quando o contador atinge um tempo pré-definido pelo programador, como por exemplo, acender uma LED a cada 2 segundos. A integração de interrupções de tempo ao compilador LRE ocorreu por meio da adição da biblioteca

TimerOne, essa mesma biblioteca é utilizada para realizar as interrupções de tempo no ambiente de desenvolvimento Arduino IDE.

Para a comparação foram escritos códigos equivalentes, apresentados no Anexo C. Esses códigos foram utilizados para testar a independência da interrupção, ou seja, se ela ocorre fora do *loop* de execução principal. Esses códigos podem ser descritos pelo seguinte fluxo de execução:

1. Inicialmente o código instancia uma variável com valor 0 e liga as portas 8 e 9;
2. O evento de interrupção de tempo é gerado, definindo o intervalo de 0.5 segundos. Ou seja, a cada 5 segundos o valor da variável x será alternado entre 1 e 0.
3. Um laço de repetição infinito é criado, onde dentro deste a porta 7 vai alternar entre ligado e desligado em intervalos de meio segundo.
4. Ainda dentro do laço existe uma estrutura de condição que verifica o valor da variável x. Caso o valor de x seja 0 as portas 8 e 9 serão ativadas, caso x seja 1, as portas 8 e 9 serão desativadas.

Após a compilação e execução de ambos os algoritmos, foi perceptível o mesmo comportamento. Quando programado a interrupção de tempo em C++, o intervalo de tempo deve ser definido em micro-segundos, enquanto que na linguagem LRE o tempo deve ser definido em segundos.

4.5 Considerações Finais

Considerando a necessidade de uma linguagem de programação para a robótica educacional que estivesse entre alta complexidade e uma linguagem limitada para a adição de componentes, a linguagem de programação LRE foi proposta e desenvolvida.

A linguagem LRE conta com aspectos provenientes de linguagens de programação de propósito geral, tais como funções, estruturas de repetição e decisão, operadores lógicos e aritméticos. A linguagem LRE também foi desenvolvida para ser programada em português brasileiro com comandos simples, e, assim, simples de ser utilizada por usuários com pouco conhecimento sobre programação em geral.

Outro fator a ser considerado consiste nas interrupções, onde, um protótipo robótico ou robô desenvolvido, seja para ensino ou para diversão, deve ser capaz de realizar interrupções. A interrupção é um parâmetro imprescindível para linguagem de programação com propósito específico para a robótica, pois permite que o programador desenvolva sua rotina sabendo que quando haja o disparo de um gatilho, seja por tempo ou ação do usuário final, esse comando será executado de imediato.

A linguagem LRE conta com duas formas de interrupção: externa, proveniente da entrada de valor em uma porta específica da plataforma de prototipação, ou por tempo, interrupção que será realizada quando atinja um tempo pré determinado. Com as interrupções e as características provenientes das linguagens de propósito geral, torna-se possível a programação de rotinas simples a rotinas complexas para uma ampla gama de protótipos robóticos para a Robótica Educacional.

Quanto a linguagem de programação, é possível perceber a facilidade de leitura de um código escrito na linguagem LRE, onde o usuário programa as ações que serão realizadas ao acionar o evento dentro do próprio bloco de texto do evento, ao contrario da linguagem C++, onde o usuário necessita criar uma função que será invocada posteriormente pela linha que declara a interrupção.

A linguagem LRE não necessita de um *main* para conter as ações que serão executadas repetidamente, isso fornece ao programador a liberdade de desenvolvimento de códigos que contenham apenas eventos, sem instruções a serem repetidas múltiplas vezes.

Para que seja possível a utilização de uma linguagem de programação, que será posteriormente executada em um *hardware*, surge a necessidade do desenvolvimento de um *software* compilador, qual nesta pesquisa apresenta-se o compilador LRE, desenvolvido para compilar a linguagem de programação LRE.

O compilador LRE está disponível no repositório do GitHub que contém o compilador ROBCMP em um *realise* com o compilador LRE para qualquer usuário interessado em utiliza-lo, dessa forma, sendo *open-source*. Caso o usuário queira, e possua conhecimento na área, poderá modificar o compilador LRE para melhor se adequar as suas necessidades, adicionar novas funções, comandos ou bibliotecas.

Neste capítulo apresentamos os testes de caixa preta para validar algumas funcionalidades desenvolvidas na linguagem. No próximo capítulo, apresentamos alguns experimentos para comparar o código gerado pelo LRE com o código gerado pela IDE Arduino.

5 EXPERIMENTOS REALIZADOS

5.1 Introdução

Nesta seção será apresentado os experimentos que foram feitos com o compilador e linguagem de programação LRE. Os experimentos foram realizados após a validação do compilador e linguagem de programação LRE.

Quanto aos experimentos realizados com o compilador e linguagem de programação LRE, foram analisados os aspectos de tempo de compilação e tamanho do arquivo de código alvo gerado.

5.2 Experimentos com Tempo de Compilação

No aspecto de tempo de compilação deve se considerar que o tempo utilizado para o processo de compilação é um atrativo para programadores de todos os níveis. Nesse sentido, foi proposto que o compilador LRE compilar a linguagem de programação LRE possua uma diferença de até 180% do tempo de compilação da linguagem C++ utilizando o compilador GCC.

Para avaliar o tempo de compilação dos códigos desenvolvidos na linguagem LRE e compilados no compilador LRE, foi utilizado o comando *time* juntamente com os comandos de compilação. Esse comando exibe o tempo utilizado para a execução do processo.

A medição do tempo de compilação da linguagem LRE ocorre em duas etapas, inicialmente é compilada a pasta *arduinowire* por meio do comando *make* e o tempo registrado. Posteriormente é compilado o código desenvolvido e o tempo resultante das duas etapas é somado, assim totalizando o tempo de compilação. Quando a pasta *arduinowire* é compilado pela primeira vez as informações ficam pré-compiladas, dessa forma, não é necessário que seja compilado novamente e assim diminuindo o tempo de compilação, contudo para a realização dos experimentos, a pasta *arduinowire* foi compilada em todas as etapas do experimento.

Para medir o tempo de compilação dos códigos desenvolvidos na linguagem de programação C++, foi cronometrado, utilizando o cronometro do sistema operacional *Windows*, o tempo utilizado pelo processo do ambiente de desenvolvimento Arduino IDE. Para realizar a compilação, a cada experimento, é necessário encerrar o ambiente de desenvolvimento e o iniciar novamente. A primeira compilação da IDE armazena na memória as bibliotecas compiladas, e portanto, a partir da primeira compilação as posteriores são compiladas em tempos menores.

Para a realização dos experimentos, foram desenvolvidos 8 algoritmos, com características distintas em LRE e equivalentes em C++. Cada algoritmo foi compilado quatro vezes, e assim foi possível obter uma média dos tempos de compilação.

A Tabela 3 apresenta a média dos tempos de compilação, em segundos, obtidos a partir dos experimentos com os algoritmos no compilador LRE e as médias dos tempos, em segundos, dos algoritmos equivalentes compilados no ambiente de desenvolvimento Arduino IDE.

Tabela 3 – Médias de Tempo de Compilação

	LRE	Arduino IDE
Algoritmo 1	0.466	8.36
Algoritmo 2	0.46	8.91
Algoritmo 3	0.488	7.72
Algoritmo 4	0.47	8.69
Algoritmo 5	0.488	8.86
Algoritmo 6	0.505	9.85
Algoritmo 7	0.484	9.47
Algoritmo 8	0.468	9.30
Média total	0.478	8.86

Os experimentos realizados com o compilador LRE foram feitos sobre uma máquina virtual *Ubuntu*, hospedada em um sistema operacional *Windows 10*. Durante a execução dos experimentos, apenas a máquina virtual estava sendo executada pelo sistema operacional hospedeiro e, na máquina virtual, o compilador LRE estava sendo executado com auxílio de um comando, fornecido pelo sistema operacional, para registrar o tempo de execução do processo.

Os experimentos realizados com o ambiente de desenvolvimento Arduino IDE foram executados diretamente sobre o sistema operacional *Windows 10*, com auxílio de um programa para cronometrar o tempo de compilação.

A partir dos resultados é possível observar que o compilador LRE pode compilar algoritmos até 17 vezes mais rápido que o compilador do ambiente de desenvolvimento Arduino IDE com códigos equivalentes.

5.3 Experimentos com Tamanho de Arquivo

No aspecto de tamanho do arquivo de código alvo, considera-se que a memória disponível em um microcontrolador é pequena para o armazenamento de programas e seu próprio sistema. Considerando que o ambiente de desenvolvimento Arduino IDE utiliza um *backend* consolidado em comparação com o *backend* experimental LLVM utilizado pelo compilador LRE, é proposto que arquivo gerado pelo compilador LRE possua uma

diferença de até 180% do tamanho de arquivo gerado através da utilização da linguagem C++.

Durante o processo de transferência do código fonte gerado pelo compilador LRE para a plataforma de prototipação, o programa *avrdude*, qual é responsável por esse processo, exibe o tamanho final do arquivo. O mesmo programa é utilizado pelo ambiente de desenvolvimento Arduino IDE. Com os dados provenientes deste programa o tamanho de código alvo da linguagem LRE foi comparado ao código alvo da linguagem C++.

Para os experimentos com tamanho de arquivo, foi-se utilizado os mesmos 8 algoritmos desenvolvidos para o experimento de tempo. Cada algoritmo foi compilado 4 vezes e a média do tamanho de código final foi registrada

A Tabela 4 apresenta o tamanho médio final obtido, em *bytes*, a partir da compilação dos algoritmos desenvolvidos em C e em LRE.

Tabela 4 – Médias de Tamanho do Código Alvo

	LRE	Arduino IDE
Algoritmo 1	1392	958
Algoritmo 2	1446	998
Algoritmo 3	1492	994
Algoritmo 4	1454	998
Algoritmo 5	1506	994
Algoritmo 6	2196	1444
Algoritmo 7	1906	1202
Algoritmo 8	1762	1240
Média total	1644	1103

Os experimentos realizados com o compilador LRE foram feitos sobre uma máquina virtual *Ubuntu*, hospedada em um sistema operacional *Windows 10*. Durante a execução dos experimentos, apenas a máquina virtual estava sendo executada pelo sistema operacional hospedeiro e, na máquina virtual, o compilador LRE estava sendo executado.

Os experimentos realizados com o ambiente de desenvolvimento Arduino IDE foram executados diretamente sobre o sistema operacional *Windows 10*.

O tamanho do código final gerado pelo uso da linguagem LRE é, aproximadamente, 149% maior que o código final gerado utilizando a linguagem C++ e compilado no ambiente de desenvolvimento Arduino IDE. Como o compilador LRE utiliza o *backend* LLVM experimental, a diferença de tamanhos entre código final foi considerada aceitável.

5.4 Considerações finais

A diferença de tempo entre os processos de compilação ocorre devido a necessidade do ambiente de desenvolvimento Arduino IDE compilar todas as bibliotecas contidas

nele antes da compilação do código escrito pelo programador. Esse processo aumenta o tempo necessário para a compilação dos programas desenvolvidos para a plataforma de prototipação Arduino.

Durante o processo de compilação, é possível notar que o arquivo gerado pela linguagem LRE é maior que o arquivo gerado pela linguagem C++. Essa diferença ocorre devido as *flags* passadas pelo compilador GCC-AVR contido na plataforma de prototipação Arduino IDE, qual otimiza o arquivo contendo o código final de forma mais eficiente que o processo realizado pelo compilador LRE.

Ao transferir o programa alvo para o Arduino, o programa *avrdude*, responsável pela transferência, informa o tamanho do arquivo final escrito na memória do Arduino.

Apesar da dificuldade de medir o tempo na plataforma Arduino IDE de forma automatizada, o erro proveniente da medição manual através de cronômetro é baixa de forma a não interferir no resultado.

O tamanho do arquivo que contém o código alvo gerado pelo Arduino IDE é exibido pelo programa *avrdude*, qual também é responsável pela transferência do código para a plataforma de prototipação.

6 CONCLUSÕES E TRABALHOS FUTUROS

6.1 Introdução

Neste Capítulo serão apresentadas as conclusões e a contribuição obtidas após a realização da pesquisa, o desenvolvimento da linguagem de programação e do compilador LRE e os experimentos do mesmo. Também são propostos temas para futuras pesquisas relacionados a esta.

6.2 Conclusões

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

A LRE é uma linguagem de programação textual, com comandos escritos em português específicas para a Robótica Educacional, ao mesmo tempo que implementa as principais características de uma linguagem de programação de propósito geral, além de interrupções quais são necessárias para a programação de protótipos de robôs.

Assim como na linguagem de programação C++, frequentemente utilizada para a programação de protótipos desenvolvidos sobre a plataforma Arduino, a linguagem LRE é compatível com a utilização de diversos tipos de sensores ou atuadores, assim, sendo capaz de gerar códigos para tarefas simples, desempenhadas por protótipos pouco complexos, ou tarefas complexas, desempenhadas por protótipos com diversas funcionalidades.

Para que fosse possível utilizar a linguagem LRE , foi desenvolvido o compilador LRE. O compilador LRE consiste em um *frontend* desenvolvido para ser compatível com a sintaxe e semântica propostas para a linguagem de programação LRE. O *frontend* posteriormente é integrado ao *backend* pronto LLVM.

Ao final desta pesquisa, obteve-se uma linguagem de programação específica para a Robótica Educacional, qual deve facilitar a programação de kits robóticos *open-source* por usuários que tenham pouco conhecimento sobre programação. A linguagem e compilador também são *open-source*, ou seja, estará disponível para qualquer usuário que tenha interesse de utilizar a linguagem de programação ou modifica-lá para que melhore o desempenho da mesma.

Por meio de experimentos foi possível concluir que o compilador LRE pode compilar

a linguagem de programação LRE até 17 vezes mais rápido que o compilador do Arduino IDE compila a linguagem C++. A diferença de tamanho do código alvo (aproximadamente 149% maior que o código C++) foi considerada aceitável pelos padrões definidos neste trabalho, devido se tratar de um compilador inicial e também de um *backend* experimental do LLVM, ao contrário da já consolidada plataforma GCC usada na Arduino IDE. Este aspecto deve ser investigado em trabalhos futuros para futuras melhorias de otimização.

6.3 Trabalhos futuros

Como possíveis melhorias para a linguagem e compilador LRE, considera-se a adição das seguintes características:

- Arquivo externo: o arquivo externo permite que o usuário ligue o código desenvolvido na linguagem de programação LRE ao código desenvolvido na linguagem de programação C++, para que possa utilizar de comandos e funções que ainda não foram adicionados a linguagem.
- Estruturas de dados: a implementação de estruturas de dados permitiria, ao usuário da linguagem LRE, a otimização do armazenamento de dados, além do acesso mais eficiente, aproveitando melhor os recursos limitados disponibilizados pelo microcontrolador das plataformas de prototipação
- Parâmetros para as funções: permitirá ao usuário da linguagem LRE, programar funções que contenham retorno de dados, podendo utilizar de técnicas como recursão.
- Implementação de funções unárias: característica importante para operações lógicas.
- *Flags* para melhorar o desempenho do compilador: ao acrescentar ou modificar as *flags* presentes no compilador LRE, é possível otimizar, ainda mais, o código alvo gerado por este, e assim, reduzir o uso da memória do microcontrolador.

REFERÊNCIAS

AHO, A. V.; LAM, M. S.; SETHI, R. *Compiladores - Princípios, Técnicas e Ferramentas*. São Paulo, Brasil: Pearson, 2007. Citado 4 vezes nas páginas 29, 30, 31 e 32.

ARDUINOLANDIA. *Sensores e Atuadores para Arduino*. 2018. Disponível em: <<https://www.arduino-landia.com.br/>>. Acesso em: 07 may 2018. Citado na página 28.

BRAVO, F. A.; GONZÁLEZ, A. M.; GONZÁLEZ, E. A review of intuitive robot programming environments for educational purposes. 2017. Citado 2 vezes nas páginas 21 e 23.

COOPER, K. D.; TORCZON, L. *Engineering a Compiler*. Houston, Texas: Elsevier, 2012. Citado 4 vezes nas páginas 28, 29, 31 e 32.

DEURSEN, A. van.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. 2000. Citado na página 19.

FERDOUSH, S.; LI, X. Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications. *Procedia Computer Science*, Elsevier, v. 34, p. 103–110, 2014. Citado na página 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 3 vezes nas páginas 29, 31 e 32.

FORNAZA, R.; WEBBER, C.; VILLASBOAS, V. Kits educacionais de robótica opções para o ensino de ciências. *SCIENTIA CUM INDUSTRIA*, UCS, v. 3, n. 3, p. 142, 2015. Citado 6 vezes nas páginas 14, 19, 20, 21, 24 e 26.

FRANGO, S. et al. Representative examples of implementing educational robotics in school based on the constructivist approach. In: CAPIN, S. et al. (Ed.). *Simulation, Modeling, and Programming for Autonomous Robots*. [S.l.]: SIMPAR, 2008. Citado 2 vezes nas páginas 18 e 19.

GOMES, J. et al. Guarateca: Uma poderosa biblioteca de funções para robôs baseados em arduino. *Mostra Nacional de Robótica*, p. 1–6, 2016. Citado 5 vezes nas páginas 16, 21, 35, 38 e 39.

JUNGTHON, G.; GOULART, C. M. Paradigmas de programação. *Acesso em 15 de Julho de 2018*, v. 15, 2016. Citado na página 23.

JUNIOR, A. de O. C.; GUEDES, E. B. Uma análise comparativa de kits para a robótica educacional. 2015. Citado 10 vezes nas páginas 15, 18, 19, 20, 21, 22, 24, 25, 26 e 27.

KIM, S. H.; JEON, J. W. Using visual programming kit and lego mindstorms: An introduction to embedded system. In: *2008 IEEE International Conference on Industrial Technology*. [S.l.: s.n.], 2008. p. 1–6. Citado 2 vezes nas páginas 35 e 38.

- LOPES, M. S. dos S. *Ambiente colaborativo para ensino aprendizagem de programação integrando laboratório remoto de robótica*. Tese (Doutorado) — Universidade Federal da Bahia, Bahia, Brazil, 2017. Disponível em: <<http://repositorio.ufba.br/ri/handle/ri/24293>>. Acesso em: 07 may 2018. Citado 3 vezes nas páginas 15, 16 e 22.
- MAHMUD, D. A. *O uso de robótica educacional como motivação a aprendizagem de matemática*. Dissertação (Mestrado) — Universidade Federal do Amapá, Macapá, 2017. Citado 2 vezes nas páginas 15 e 20.
- MAJOR, L.; KYRIACOU, T.; BRERETON, O. Systematic literature review: teaching novices programming using robots. *IET Software*, Institution of Engineering and Technology (IET), v. 6, n. 6, p. 502, 2012. Citado na página 14.
- MARTINS, N. A. *Uma Abordagem com o Microcontrolador PIC 16F84*. [S.l.]: Novatec, 2005. Citado 3 vezes nas páginas 24, 26 e 27.
- MCKAY, D. *AVR-LLVM*. 2017. Disponível em: <<https://github.com/avr-llvm/llvm/wiki/Getting-Started>>. Acesso em: 25 nov. 2018. Citado na página 33.
- MONDADA, F.; BONANI, M.; RIEDO, F. Bringing robotics to formal education The thymio open-source hardware robot. *IEEE Robotics & Automation Magazine*, IEEE, v. 24, n. 1, p. 77, 2017. Citado 3 vezes nas páginas 14, 24 e 25.
- NETO, O. S. M.; MARTINS, T. A. dos S.; ARAÚJO, R. C. C. de. *Brasilino: biblioteca para arduino que permite o desenvolvimento de projetos em português do brasil*. 2016. Citado 3 vezes nas páginas 36, 38 e 39.
- PECORELLI, J. M. F. *Simulador de Linguagem em Texto Estruturado para Autômato TSX3721*. Dissertação (Mestrado) — Universidade Nova de Lisboa, Lisboa, 2014. Citado 5 vezes nas páginas 28, 29, 30, 31 e 32.
- QUEIROZ, R.; SAMPAIO, F. F.; SANTOS, M. P. dos. Duinoblocks4kids: Ensinando conceitos básicos de programação a crianças do ensino fundamental i por meio da robótica educacional. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2016. v. 5, n. 1, p. 1169. Citado 7 vezes nas páginas 15, 20, 21, 22, 36, 38 e 39.
- ROBOCORE. *Kit iniciante de robótica*. 2018. Disponível em: <<https://www.robocore.net/loja/produtos/kit-iniciante-para-robotica.html>>. Acesso em: 07 may 2018. Citado 3 vezes nas páginas 25, 27 e 28.
- RODRIGUEZ, F. M. L.; CUESTA, F. Low-cost educational mobile robot based on android and arduino. *Journal of Intelligent & Robotic Systems*, Kluwer Academic Publishers, v. 81, n. 1, p. 63, 2016. Citado 6 vezes nas páginas 14, 15, 16, 19, 21 e 38.
- SILVA, A. J. B. D. *Um modelo de baixo custo para aulas de robótica educativa usando a interface arduino*. Dissertação (Mestrado) — Universidade Federal de Alagoas, Maceió, may. 2014. Citado 8 vezes nas páginas 14, 15, 18, 19, 24, 25, 26 e 27.
- XAVIER, R. D. *Paradigmas de Desenvolvimento de Software: Comparação Entre Abordagens Orientada a Eventos e Orientada a Notificações*. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, Curitiba, 2014. Citado na página 23.

Anexos

ANEXO A – INSTALAÇÃO DO LLVM NO SISTEMA OPERACIONAL UBUNTU

Neste anexo serão detalhados os passos para a instalação do LLVM em um sistema operacional Ubuntu, para a utilização do compilador LRE, proposto por esta pesquisa.

1. Criar uma pasta, em qualquer diretório, para armazenar os arquivos do LLVM
2. Criar uma pasta *build* e uma pasta LLVM.
3. Na pasta LLVM, utilizar o comando para a instalação do llvm: 'apt-get install LLVM'
4. Na pasta *build*, utilizar o seguinte comando:

```
cmake -G 'Unix Makefiles' -DCMAKE_BUILD_TYPE=Release  
-DLLVM_TARGETS_TO_BUILD=X86 -DLLVM_EXPERIMENTAL_TARGETS_TO_BUILD=AVR  
../llvm
```

5. Na pasta *build*, utilizar o comando 'make -jN', onde N é o número de núcleos virtuais para executar o processo
6. Durante a execução, será solicitado a instalação de alguns outros programas, sendo estes, o avrdude e o svn, os mesmos podem ser instalados da seguinte forma: 'apt-get install gcc-avrduke' e "apt-get install gcc-avr subversion"

O processo apresentado pode levar algumas horas para instalação. Após completar o procedimento, será possível utilizar o compilador LRE.

ANEXO B – CÓDIGOS DE INTERRUPÇÃO EXTERNA

Código de interrupção externa escrito em LRE:

```
x = 0;
saida8 = 255;
saida9 = 255;

quando porta1 esta crescendo {
    x = (x - 1) * (0 - 1);
}

enquanto(1 > 0){
    saida7 = 255;
    espera(500);
    saida7 = 0;
    espera(500);

    se(x == 0){
        saida8 = 255;
        saida9 = 255;
    }senao{
        saida8 = 0;
        saida9 = 0;
    }
}
}
```

Código de interrupção externa escrito em C++:

```
int x = 0;
int led7 = 7;
int led8 = 8;
```

```
int led9 = 9;

void setup() {
    pinMode(led7, OUTPUT);
    pinMode(led8, OUTPUT);
    pinMode(led9, OUTPUT);
    attachInterrupt(1, interrupcao, RISING);
}

void interrupcao(){
    x = (x - 1) * (0 - 1);
}

void loop() {
    digitalWrite(led7, HIGH);
    delay(500);
    digitalWrite(led7, LOW);
    delay(500);

    if (x == 0) {
        digitalWrite(led8, HIGH);
        digitalWrite(led9, HIGH);
    } else {
        digitalWrite(led8, LOW);
        digitalWrite(led9, LOW);
    }
}
}
```

ANEXO C – CÓDIGOS DE INTERRUPÇÃO DE TEMPO

Código de interrupção de tempo escrito em LRE:

```
x = 0;
saida8 = 255;
saida9 = 255;

a cada 0.5 segundos {
    x = (x - 1) * (0 - 1);
}

enquanto(1 > 0){
    saida7 = 255;
    espera(500);
    saida7 = 0;
    espera(500);

    se(x == 0){
        saida8 = 255;
        saida9 = 255;
    }senao{
        saida8 = 0;
        saida9 = 0;
    }
}
```

Código de interrupção de tempo escrito em C++:

```
#include "TimerOne.h"

int led1 = 7;
int led2 = 8;
int led3 = 9;
int estado = 0;
```



```
void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  Timer1.attachInterrupt(interrompetempo, 500000);
}
```

```
void loop() {
  digitalWrite(led1, HIGH);
  delay(500);
  digitalWrite(led1, LOW);
  delay(500);

  if (estado == 0) {
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
  } else {
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
  }
}
```

```
void interrompetempo(){
  estado = (estado - 1) * (0 - 1);
}
```