UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ROBOTOY: FERRAMENTA PARA ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS USANDO ROBÔS ARDUINO

JULIANA CAROLINA BATISTA

BLUMENAU
2016

JULIANA CAROLINA BATISTA

ROBOTOY: FERRAMENTA PARA ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS USANDO ROBÔS ARDUINO

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Profa. Joyce Martins, Mestre - Orientadora

BLUMENAU
2016

ROBOTOY: FERRAMENTA DE ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS USANDO ROBÔS ARDUINO

Por

JULIANA CAROLINA BATISTA

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Profa. Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB
Membro:	Prof. Miguel Alexandre Wisintainer, Mestre – FURB

Blumenau, 06 de julho de 2016

Dedico este trabalho aos meus pais, por terem me ensinado a lutar pelos meus sonhos.

AGRADECIMENTOS

A Deus, por guiar meus caminhos conduzindo-me até aqui.

À minha família, especialmente aos meus pais Sergio Batista e Otilia Simon Batista, pelo amor, pelos conselhos e pelas palavras de incentivo que me ajudaram a concluir mais esta etapa da minha vida.

Ao meu namorado Júlio César Batista pelo amor, compreensão e ajuda.

Ao amigo Marcos Antônio Salvador, por me ajudar nas questões relacionadas à eletrônica.

A minha orientadora, Joyce Martins pela confiança e pela atenção dedicada na orientação deste trabalho.

Ao professor Aurélio Faustino Hoppe por disponibilizar, durante o desenvolvimento deste trabalho, uma das placas do laboratório de robótica para testes.

A todos os professores que contribuíram para a minha formação.

Se a educação sozinha não transforma a sociedade, sem ela tampouco a sociedade muda.

Paulo Freire

RESUMO

Este trabalho apresenta uma extensão de uma ferramenta voltada para o ensino de programação para crianças. A ferramenta consiste em um compilador que através de uma linguagem de programação textual e simplificada, a Robotoy, permite a escrita de programas para serem enviados e executados por robôs. Originalmente, os programas eram executados em robôs do tipo Lego Mindstorms NXT. Com a extensão proposta, a ideia é que um mesmo programa possa ser executado tanto em robôs do tipo Lego Mindstorms NXT como em robôs construídos com base na placa Arduino Mega 2560. Para isto, basta o programador configurar a plataforma utilizada para montagem do robô, além de informações referentes aos componentes que o constituem. Assim, através dos comandos disponibilizados pela linguagem é possível enviar instruções para o robô executar, tais como: andar, parar, identificar cores, entre outras. Destaca-se ainda que a inclusão da plataforma Arduino, uma opção mais acessível devido a sua característica de hardware e software livre, se deve ao fato de que boa parte das instituições de ensino deixam de adotar o ensino de robótica em razão do alto custo dos *kits* proprietários, como o *kit* da Lego, por exemplo. Por fim, no intuito de validar o funcionamento da ferramenta e do robô montado, foram feitos testes e observou-se que o robô foi capaz de executar todos os comandos da linguagem, apesar de apresentar algumas limitações quanto à movimentação e o cenário no qual foi inserido.

Palavras-chave: Linguagens de programação. Ensino de programação. Robótica educacional. LEGO Mindstorms NXT. Arduino Mega 2560.

ABSTRACT

This work presents an extension of a tool focused in teaching programming for children. The tool consists of a compiler that through a simplified textual programming language, the Robotoy, allows the writing of programs to be sent to robots and executed by them. Originally, the programs were executed by robots of Lego Mindstorms NXT type. Using the proposed extension, the idea is that the same program can be executed in Lego Mindstoms NXT robots and robots built based on Arduino Mega 2560 board. To achieve this, the programmer just needs to configure the platform used to build the robot, and the informations about it is components. Thereby, it is possible to send instructions to be executed by the robot using the available commands in the language, like: walk, stop, identify colors, and others. It is important to notice that the use of the Arduino platform is a more accessible option due to its free hardware and software characteristic, since many institutions do not adopt the teaching of robotics due to the high costs of proprietary kits, as for example the Lego's kit. Lastly, to validate the behavior of the tool and the built robot, tests were performed and we observed that the robot was able to execute all language commands, although it showed some limitations about the movements and the scenario where it was inserted.

Keywords: Programming languages. Programming teaching. Educational robotics. LEGO Mindstorms NXT. Arduino Mega 2560.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

DSC – Departamento de Sistemas e Computação

FURB – Universidade Regional de Blumenau

ICSP – *In Circuit Serial Programming*

IDE – *Integrated Development Environment*

GPS – *Global Positioning System*

LCD – *Liquid Crystal Display*

LED – *Light Emitting Diode*

PWM – *Pulse-Width Modulation*

RF – Requisito Funcional

RNF – Requisito Não Funcional

USB – *Universal Serial Bus*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, os processos educacionais são restritos ao solicitar que o aluno faça várias atividades, as quais podem ou não ser realizadas com sucesso. Porém, o fato do aluno ter sido bem sucedido não significa que ele compreendeu o que fez (VALENTE, 1999, p. 38). Paulo Freire, considerado um dos mais influentes educadores brasileiros da história, defendia uma pedagogia da pergunta e não da resposta. Freire acreditava que a pedagogia não poderia existir se não houvesse a curiosidade (FREIRE; FAUNDEZ; COSTA, 1986, p. 23). O também educador e matemático Seymour Papert, responsável pela criação da linguagem Logo em 1968, lembra que é preciso dar mais consciência às crianças sobre o processo de aprendizagem, incentivando-as a participar deste processo. As crianças terão um melhor aprendizado "pescando" por si mesmas o conhecimento de que precisam, do que sendo alimentadas pelo "peixe" que lhes é oferecido na educação tradicional (PAPERT, 1994, p. 125).

Neste contexto, a robótica educativa é considerada uma ferramenta poderosa no processo de aprendizagem, pois exercita e instiga a curiosidade, a imaginação e a intuição, elementos centrais que favorecem experiências estimuladoras da decisão e da responsabilidade (GOMES et al., 2010, p. 211). Além disso, a robótica proporciona melhora do raciocínio lógico e das habilidades manuais, através das atividades de programação e montagem dos robôs. Entretanto, a sua inserção no processo de ensino-aprendizagem esbarra em dificuldades financeiras, pois boa parte das instituições não têm condições de arcar com o alto custo dos *kits* proprietários, usados para o desenvolvimento dos projetos (MEDEIROS FILHO; GONÇALVES, 2008, p. 265).

Assim, uma alternativa para baratear a adoção da robótica é a utilização de plataformas de hardware e de software *open source*, ou seja, plataformas de código aberto que podem ser estendidas e distribuídas, desde que sob a licença original. Dentre estas plataformas, o Arduino ganha destaque por ser uma plataforma de fácil utilização. De acordo com McRoberts (2011, p. 20), "[...] com o Arduino, pessoas que não são da área técnica podem, rapidamente, aprender o básico e criar seus próprios projetos em um intervalo de tempo relativamente curto".

Diante do exposto, este trabalho apresenta uma extensão da ferramenta Robotoy (TORRENS, 2014), que foi construída visando o desenvolvimento de programas para robôs Lego Mindstorms NXT, integrando-a também com a plataforma Arduino. A inclusão da

plataforma Arduino à ferramenta tem como objetivo baratear os custos da sua adoção e oferecer outra opção para programação e construção de robôs[1].

1.1 OBJETIVOS

O objetivo deste trabalho é estender a Robotoy com suporte ao Arduino como plataforma de programação.

Os objetivos específicos do trabalho são:

a) gerar código para o Arduino a partir dos comandos da linguagem Robotoy;
b) fazer a montagem de um robô, conforme especificado por Torrens (2014), tendo como base o Arduino;
c) enviar o código gerado para ser executado na placa Arduino.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. Neste primeiro capítulo, apresenta-se a introdução, os objetivos e a estrutura do trabalho. No segundo capítulo, tem-se a fundamentação teórica, onde é feita a descrição do conceito de compiladores, da história e especificação da placa Arduino, da ferramenta Robotoy e de três trabalhos correlatos a este. No terceiro capítulo é apresentado o desenvolvimento do trabalho, detalhando-se os requisitos principais da ferramenta, a montagem do robô Arduino, a especificação da linguagem e da ferramenta Robotoy e aspectos relacionados à implementação. E, por fim, no quarto capítulo, têm-se as conclusões e sugestões de extensões para trabalhos futuros.

[1] Para o desenvolvimento deste trabalho foi utilizado o Arduino Mega 2560. O valor gasto na aquisição da placa e dos demais componentes foi de aproximadamente R$480,00. Já o *kit* Lego Mindstorms NXT foi descontinuado após o lançamento do *kit* Lego Mindstorms EV3, similar ao NXT. Na loja virtual oficial da Lego ele é vendido por $349,99.

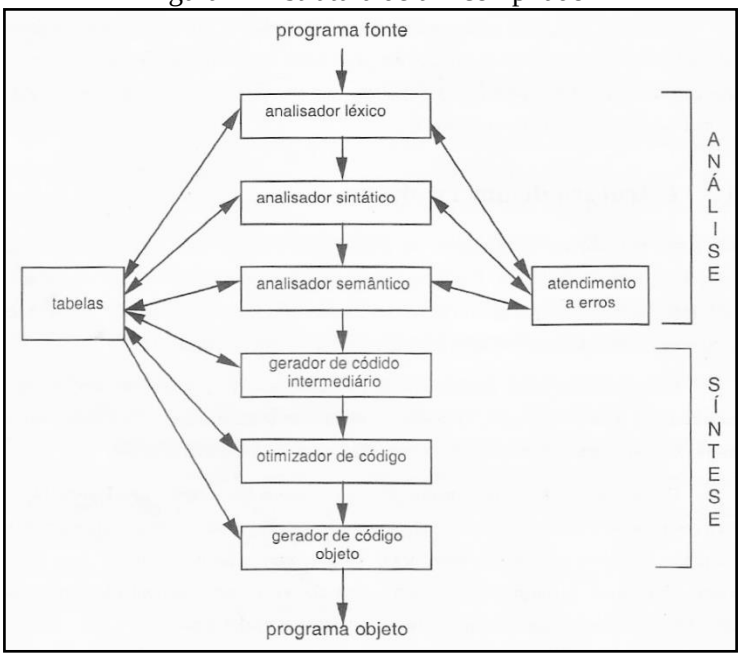
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está dividido em quatro seções. Na seção 2.1 é feita uma breve descrição do conceito e da estrutura de um compilador. A seção 2.2 apresenta a história do Arduino e aspectos referentes ao hardware e à programação da placa. Na seção 2.3 é descrita a ferramenta Robotoy, que foi estendida neste trabalho, e a seção 2.4 detalha três trabalhos correlatos a este.

2.1 COMPILADORES

Um compilador é um sistema que aceita como entrada um programa escrito em uma linguagem de programação (programa fonte) e produz como resultado um programa equivalente em outra linguagem (programa objeto) (PRICE; TOSCANI, 2001, p. 4). Como mostra a Figura 1, o processo que faz o mapeamento do programa fonte para o programa objeto consiste em duas fases: análise e síntese. A primeira fase é composta pelos analisadores léxico, sintático e semântico. Já a segunda fase, a síntese, que constrói o programa objeto desejado, é composta pelo gerador de código intermediário, pelo otimizador de código e pelo gerador de código objeto (AHO et al., 2008, p. 3).

Figura 1 - Estrutura de um compilador



Fonte: Price e Toscani (2001, p. 8).

A primeira etapa do processo de compilação é a análise léxica. O analisador léxico tem a função de percorrer o texto do programa fonte, caractere a caractere, identificando e classificando os símbolos pertencentes à linguagem.

A função do analisador sintático é verificar se os símbolos identificados pelo analisador léxico aparecem na ordem correta, conforme a sintaxe da linguagem. Para isto, o analisador utiliza uma estrutura em árvore, denominada árvore de derivação, que descreve as construções da linguagem reconhecida pelo compilador (DELAMARO, 2004, p. 5). Conforme Price e Toscani (2001, p. 9), "Em geral, a árvore de derivação não é produzida explicitamente, mas sua construção está implícita nas chamadas de rotinas recursivas que executam a análise sintática".

O analisador semântico, por sua vez, utiliza a árvore de derivação para identificar possíveis inconsistências nas construções utilizadas pelo programador, tais como: variáveis não declaradas, redeclaração de variáveis, chamadas de métodos com número incorreto de parâmetros, entre outras.

Vale lembrar que durante todo o processo de análise pode haver a detecção de algum erro no programa fonte. Caso isso ocorra, mensagens claras devem ser apresentadas ao programador, para que ele tenha capacidade de identificar o problema e fazer as correções que forem necessárias (AHO et al., 2008, p. 3).

Passada a fase de análise do programa fonte, inicia-se a fase de síntese. Nesta fase é feita a geração de código intermediário que pode eventualmente ser o código objeto final. Porém, conforme afirmam Price e Toscani (2001), a tradução do código fonte para objeto em mais de uma etapa apresenta algumas vantagens, como a possibilidade de otimização do código intermediário, segunda etapa do processo de síntese. Segundo Aho et al. (2008, p. 6), "A fase de otimização de código [...] faz algumas transformações no código intermediário com o objetivo de produzir um código objeto melhor. Normalmente, melhor significa mais rápido, mas outros objetivos podem ser desejados [...]".

Por fim, é executada a última etapa do processo, a geração de código objeto. Nesta etapa o código intermediário é mapeado em uma linguagem objeto, mediante instruções de baixo nível. Price e Toscani (2001, p. 13) definem esta etapa como, "[...] a fase mais difícil, pois requer uma seleção cuidadosa das instruções e dos registradores da máquina alvo a fim de produzir código objeto eficiente".

2.2 ARDUINO

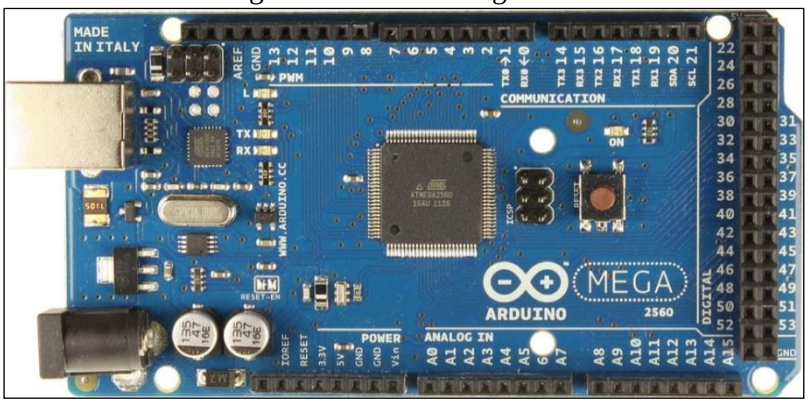
O Arduino foi criado em 2005 na Itália pelo professor Massimo Banzi e o pesquisador David Guartielles. Banzi procurava um meio barato e que tornasse fácil o trabalho com tecnologia por seus alunos de design de iteração. Em conjunto, eles resolveram criar um microcontrolador que pudesse ser utilizado nos projetos dos estudantes de arte e design.

Guartielles desenhou a placa e um aluno de Massimo, David Mellis, programou o software para executá-la. A facilidade no uso e o seu baixo custo, fez com que a primeira tiragem da plataforma fosse rapidamente vendida. Hoje já são cerca de 300 mil unidades vendidas pelos diversos distribuidores espalhados pelo mundo (EVANS; HOCHENBAUM; NOBLE, 2013, p. 25).

Atualmente, existem vários modelos de placas Arduino disponíveis no mercado. Todos os modelos consistem em uma pequena placa composta de um microprocessador Atmel AVR, um cristal ou oscilador (relógio simples que envia pulsos de tempo em uma frequência especificada, para permitir sua operação na velocidade correta) e um regulador linear de 5 volts (MCROBERTS, 2011, p. 21). Diversos dispositivos externos, como motores, relés, sensores luminosos, diodos a laser e alto-falantes podem ser conectados ao Arduino (MONK, 2013, p. 1). Para adicionar mais funcionalidades ao Arduino, pode-se ainda utilizar placas especializadas conhecidas como *shields*. As *shields* são placas de circuito contendo outros dispositivos como, por exemplo, *Global Positioning System* (GPS), *Liquid Crystal Display* (LCD), módulos de Ethernet, entre outros (MCROBERTS, 2011, p. 24).

Para o desenvolvimento deste trabalho foi utilizado o Arduino Mega 2560, modelo ilustrado na Figura 2.

Figura 2 - Arduino Mega 2560



Fonte: Silva (2013).

O Arduino Mega 2560 é composto por um microcontrolador baseado no ATmega2560. Ele possui 54 pinos digitais de entrada e saída, dos quais 15 podem ser usados como saídas *Pulse-Width Modulation* (PWM) e 16 entradas analógicas, um cristal oscilador de 16 MHz, uma conexão *Universal Serial Bus* (USB), um conector de alimentação, um barramento *In-Circuit Serial Programming* (ICSP) e um botão *reset* (BANZI et al., 2016a).

No *site* oficial da plataforma é possível baixar a *Integrated Development Environment* (IDE) do Arduino, um software livre usado para escrever e enviar os programas para a placa. Os programas, no mundo Arduino, são conhecidos como *sketches* e a programação dos

mesmos é feita utilizando Wiring, uma linguagem de programação baseada em C/C++ (BANZI et al., 2016c). Todo *sketch* deve conter as funções `void setup()` e `void loop()`, conforme mostrado no Quadro 1.

Quadro 1 - *Sketch* `Blink`

```
void setup() {
   pinMode(13, OUTPUT);
}

void loop() {
   digitalWrite(13, HIGH);
   delay(1000);
   digitalWrite(13, LOW);
   delay(1000);
}
```

Fonte: Banzi et al. (2016a).

O *sketch* `Blink`, mostrado no Quadro 1, é um dos exemplos disponibilizados pela IDE do Arduino para compreensão dos principais comandos da linguagem. Na função `void setup()` é feita a configuração do pino 13 da placa como saída e a função `void loop()` contém os comandos `digitalWrite(13, HIGH)`, que acende um *Light Emitting Diode* (LED), e `digitalWrite(13, LOW)`, que apaga o LED. Já a função `delay(1000)` é necessária para que as mudanças de estado do pino 13 sejam percebidas, fazendo com que o programa aguarde 1 segundo após cada instrução. Resumidamente, o objetivo deste *sketch* é fazer um LED piscar.

2.3 ROBOTOY

O Robotoy foi desenvolvido em 2014 (TORRENS, 2014). O objetivo da ferramenta é permitir que crianças possam desenvolver programas para a plataforma Lego Mindstorms NXT. A programação do robô é feita de forma textual, porém, segundo Torrens (2014, p. 13), "[...] utilizando uma linguagem simplificada, para que o indivíduo se concentre na solução de um problema específico, relevando detalhes da linguagem de programação e do ambiente de desenvolvimento".

O Quadro 2 mostra um exemplo de algoritmo elaborado utilizando a linguagem Robotoy. O algoritmo tem como objetivo resolver o seguinte problema: o robô inicia na coluna 2 do cenário e deve atravessá-lo, chegando na coluna 6, sendo que ele sempre deve terminar na linha que começou. É possível que o robô encontre um obstáculo durante a travessia. Neste caso, deverá desviar ao passar pelo obstáculo (TORRENS, 2014, p. 68). Na Figura 3 pode-se observar dois cenários possíveis de serem resolvidos com o algoritmo proposto no Quadro 2.

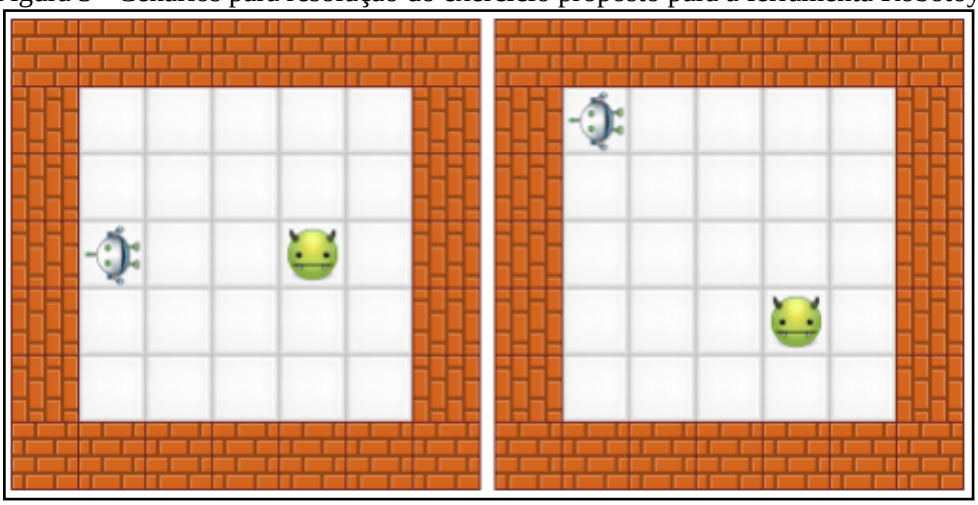
Quadro 2 - Algoritmo elaborado na linguagem Robotoy

```
número células Percorridas <- 0
enquanto não tem obstáculo
   andar para frente 1
   células Percorridas <- células Percorridas + 1
fim do enquanto

se células Percorridas < 4
   virar para a direita
   andar para frente
   virar para a esquerda
   andar para frente 2
   virar para a esquerda
   andar para frente
   virar para a direita
fim do se
```

Fonte: Torrens (2014, p. 68).

Figura 3 - Cenários para resolução do exercício proposto para a ferramenta Robotoy



Fonte: Torrens (2014, p. 68).

A linguagem também foi planejada para não ser exclusiva para a plataforma Lego Mindstorms NXT. Por isso, disponibiliza uma interface que determina o que é um robô suportado (TORRENS, 2014, p. 39), ou seja, define quais as ações que o robô é capaz de executar. Apesar disso, não existem restrições quando aos componentes utilizados na montagem do robô Lego Mindstorms NXT, sendo o único item obrigatório o *brick*. No entanto, para que um robô suporte todos os comandos oferecidos pela linguagem, ele deve possuir os seguintes componentes: *display*, caixa de som, sensor ultrassônico, sensor de cor, rodas e motor. Se o usuário utilizar um comando que dependa de um componente específico e o mesmo não esteja conectado ao robô, será apresentada uma mensagem de erro no momento em que a instrução for executada.

2.4 TRABALHOS CORRELATOS

Nesta seção são descritos três trabalhos correlatos a este: o FURBOT, uma biblioteca de apoio ao ensino de lógica de programação; o *plugin* RoboMind FURB, desenvolvido para ser utilizado no ambiente RoboMind; e, por fim, o Cubetto, um brinquedo que objetiva

ensinar programação para crianças e foi construído utilizando placas compatíveis com Arduino.

2.4.1 FURBOT

O FURBBOT foi criado em 2008, no âmbito do Departamento de Sistemas e Computação (DSC) da Universidade Regional de Blumenau (FURB). O projeto consiste em um ambiente de apoio ao ensino de lógica de programação com forte apelo na área de jogos. A ideia é que o aluno desenvolva algoritmos de controle de personagens de tal forma a criar uma atmosfera facilitadora do aprendizado (VAHLDICK et al., 2008).

Para utilizar o FURBOT basta criar um projeto Java em um ambiente de programação como o Eclipse ou o Netbeans, por exemplo. Em seguida, deve-se adicionar ao projeto a biblioteca do FURBOT e outras três bibliotecas auxiliares, necessárias para o seu funcionamento. Além disso, é preciso copiar para o projeto o arquivo *eXtensible Markup Language* (XML) referente ao exercício a ser solucionado. A ideia é que este XML seja criado pelo professor e contenha as configurações do cenário de acordo com os objetivos do exercício. Feito isto, o aluno deve criar uma classe com o mesmo nome do arquivo XML referente à atividade, sendo que esta classe deve estender a classe `br.furb.furbot.Furbot`.

Na Figura 4 pode ser observado o ambiente Eclipse devidamente configurado para execução de um exercício. Ao executar a classe `Exercicio`, será gerado o cenário ilustrado na Figura 5 e o aluno deve então criar a lógica para resolução do problema dentro do método `void inteligência()`. No Quadro 3 pode ser observada uma das soluções possíveis para resolver o problema ilustrado na Figura 5.

Figura 4 - Ambiente Eclipse configurado para uso do FURBOT

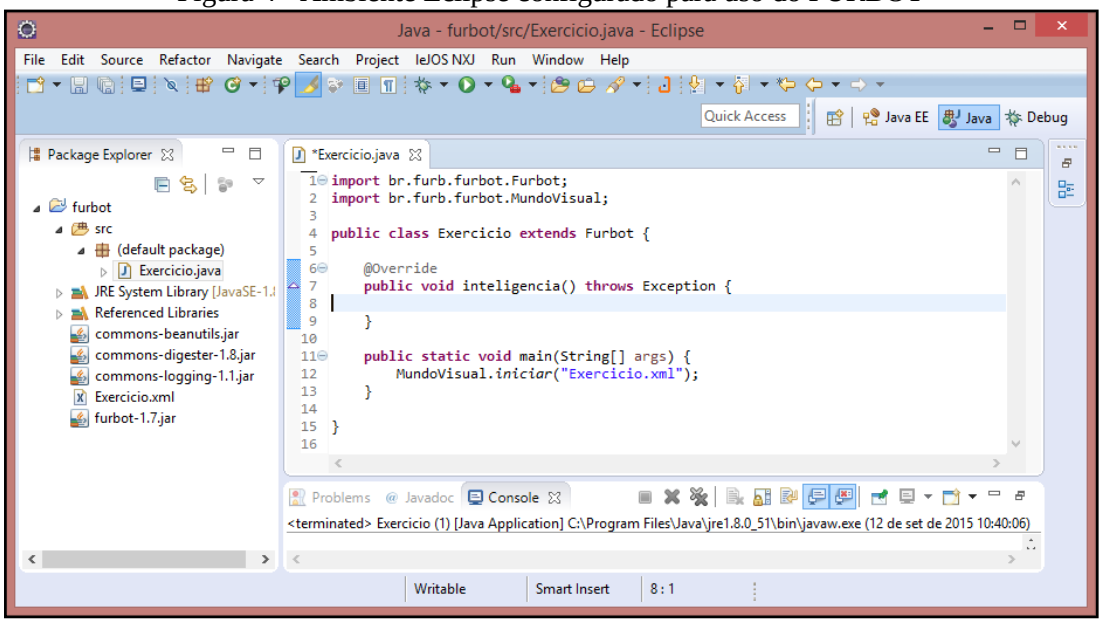
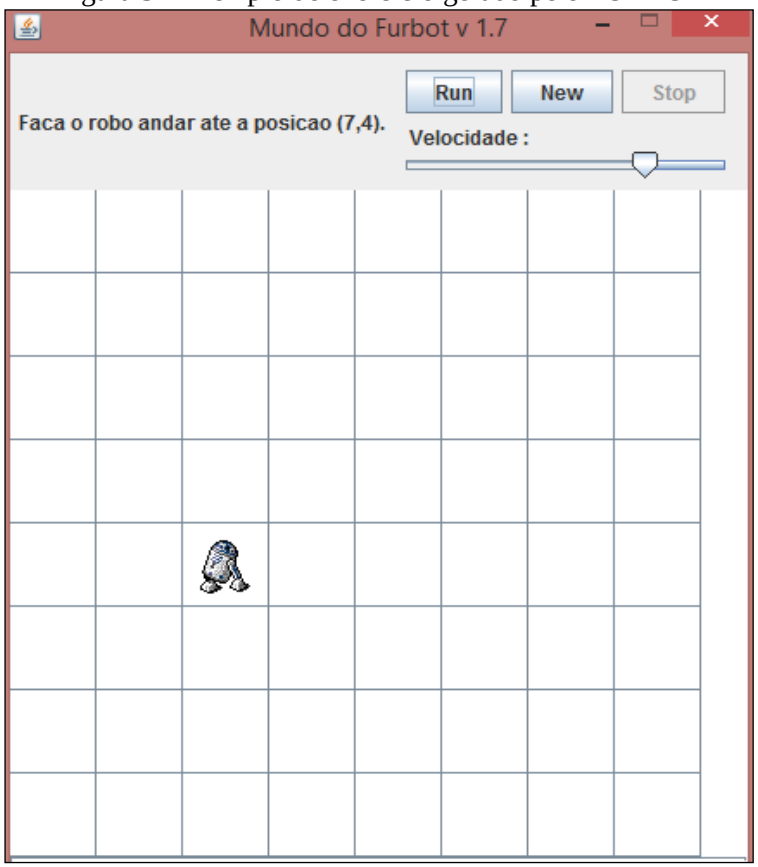


Figura 5 - Exemplo de exercício gerado pelo FURBOT



Quadro 3 - Solução do exercício proposto

```
@Override
public void inteligencia() throws Exception {
   andarDireita();
   andarDireita();
   andarDireita();
   andarDireita();
   andarDireita();
}
```

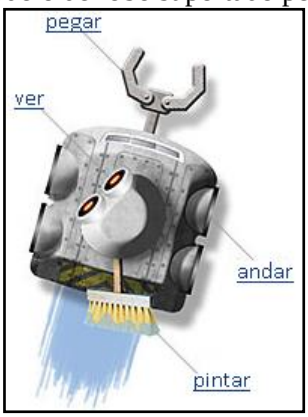
Conforme pode ser visto, a linguagem utilizada pela ferramenta é bastante simples e de fácil entendimento, ideal para o aprendizado de lógica de programação, embora envolva código de programação em Java.

2.4.2 RoboMind FURB

O RoboMind FURB é um *plugin* também desenvolvido no DSC da FURB para ser utilizado no RoboMind. O RoboMind é um ambiente de programação de robôs *open source* criado por Arvid Halma, da Universidade de Amsterdam. Ele foi inspirado na linguagem Logo e permite o aprendizado dos conceitos básicos de programação de computadores. (BENITTI et al., 2008).

A Figura 6 mostra um esboço de como seria um robô suportado pela ferramenta, identificando os componentes necessários para execução de operações básicas como: andar, pegar, ver e pintar objetos.

Figura 6 - Modelo de robô suportado pelo RoboMind



Fonte: Benitti et al. (2008).

Algumas das instruções básicas para programar um robô no RoboMind podem ser observadas no Quadro 4.

Quadro 4 - Instruções básicas do RoboMind

```
andarFrente(n)
andarTrás(n)
virarEsquerda()
virarDireita()
pintarBranco()
pararPintar()
pegar()
soltar()
```

Com a utilização do *plugin* desenvolvido na FURB, é possível testar um *script* no RoboMind e posteriormente enviá-lo para um robô montado com o *kit* Lego Mindstorms NXT. No Quadro 5 podem ser observadas algumas funções que são adicionadas ao RoboMind com a utilização do RoboMind FURB.

Quadro 5 - Instruções disponibilizadas pelo RoboMind FURB

```
inc(var, n)
dec(var, n)
iguais(var, n)
diferentes(var, n)
menorIgual(var, n)
maiorIgual(var, n)
```

As funções mostradas acima realizam operações sobre variáveis a partir de um valor passado como parâmetro (BENITTI et al., 2008).

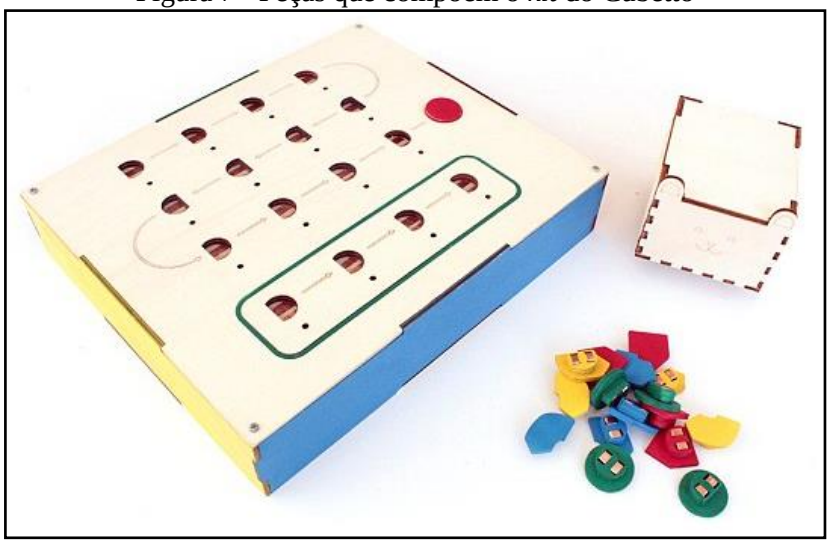
2.4.3 Cubetto

O Cubetto é um brinquedo que visa ensinar lógica de programação para crianças. Ele foi projetado por Matteo Loglio, que hoje é co-fundador e diretor de design da Primo, empresa que produz o brinquedo. O conceito é inspirado na obra de Seymour Papert. Pode-se dizer que o brinquedo é uma simplificação do ambiente Logo tradicional, criado por Papert, composto pela linguagem Logo e uma tartaruga gráfica que responde aos comandos do

usuário. No entanto, as instruções do Cubetto foram desenvolvidas de forma a evitar qualquer tipo de texto ou linguagem numérica (PRIMO, 2015a).

O *kit* do Cubetto contém (Figura 7): um tabuleiro; blocos de madeira, que devem ser encaixados ao tabuleiro e representam as instruções `frente`, `esquerda`, `direita` e `função`; e um robô motorizado que é movido à bateria e se conecta sem fio ao tabuleiro para executar as instruções.

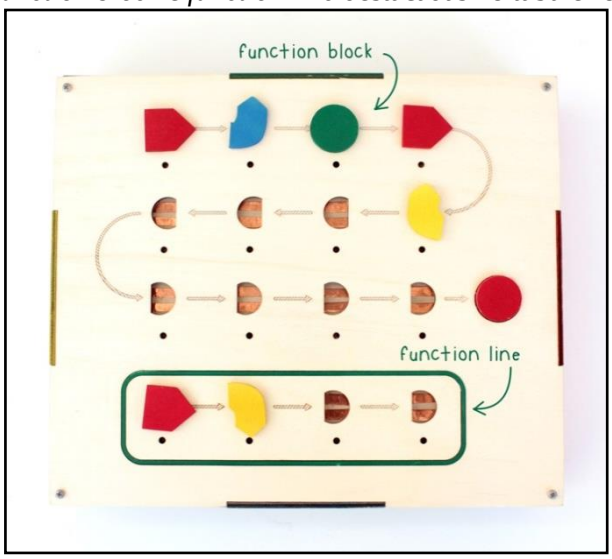
Figura 7 - Peças que compõem o *kit* do Cubetto



Fonte: Ventura (2014).

O Cubetto possui uma área chamada *function line*. Nesta área, pode-se guardar uma sequência de instruções específica e depois para ativá-la basta inserir no tabuleiro o bloco correspondente à instrução *function block*, que é representado por um círculo verde, conforme ilustrado na Figura 8.

Figura 8 - *Function block* e *function line* destacadas no tabuleiro do Cubetto



Fonte: Ventura (2014).

Pode-se observar que o brinquedo aborda de forma simples os conceitos básicos de programação, de modo que até mesmo uma criança que não sabe ler e escrever pode fazer uso dele. Outro detalhe é que o Cubetto foi construído com base em placas compatíveis com Arduino e no *site* do fabricante é disponibilizada uma documentação completa para qualquer pessoa interessada em desenvolver seu próprio protótipo do brinquedo. De acordo com a documentação, o Cubetto é composto por duas placas, sendo uma compatível com os modelos de Arduino Uno e Leonardo, utilizada para construção do robô, e uma placa compatível com o Arduino Mega 2560, que compõe o tabuleiro (PRIMO, 2015b).

3 DESENVOLVIMENTO DA FERRAMENTA

Este capítulo está dividido em seis seções. A primeira seção apresenta os requisitos da ferramenta. A segunda, a montagem, especificação e implementação do robô Arduino. A terceira e a quarta seção trazem, respectivamente, a especificação da linguagem e a especificação da ferramenta. Na quinta seção, tem-se a implementação, onde são descritas as técnicas e ferramentas utilizadas, bem como, a operacionalidade da implementação. E na sexta e última seção, apresentam-se os resultados dos experimentos realizados.

3.1 REQUISITOS PRINCIPAIS DA FERRAMENTA

A ferramenta descrita neste trabalho deve:

a) gerar código para Arduino a partir dos comandos da linguagem Robotoy (Requisito Funcional - RF);
b) disponibilizar uma tela para que o usuário escolha para qual plataforma será gerado o programa: Lego Mindstorms NXT ou Arduino (RF);
c) disponibilizar uma tela de configuração para o robô Arduino (RF);
d) fazer o envio do programa traduzido para o Arduino (RF);
e) executar os programas em robôs montados com base no Arduino (Requisito Não Funcional - RNF);
f) ser implementada utilizando a linguagem de programação Java (RNF);
g) ser desenvolvida no Eclipse (RNF).

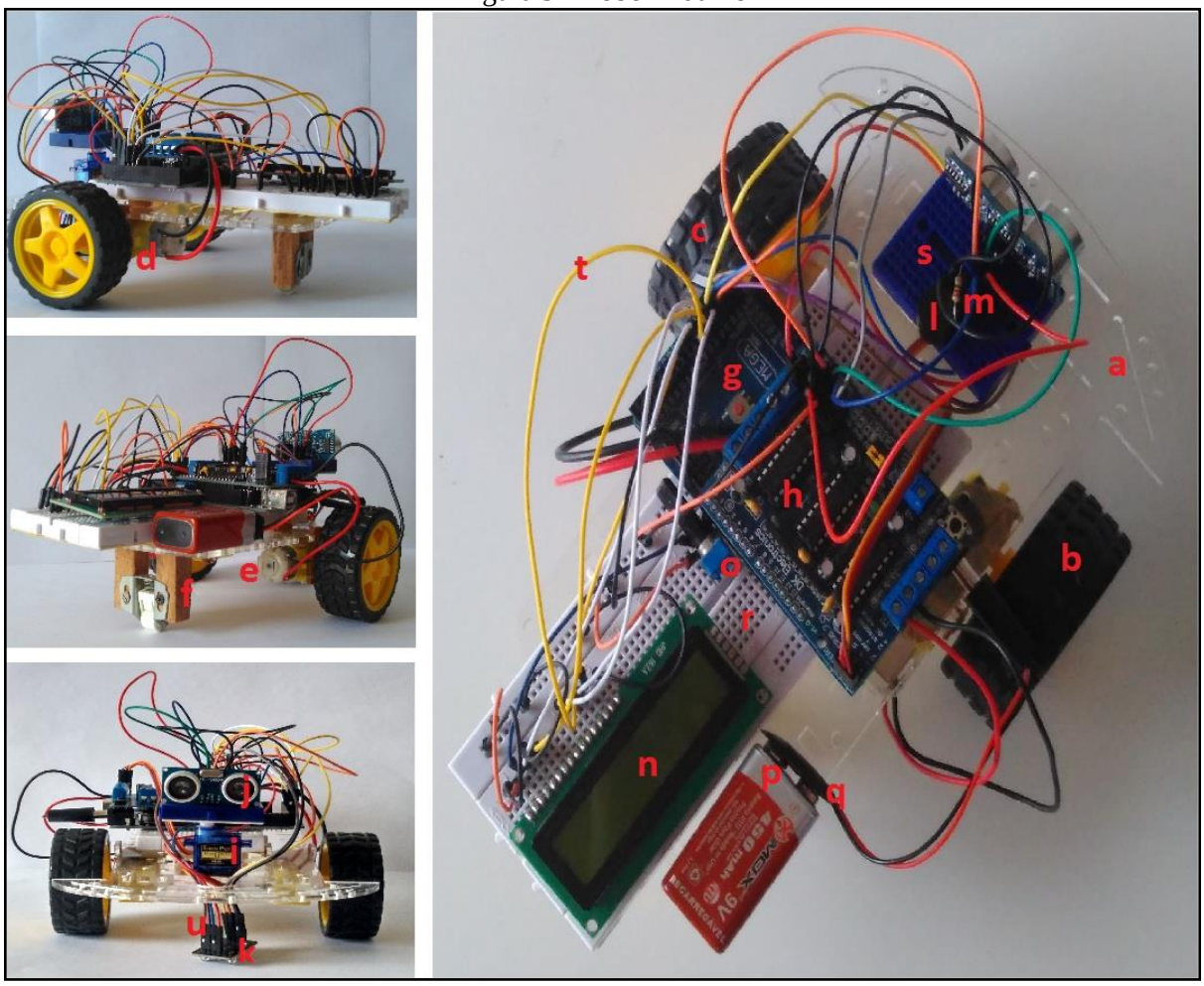
3.2 ROBÔ ARDUINO

Esta seção apresentada a montagem, a especificação e a implementação do robô Arduino.

3.2.1 Montagem do robô

O modelo de robô que foi montado, visualizado na Figura 9, é capaz de executar todos os comandos disponibilizados pela linguagem Robotoy. Porém, conforme especificado por Torrens (2014), existem vários modelos de robôs suportados, sendo o único componente obrigatório na montagem, o responsável pela inteligência do robô. No caso do robô Arduino, a própria placa é a responsável pela inteligência do robô, ou seja, é nela que todos os outros componentes devem estar conectados para receberem os comandos que devem executar.

Figura 9 - Robô Arduino



Na Figura 9 é possível observar todos os itens que compõem o robô, são eles: (a) base de acrílico; (b) roda direita; (c) roda esquerda; (d) motor esquerdo; (e) motor direito; (f) roda de sustentação do robô, sem tração; (g) Arduino Mega 2560; (h) Motor Shield L293D Ponte H; (i) micro servo motor; (j) sensor de distância ultrassônico HC-SR04; (k) sensor de cores RGB TCS230; (l) *buzzer*; (m) resistor de 220 ohms; (n) *display* LCD 16x2; (o) potenciômetro de 10K; (p) bateria de 9V; (q) *clip* para bateria de 9V com *plug* P4; (r) *protoboard* de 840 pontos; (s) mini *protoboard* de 170 pontos; (t) 25 fios *jumpers* macho/macho e (u) 8 fios *jumpers* macho/fêmea. Mais detalhes sobre ao funcionamento dos componentes utilizados, bem como o esquema com as conexões realizadas podem ser vistos no Apêndice A.

3.2.2 Especificação e implementação do robô

Segundo Torrens (2014), para que um robô seja suportado pela linguagem Robotoy ele deve implementar a interface `IRobot`. A interface `IRobot` é responsável por implementar outras interfaces, as quais definem os métodos relacionados às ações do robô, como: andar, parar, emitir som, detectar objetos, entre outras.

Ainda, segundo Torrens (2014), utilizando a máquina Virtual HaikuVM é possível programar o Arduino usando a linguagem Java. Com base nisto, a proposta inicial deste trabalho era fazer um estudo da máquina virtual HaikuVM para programar o Arduino usando Java. Porém, a linguagem nativa do Arduino é Wiring, uma linguagem baseada em C/C++ e que possui uma série de bibliotecas desenvolvidas para trabalhar com sensores, motores e demais componentes utilizados neste trabalho. Sendo assim, para programar o Arduino em Java, seria necessário buscar as correspondências de todos os comandos e bibliotecas da linguagem do Arduino para a linguagem Java. Diante disto, foi decidido fazer a tradução da linguagem Robotoy diretamente para a linguagem do Arduino.

Desta forma, ao invés de criar uma classe Java que implementa a interface `IRobot`, foi criado um *sketch*, programa em Arduino, com todas as funções necessárias para o funcionamento do robô. Este arquivo foi denominado `methods-robot-arduino.ino` e deve estar no mesmo diretório onde será gerado o programa traduzido para a linguagem do Arduino a partir do programa escrito em Robotoy. Assim, quando o programa for compilado através da IDE do Arduino, que é chamada via linha de comando pela ferramenta Robotoy, as funções correspondentes às ações do robô serão reconhecidas. Destaca-se ainda que o nome de todas as funções e variáveis globais relativas às ações do robô iniciam com *underline*, como pode ser visto no Quadro 6 que traz dois exemplos de ações do robô.

Quadro 6 - Exemplos de ações do robô Arduino

```
void _emitirSom() {
  tone(_BUZZER, 300, 300);
  digitalWrite(_BUZZER, HIGH);
  delay(500);
  digitalWrite(_BUZZER, LOW);
  delay(500);
}

void _pararAndar(){
  _motorEsquerdo.run(RELEASE);
  _motorDireito.run(RELEASE);
  delay(1000);
}
```

Este padrão é necessário para que não haja conflito entre o código criado pelo programador e as rotinas correspondentes às ações do robô, já que a linguagem Robotoy não permite identificadores que iniciem com *underline*. As funções do *sketch* `methods-robot-arduino.ino` (assinatura, descrição e componentes necessários) podem ser vistas no Quadro 7.

Quadro 7 - Funções correspondentes às ações do robô

assinatura	descrição	componente(s)
`void _escrever(String texto)`	Escreve o texto no *display*.	*display* lcd
`void _emitirSom()`	Emite um som.	*buzzer*
`void _andarFrente (int qtde = 1)` `void _andarTras (int qtde = 1)`	Movimenta o robô para frente ou para trás, respectivamente. Caso não seja informada a `qtde`, movimenta o robô apenas uma célula para frente ou para trás.	motor esquerdo motor direito
`void _virarDireita (int qtde = 1)` `void _virarEsquerda (int qtde = 1)`	Vira o robô para a direita ou para a esquerda (`qtde*`90°), respectivamente. Caso não seja informada a `qtde`, vira o robô apenas 90º para a direita ou para a esquerda.	motor esquerdo motor direito
`void _girarRodaDireitaFrente()` `void _girarRodaDireitaTras()`	Gira a roda direita do robô para frente ou para trás, respectivamente, até que seja chamada a função `pararGirar()`.	motor direito
`void _girarRodaEsquerdaFrente()` `void _girarRodaEsquerdaTras()`	Gira a roda esquerda do robô para frente ou para trás, respectivamente, até que seja chamada a função `pararGirar()`.	motor esquerdo

`void _pararAndar()`
`void _pararGirar()`
`void _pararVirar()`
	Para de movimentar o robô.	motor esquerdo
		motor direito

`boolean _temObstaculo()`	Verifica se existe algum obstáculo a uma distância de até 15 centímetros do robô.	sensor de distância ultrassônico
`String _identificarCor()`	Identifica as cores: vermelho, verde, azul, amarelo, preto e branco.	sensor RGB
`void _virarMotorMultiusoDireita(int qtde)` `void _virarMotorMultiusoEsquerda(int qtde)`	Vira o motor multiuso (responsável por movimentar o sensor de distância ultrassônico) para a direita ou para a esquerda (`qtde*`90°), respectivamente.	micro servo motor
`void _setarVelocidadeDeslocamento()`	Para a movimentação do robô e seta a velocidade de descolamento para 150 PWM.	motor esquerdo motor direito
`void _setarVelocidadeRotacao()`	Para a movimentação do robô e seta a velocidade de rotação para 130 PWM.	motor esquerdo motor direito
`void _calcularDistanciaObjeto`	Calcula a distância do robô de um possível objeto (obstáculo).	sensor de distância ultrassônico
`void _lerSensorDistancia`	Lê o sensor de distância ultrassônico.	sensor de distância ultrassônico

3.3 ESPECIFICAÇÃO DA LINGUAGEM

De acordo com Torrens (2014), os comandos da linguagem Robotoy podem ser classificados como: declaração e uso de variáveis, expressões, comandos de controle de fluxo, declaração e invocação de rotinas e comandos direcionados ao robô. Os comandos direcionados ao robô podem ainda ser divididos em subcategorias, são elas: comunicação, responsável pela escrita e emissão de som; detecção, responsável pela identificação de cores e obstáculos; e movimentação, responsável pela movimentação do robô e do sensor de distância ultrassônico.

A seguir são apresentados os comandos pertencentes à linguagem Robotoy e a sua correspondência na linguagem do Arduino, gerados pela ferramenta implementada. No Quadro 8 podem ser observados exemplos da declaração e uso de variáveis e expressões.

Quadro 8 - Variáveis e expressões em Robotoy x variáveis e expressões na linguagem do Arduino

`(cor | número | texto) <nome> <- <valor>`

Definição:

Declara uma variável de um determinado `<tipo>` com `<nome>` e `<valor>` inicial. O tipo da variável pode ser: `cor`, `número` ou `texto`.

Exemplo em Robotoy:

```
cor minhaCor <- preta
número valor <- 0
texto msg <- "Hello, Robotoy"
```

Código correspondente em Arduino:

```
String minhaCor;
double valor;
String msg;

minhaCor = preto;
valor = 0;
msg = "Hello, Robotoy";
```

`<nome> <- <valor>`

Definição:

Atribui um valor à variável sendo que este valor deve ser compatível com o tipo da variável, anteriormente declarada. O `<valor>` pode ser uma cor, um número ou um literal, outra variável ou uma expressão aritmética ou literal.

Exemplo em Robotoy:

```
minhaCor <- branca
valor <- valor + 1
msg <- "Hello!"
```

Código correspondente em Arduino:

```
minhaCor = branco;
valor = valor + 1;
msg = "Hello!";
```

`<expressão aritmética>`

Definição:

Relaciona os números e variáveis numéricas através das operações aritméticas que, podem ser: soma (`+`), subtração (`-`), multiplicação (`*`), divisão (`/`) e resto (`%`). Também pode-se utilizar parênteses para definir a prioridade das operações.

Exemplo em Robotoy:

```
valor <- (10 + 5) * 2
```

Código correspondente em Arduino:

```
valor =  (int) (10 + 5) * 2;
```

`<expressão literal>`

Definição:

Une dois ou mais valores independentemente do seu tipo.

Exemplo em Robotoy:

```
msg <- "Valor total: " . valor
```

Código correspondente em Arduino:

```
msg = "Valor total: " + (String)
valor;
```

`<expressão lógica ou relacional>`

Definição:

As operações lógicas podem ser dos tipos: `e`, que retorna verdadeiro se todos os operandos forem verdadeiros; `ou`, que retorna verdadeiro se um dos operandos for verdadeiro; e `não`, que retorna verdadeiro se o operando for falso e vice-versa. Já as operações relacionais fazem a comparação entre dois ou mais valores através dos sinais: maior (>), menor (<), maior ou igual (>=), menor ou igual (<=), igual (=) e diferente (=/).

Exemplo em Robotoy:

```
se media >= 6 e faltas <= 25
  msg <- "Aluno aprovado"
senão
  msg <- "Aluno reprovado"
fim do se

se idade <= 10 ou idade >= 65
  msg <- "Meia entrada"
senão
  msg <- "Inteira"
fim do se

se não tem obstáculo
  andar para frente
fim do se
```

Código correspondente em Arduino:

```
if (media >= 6 && faltas <= 25) {
    msg = "Aluno aprovado";
} else {
    msg = "Aluno reprovado";
}

if (idade <= 10 || idade >= 65) {
    msg = "Meia entrada";
} else {
    msg = "Inteira";
}

if (!_temObstaculo()) {
    _andarFrente();
}
```

No Quadro 9 são apresentados os comandos correspondentes ao controle de fluxo.

Quadro 9 - Comandos de controle de fluxo

`se <condição> <lista de comandos> senão <lista de comandos> fim do se`

Definição:

Verifica o valor de uma condição. Se a condição for verdadeira, os comandos subsequentes são executados. Se a condição for falsa, os comandos subsequentes à cláusula `senão` são executados, sendo que o `senão` é opcional.

Exemplo em Robotoy:

```
se valor = 0
  valor <- valor + 1
senão
  valor <- valor - 1
fim do se
```

Código correspondente em Arduino:

```
if (valor == 0) {
  valor = valor + 1;
} else {
  valor = valor - 1;
}
```

`enquanto <condição> <lista de comandos> fim do enquanto`

Definição:

Verifica o valor de uma condição. Se a condição for verdadeira, os comandos subsequentes são executados repetidas vezes até que a condição se torne falsa.

Exemplo em Robotoy:

```
enquanto valor <= 10
  valor <- valor + 1
fim do enquanto
```

Código correspondente em Arduino:

```
while (valor <= 10) {
  valor = valor + 1;
}
```

No Quadro 10 são apresentados exemplos de declaração e invocação de rotinas.

Quadro 10 - Declaração e invocação de rotinas

`declaração e invocação de rotinas`

Definição:

Quando um trecho de código se repete várias vezes ao longo do programa é comum incluir este trecho de código em uma rotina e no decorrer do programa fazer a chamada da rotina pelo seu nome.

Exemplo em Robotoy:

```
moverRobo

rotina moverRobo
  enquanto não tem obstáculo
    andar para frente
  fim do enquanto
  desviarParaDireita
  desviarParaEsquerda
fim da rotina

rotina desviarParaDireita
  virar motor multiuso para a direita
1
  se não tem obstáculo
    virar para a direita
    andar para frente
    virar para a esquerda
    andarParaFrente
  fim do se
fim da rotina

rotina desviarParaEsquerda
  virar motor multiuso para a
esquerda 1
  se não tem obstáculo
    virar para a esquerda
    andar para frente
    virar para a direita
    andarParaFrente
  fim do se
fim da rotina
```

Código correspondente em Arduino:

```
moverRobo();

void moverRobo() {
  while (!_temObstaculo()) {
    _andarFrente();
  }
  desviarParaDireita();
  desviarParaEsquerda();
}

void desviarParaDireita() {
  _virarMotorMultiusoDireita(1);
  if (!_temObstaculo()) {
    _virarDireita();
    _andarFrente();
    _virarEsquerda();
    _andarParaFrente();
  }
}

void desviarParaEsquerda() {
  _virarMotorMultiusoEsquerda(1);
  if (!_temObstaculo()) {
    _virarEsquerda();
    _andarFrente();
    _virarDireita();
    _andarParaFrente();
  }
}
```

No Quadro 11 podem ser observados os comandos de comunicação do robô.

Quadro 11 - Comandos de comunicação do robô

`escrever <texto>`

Definição:

Escreve um texto no *display*, sendo que `<texto>` deve ser um valor literal, uma variável literal, ou uma expressão literal.

Exemplo em Robotoy:

```
escrever "Hello, Robotoy"
```

Código correspondente em Arduino:

```
_escrever("Hello, Robotoy");
```

`emitir som`

Definição:

Emite um som através do *buzzer*.

Exemplo em Robotoy:

```
emitir som
```

Código correspondente em Arduino:

```
_emitirSom();
```

No Quadro 12 são apresentados exemplos da utilização dos comandos de detecção.

Quadro 12 - Comandos de detecção do robô

`cor identificada`

Definição:

Detecta cores através do sensor de cores RGB TCS230. É capaz de detectar as cores: amarelo, azul, branco, verde, vermelho e preto. A função `cor identificada` da linguagem Robotoy retorna um objeto do tipo `cor`. Já a função correspondente na linguagem do Arduino retorna uma `String`, que pode ser comparada com o valor das constantes `amarelo`, `azul`, `branco`, `verde`, `vermelho` e `preto`.

Exemplo em Robotoy:

```
número qtdeCelulasVermelhas <- 0
cor umaCor <- azul
umaCor <- cor identificada
se umaCor = vermelho
  qtdeCelulasVermelhas <- qtdeCelulasVermelhas + 1
 fim do se
```

Código correspondente em Arduino:

```
double qtdeCelulasVermelhas;
String umaCor;

qtdeCelulasVermelhas = 0;
umaCor = azul;
umaCor = _identificarCor();
if (umaCor.equals(vermelho)) {
    qtdeCelulasVermelhas = qtdeCelulasVermelhas + 1;
}
```

`tem obstáculo | não tem obstáculo`

Definição:

Identifica obstáculos através do sensor de distância ultrassônico HC-SR04. É capaz de identificar os obstáculos que estejam a uma distância de até 15 centímetros do robô. Retorna um valor do tipo `boolean`.

Exemplo em Robotoy:

```
se não tem obstáculo
  andar para frente
fim do se
```

Código correspondente em Arduino:

```
if (!_temObstaculo) {
  _andarFrente();
}
```

No Quadro 13 são apresentados os comandos responsáveis pela movimentação do robô e do sensor de distância ultrassônico.

Quadro 13 - Comandos de movimentação do robô

`andar para (frente | trás) <quantidade>`

Definição:

Movimenta o robô para frente ou para trás uma determinada `quantidade` de vezes. Caso não seja informada a `quantidade`, movimenta o robô apenas uma célula.

Exemplo em Robotoy:

```
se não tem obstáculo
  andar para frente
fim do se
```

Código correspondente em Arduino:

```
if (!_temObstaculo) {
  _andarFrente();
}
```

`virar para (esquerda | direita) <quantidade>`

Definição:

Vira o robô para o lado esquerdo ou para o lado direito em um ângulo de 90º uma determinada `quantidade` de vezes. Caso não seja informada a `quantidade`, vira o robô 90º para o lado apenas uma vez.

Exemplo em Robotoy:

```
se tem obstáculo
  virar para a direita
  andar para frente
  virar para a esquerda
fim do se
```

Código correspondente em Arduino:

```
if (_temObstaculo()) {
  _virarDireita();
  _andarFrente();
  _virarEsquerda();
}
```

`girar roda (esquerda | direita) para (frente | trás)`

Definição:

Gira a roda esquerda ou direita do robô para frente ou para trás até que seja chamada uma das seguintes funções: `parar de andar`, `parar de virar` ou `parar de girar`.

Exemplo em Robotoy:

```
se não tem obstáculo
  girar roda direita para frente
  girar roda esquerda para frente
fim do se
parar de girar
```

Código correspondente em Arduino:

```
if (!_temObstaculo) {
  _girarRodaDireitaFrente();
  _girarRodaEsquerdaFrente();
}
_pararGirar();
```

`parar de (andar | virar | girar)`

Definição:

Desliga os motores esquerdo e direito do robô para parar a movimentação.

Exemplo em Robotoy:

```
se não tem obstáculo
  girar roda direita para trás
  girar roda esquerda para trás
fim do se
parar de girar
```

Código correspondente em Arduino:

```
if (!_temObstaculo) {
  _girarRodaDireitaTras();
  _girarRodaEsquerdaTras();
}
_pararGirar();
```

`virar motor multiuso para (esquerda | direita) <quantidade>`

Definição:

Vira o servo motor, responsável por girar o sensor de distância ultrassônico, 90º para a esquerda ou para a direita uma determinada `quantidade` de vezes.

Exemplo em Robotoy:

```
virar motor multiuso para a direita 1
se não tem obstáculo
  andar para frente
fim do se
```

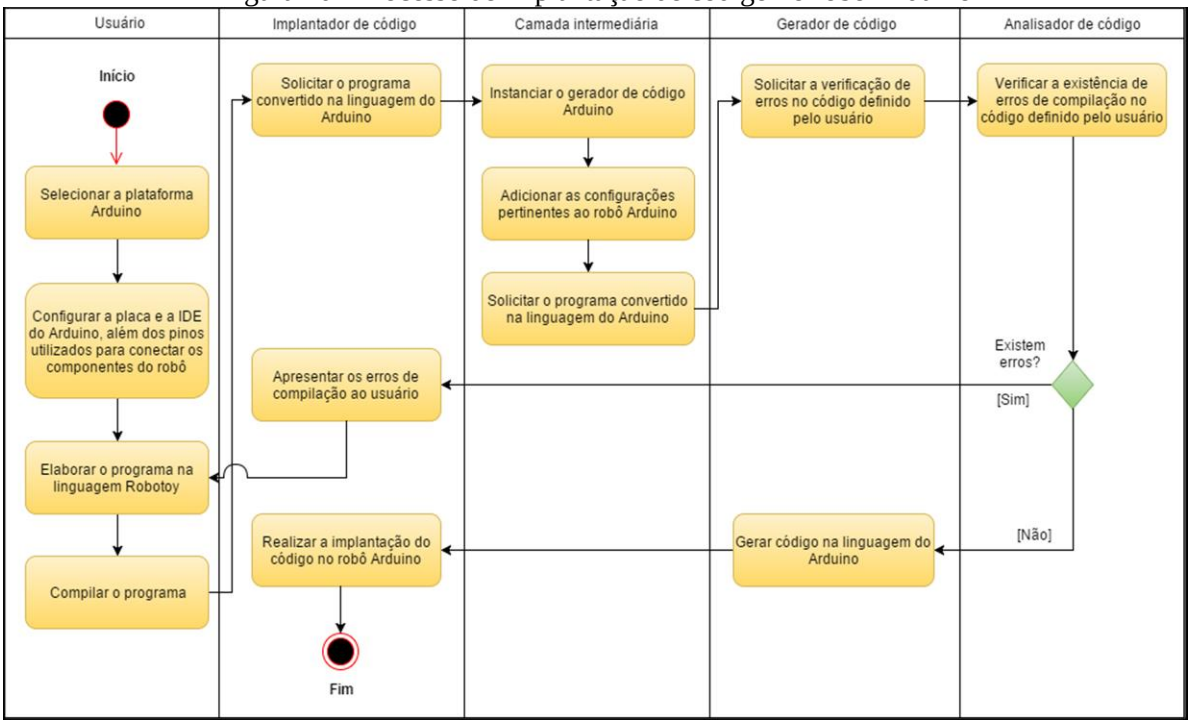
Código correspondente em Arduino:

```
_virarMotorMultiusoDireita(1);
if (!_temObstaculo) {
  _andarFrente();
}
```

3.4 ESPECIFICAÇÃO DA FERRAMENTA

Para mostrar o processo de implantação de código no robô Arduino foi especificado o diagrama de atividades, apresentado na Figura 10. Conforme pode ser visto no diagrama, o primeiro passo deste processo é a escolha da plataforma para o qual será gerado o código, que deve ser Arduino Mega. Em seguida, o programador deve abrir a tela de configuração, própria para a plataforma selecionada, e especificar em quais pinos estão conectados os sensores, os motores e os demais componentes do robô. Além disso, também deve ser informado o diretório de instalação da IDE do Arduino, o modelo e a porta na qual a placa está conectada. Feito isto, basta escrever o programa na linguagem Robotoy e compilá-lo.

Figura 10 - Processo de implantação de código no robô Arduino



Inicia-se então o processo de implantação de código, onde o implantador deve consultar o arquivo de propriedades para identificar qual a plataforma selecionada e chamar o seu respectivo integrador. O integrador faz parte da camada intermediária e é responsável por adicionar o código específico para a plataforma selecionada, no caso do Arduino o integrador é o `ArduinoIntegrator`. Além disso, o `ArduinoIntegrator` também instancia o gerador de código, que solicita aos analisadores léxico, sintático e semântico a verificação de erros no programa definido pelo programador. Assim, caso seja encontrado algum erro, o mesmo é apresentado no *console* da ferramenta. Caso contrário, é iniciada a geração de código para Arduino, feita através da classe `ArduinoClassGenerator`. Por fim, após o programa ter sido traduzido da linguagem Robotoy para a linguagem do Arduino, inicia-se o processo de implantação do código no robô. Este processo é feito com o auxílio da IDE do Arduino, que é chamada via linha de comando através da ferramenta Robotoy.

No Quadro 14 pode ser visto um exemplo de programa escrito na linguagem Robotoy e o programa equivalente na linguagem do Arduino, gerado pela ferramenta, onde destaca-se o código adicionado pelo integrador. Nas linhas 1, 2, 3 e 4 encontram-se os *includes* necessários para o funcionamento do programa conforme os componentes configurados. Nas linhas 5 e 6 tem-se a declaração das constantes que representam respectivamente os pinos *trigger* e *echo* do sensor de distância ultrassônico. Na linha 7 tem-se a declaração de uma variável global que representa o micro servo motor, utilizado para movimentar o sensor de distância ultrassônico. Na linha 8 tem-se a declaração da variável `_anguloSensor` que define

o ângulo ao qual o sensor de distância ultrassônico deve estar posicionado. Nas linhas 9 e 10 tem-se a declaração de variáveis globais referentes aos motores das rodas esquerda e direita do robô. Na linha 11 tem-se a declaração de uma variável global que representa o *display* LCD. E, por fim, na linha 12, tem-se a declaração da variável `_ligado`. Esta variável é utilizada na função `loop()` para que o código desenvolvido pelo programador, gerado dentro do função `loop()`, seja executado apenas uma vez. Isto é necessário, pois como o próprio nome da função sugere, ela é executado repetidas vezes enquanto a placa Arduino permanecer conectada a uma fonte de energia. No código exibido no Quadro 14 foram omitidas as variáveis globais correspondentes aos componentes do robô que não foram configurados. O código completo correspondente ao programa pode ser visto no Apêndice B.

Quadro 14 - Programa na linguagem Robotoy e sua equivalência na linguagem do Arduino

Programa em Robotoy

```
1  número passos <- 0
2  enquanto não tem obstáculo
3    andar para frente
4    passos <- passos + 1
5  fim do enquanto
6  texto mensagem <- "Nro de passos: " . passos
7  escrever mensagem
```

Programa na linguagem do Arduino

```
1  #include <Ultrasonic.h>
2  #include <Servo.h>
3  #include <AFMotor.h>
4  #include <LiquidCrystal.h>
5  #define _HC_SR04_TRIGGER A2
6  #define _HC_SR04_ECHO A3
7  Servo _motorMultiuso;
8  int _anguloSensor = 90;
9  AF_DCMotor _motorEsquerdo(4);
10 AF_DCMotor _motorDireito(1);
11 LiquidCrystal _lcd(43, 44, 45, 42, 46, 41);
12 boolean _ligado = true;
13 double passos;
14 String mensagem;
15 void setup() {
16   pinMode(_HC_SR04_TRIGGER, OUTPUT);
17   pinMode(_HC_SR04_ECHO, INPUT);
18   _motorMultiuso.attach(10);
19   _motorMultiuso.write(_anguloSensor);
20   _setarVelocidadeDeslocamento();
21   _lcd.begin(16, 2);
22   delay(1000);
23 }
24 void loop() {
25   while (_ligado){
26     passos = 0;
27     while (!_temObstaculo()) {
28        _andarFrente();
29        passos = passos + 1;
30     }
31     mensagem = "Nro de passos: " + (String) passos;
32     _escrever(mensagem);
33     _ligado = false;
34   }
35 }
```

Ressalta-se que, no código apresentado no Quadro 14, todas as variáveis já são inicializadas no momento da sua declaração com os pinos aos quais os componentes que elas representam estão conectados, com exceção da variável `_motorMultiuso` que é inicializada na função `setup()`, da variável `_anguloSensor` que é do tipo `int`, com valor inicial `90`, e da variável `_ligado` que é do tipo `boolean`, cujo valor inicial é `true`. Depois da declaração das variáveis e constantes, são adicionadas mais algumas configurações pelo integrador, dentro da função `setup()`: na linha 16 é feita a configuração do pino *trigger* do sensor de distância ultrassônico como sendo um pino de saída de dados; na linha 17 é configurado o pino *echo* do sensor como um pino de entrada de dados; na linha 18 é indicado o pino ao qual o micro servo motor está conectado; e, em seguida, na linha 19 é enviado um ângulo para inicializar a posicionamento do mesmo, possibilitando a orientação do robô; na linha 20 é chamada a função responsável por setar a velocidade dos motores esquerdo e direito; na linha 21 é inicializado o *display* LCD, indicando que o mesmo é composto por dezesseis colunas e duas linhas; na linha 22 é chamada a função `delay`, que recebe `1000` como parâmetro, fazendo com que a próxima instrução seja executada após um segundo. Além das configurações inseridas pelo integrador, observa-se o código definido pelo programador. Nas linhas 13 e 14 são declaradas duas variáveis e, a partir da linha 25, dentro da função `loop()`, observa-se o código principal do programa.

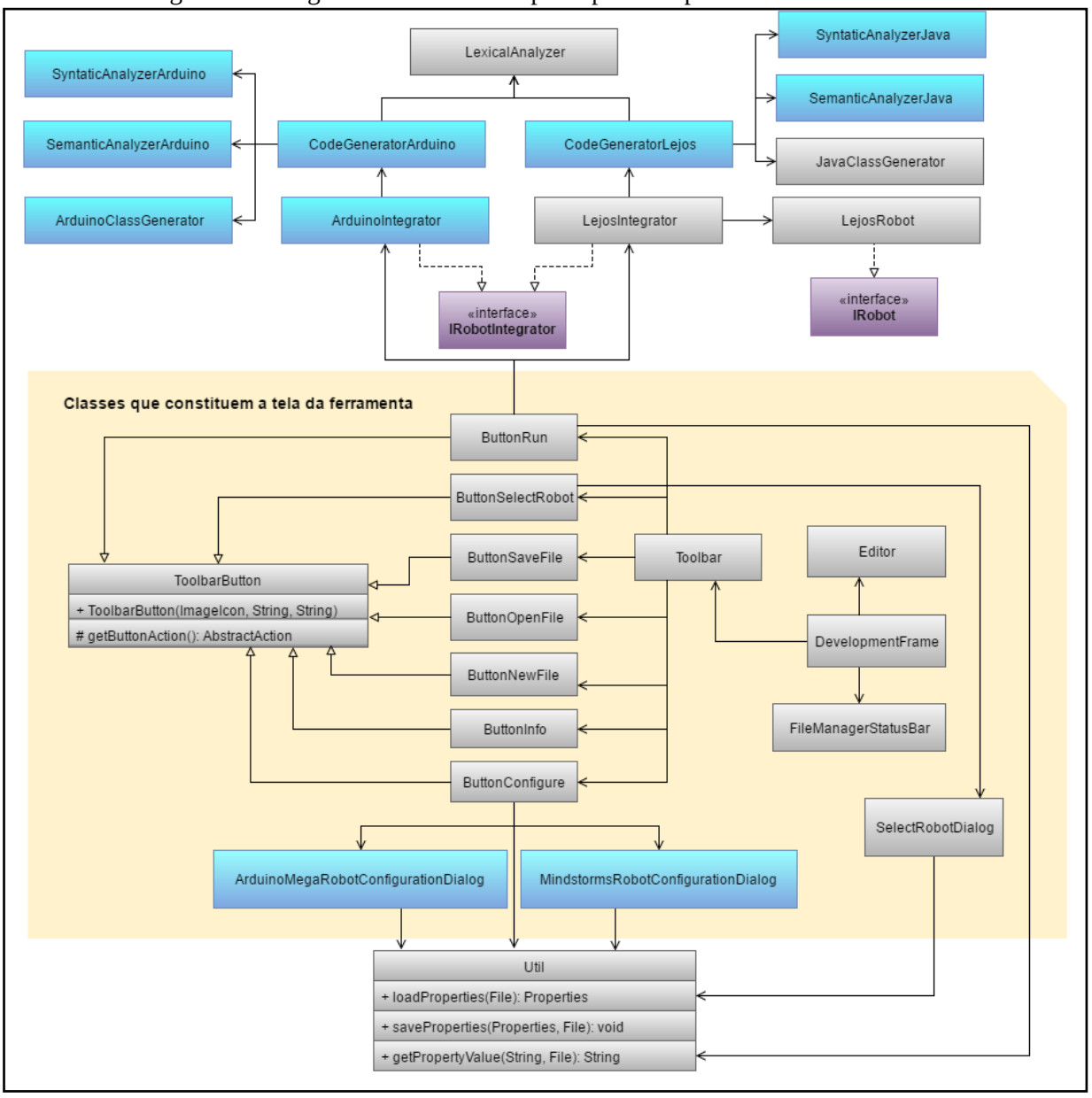
No diagrama apresentado na Figura 11, podem ser observadas as principais classes que compõem a ferramenta, destacando-se as classes que foram adicionadas ou modificadas para inclusão da plataforma Arduino, que apresentam-se na cor azul.

Observa-se, na área em amarelo do diagrama, as classes responsáveis pela interface da ferramenta, sendo a classe `DevelopmentFrame` a classe principal, que relaciona-se com as classes `Editor`, `FileManagerStatusBar` e `Toolbar`. A classe `Toolbar`, por sua vez, relaciona-se com as classes correspondentes aos botões disponíveis na ferramenta, como a classe `ButtonConfigure`. A classe `ButtonConfigure`, de acordo com a plataforma selecionado para geração de código, instancia a classe correspondente à tela de configuração da plataforma, que pode ser: a classe `ArduinoMegaConfigurationDialog` ou a classe `MindstormsConfigurationDialog`. Já a classe `ButtonRun`, instancia o integrador, de acordo com a plataforma selecionada.

Para os robôs do tipo Arduino, o integrador é representado pela classe `ArduinoIntegrator` e para os robôs do tipo Lego, o integrador é representado pela classe `LejosIntegrator`. Estes dois integradores implementam a interface `IRobotIntegrator` e

instanciam o gerador de código próprio para a plataforma pela qual são responsáveis, sendo que cada gerador de código instancia as classes referentes às análises sintática e semântica de acordo com a linguagem para o qual geram código, além da classe responsável pela análise léxica, única para as duas plataformas. Por fim, o gerador de código instancia a classe que gera o arquivo final, com o programa traduzido. No caso do Arduino a classe é `ArduinoClassGenerator` e no caso do Java a classe é `JavaClassGenerator`.

Figura 11 - Diagrama de classes dos principais componentes da ferramenta



3.5 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas no desenvolvimento deste trabalho. Assim como, também são apresentadas as principais telas da ferramenta desenvolvida, a fim de demonstrar a operacionalidade da implementação.

3.5.1 Técnicas e ferramentas utilizadas

A ferramenta Robotoy foi desenvolvida utilizando a linguagem de programação Java através da IDE Eclipse. Para construir a interface, foi utilizada a biblioteca `Swing` do Java. Também foi necessário fazer a instalação do *plugin* leJOS, pois alguns projetos que constituem a ferramenta, dependem do leJOS para realizar a implantação do código nos robôs Lego Mindstorms NXT. Já para realizar a implantação do código nos robôs Arduino foi necessário instalar a IDE do Arduino.

Com a IDE do Arduino é possível compilar e enviar os programas para a placa, que pode ser feito através da ferramenta ou a partir de linha de comando. Esta última opção foi a utilizada para compilar e enviar os programas, através da ferramenta Robotoy, para o Arduino. O Quadro 15 mostra os comandos e parâmetros utilizados nesta operação.

Quadro 15 - Comandos utilizados para compilar e enviar o programa para a placa Arduino

```
cd C:\Arduino-1.5.6
arduino --board arduino:avr: mega  --port  COM4  --upload C:\robotoy\gen\gen.ino
```

Como pode ser observado, na primeira linha, tem-se o comando para entrar no diretório de instalação da IDE do Arduino. Já na segunda linha, tem-se o comando responsável por compilar e enviar o programa para a placa, sendo necessários os seguintes parâmetros: modelo da placa utilizada (`mega`); porta a qual a placa está conectada (`COM4`); e o *sketch* a ser compilado e enviado (`C:\robotoy\gen\gen.ino`). O *sketch* gerado pelo compilador é salvo sempre no mesmo diretório, com o mesmo nome. Já o diretório da IDE do Arduino, a placa utilizada e a porta a qual a placa está conectada são configuráveis.

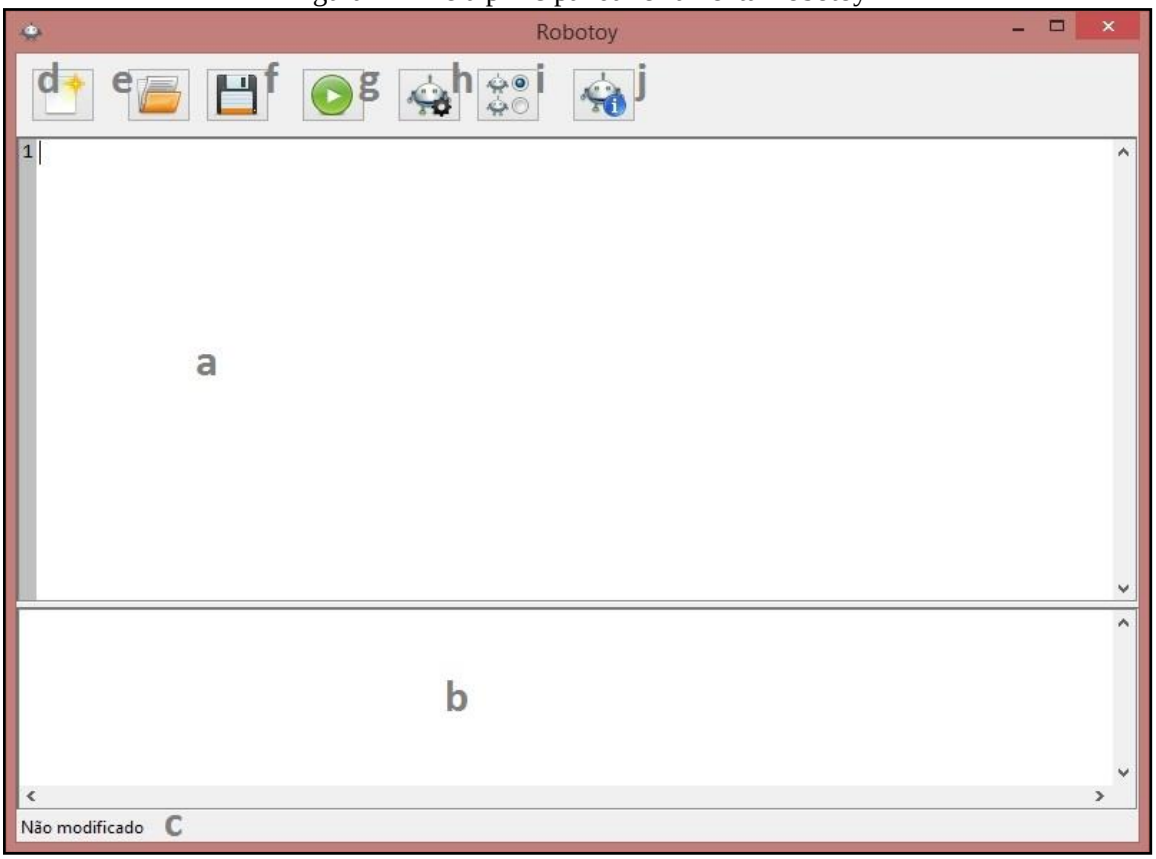
Além disso, a IDE do Arduino traz as principais bibliotecas da linguagem, necessárias para o funcionamento dos programas. Das bibliotecas utilizadas para programar o robô, apenas as seguintes não acompanham a instalação da IDE: `AFMotor`, utilizada para controlar os motores DC; e `Ultrassonic`, utilizada para controlar o sensor de distância ultrassônico.

3.5.2 Operacionalidade da implementação

A interface da ferramenta Robotoy, como mostra a Figura 12, conta com: (a) uma área para edição do código, com linhas numeradas para facilitar a descoberta de erros nos programas; (b) um console, onde são apresentadas mensagens referentes à compilação do programa; (c) uma barra de *status*, que apresenta o *status* do programa (Não modificado ou Modificado); (d) botão `Novo`, responsável por limpar o editor de código; (e) botão `Abrir`, que possibilita carregar um programa salvo com a extensão `.toy`; (f) botão `Salvar`, que salva o

programa usando a extensão `.toy`; (g) botão `Compilar`, que compila e envia o programa para o robô; (h) botão `Configurações`, que abre a tela de configurações própria para a plataforma selecionada; (i) botão `Selecionar robô`, que abre a tela de seleção do tipo de plataforma para o qual será gerado código; e (j) botão `Sobre`.

Figura 12 - Tela principal da ferramenta Robotoy



Na Figura 13 é apresentada a tela de seleção do tipo do robô para o qual se deseja gerar código. Atualmente, a ferramenta dispõe de duas opções, são elas: Lego Mindstorms e Arduino Mega.

Figura 13 - Tela para seleção do tipo do robô

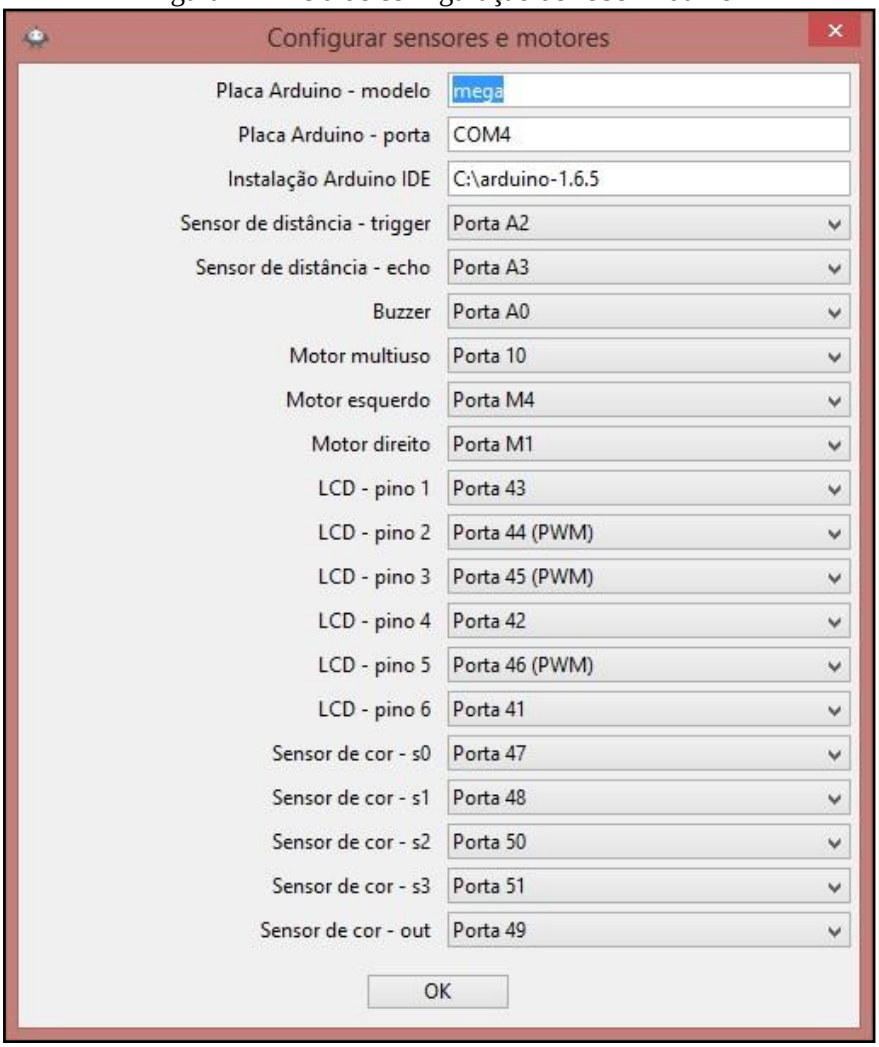
Esta informação é gravada em um arquivo de propriedades, que contém, além do tipo do robô, o integrador e o caminho do arquivo de configurações do robô selecionado, como mostra o Quadro 16.

Quadro 16 - Conteúdo do arquivo de propriedades

```
robot.type=arduino
robot.integrator=br.inf.furb.tcc.robotoy.integrator.arduino.ArduinoIntegrator
robot.configuration.file=C\:\\Users\\Juliana\\robotoy\\robotoy-
ui\\..\\gen\\arduino-mega-robot.properties
```

Na Figura 14 pode ser observada a tela de configuração própria para o robô do tipo Arduino. Nela é possível configurar o modelo e a porta na qual a placa Arduino está conectada. Além disso, também é informado o diretório de instalação da IDE do Arduino e os pinos aos quais os componentes que constituem o robô estão conectados, sendo que os pinos disponibilizados para cada componente devem ser compatíveis com os mesmos. Por exemplo, os motores esquerdo e direito, só podem ser conectados nos pinos M1, M2, M3 e M4 da motor *shield*, sendo assim, apenas estas opções são disponibilizadas para preenchimento.

Figura 14 - Tela de configuração do robô Arduino



Todas estas configurações são gravadas em um arquivo denominado `arduino-mega-robot.properties`, para posteriormente serem utilizadas na geração de código. O conteúdo do arquivo de configuração do robô Arduino pode ser observado no Quadro 17.

Quadro 17 - Conteúdo do arquivo arduino-mega-robot.properties

```
directory.arduino.ide=C\:\\arduino-1.6.5
left.wheel.port=4
lcd.pin5.port=46
distance.sensor.trigger.port=A2
color.sensor.s1.port=48
right.wheel.port=1
lcd.pin6.port=41
buzzer.port=A0
color.sensor.s2.port=50
color.sensor.s3.port=51
model.arduino.board=mega
lcd.pin1.port=43
lcd.pin2.port=44
port.arduino.board=COM4
distance.sensor.echo.port=A3
lcd.pin3.port=45
color.sensor.out.port=49
lcd.pin4.port=42
multiuse.motor.port=10
color.sensor.s0.port=47
```

Destaca-se que a tela de configuração dos robôs, tanto do tipo Arduino como do tipo Lego Mindstorms, é acessada através do mesmo botão `Configurações`, sendo que, no momento em que o usuário clica no botão, é consultado o arquivo de propriedades. Assim, é possível verificar qual o tipo de robô selecionado e abrir a tela de configuração própria para o modelo em questão. O trecho de código que faz a consulta do tipo de robô no arquivo de propriedades pode ser visto no Quadro 18.

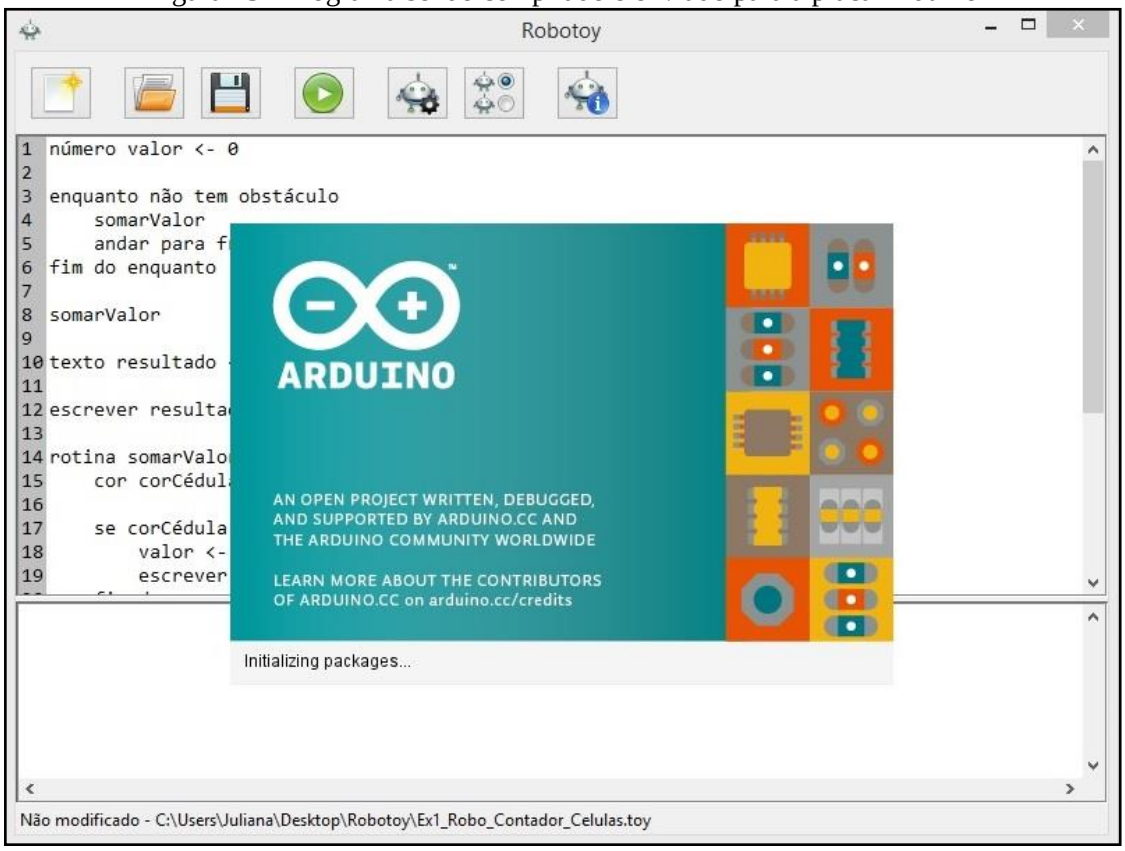
Quadro 18 - Verificação do tipo de robô selecionado para abertura da tela de configuração apropriada

```
if (robotType == RobotType.LEGO_MINDSTORMS) {
  mindstormsConfigurationDialog = new MindstormsRobotConfigurationDialog();
} else if (robotType == RobotType.ARDUINO_MEGA) {
  arduinoMegaConfigurationDialog = new ArduinoMegaRobotConfigurationDialog();
}
```

Também é feita a verificação do tipo de robô selecionado no momento de compilar o programa, para que o seu respectivo integrador seja chamado, dando início ao processo de geração de código. É importante lembrar que é necessário conectar a placa Arduino ao computador antes de compilar o programa, no caso do robô selecionado pertencer a esta plataforma.

Tendo sido feita a geração de código, é iniciado o processo de implantação do programa no robô. E, como já foi mencionado, para os robôs do tipo Arduino, este processo é feito através da IDE do Arduino que é chamada via linha de comando através da ferramenta Robotoy. Como pode ser observado na Figura 15, durante a compilação do programa, a IDE do Arduino não chega a ser aberta, apenas a tela de inicialização da ferramenta, *splash screen*, é apresentada.

Figura 15 - Programa sendo compilado e enviado para a placa Arduino



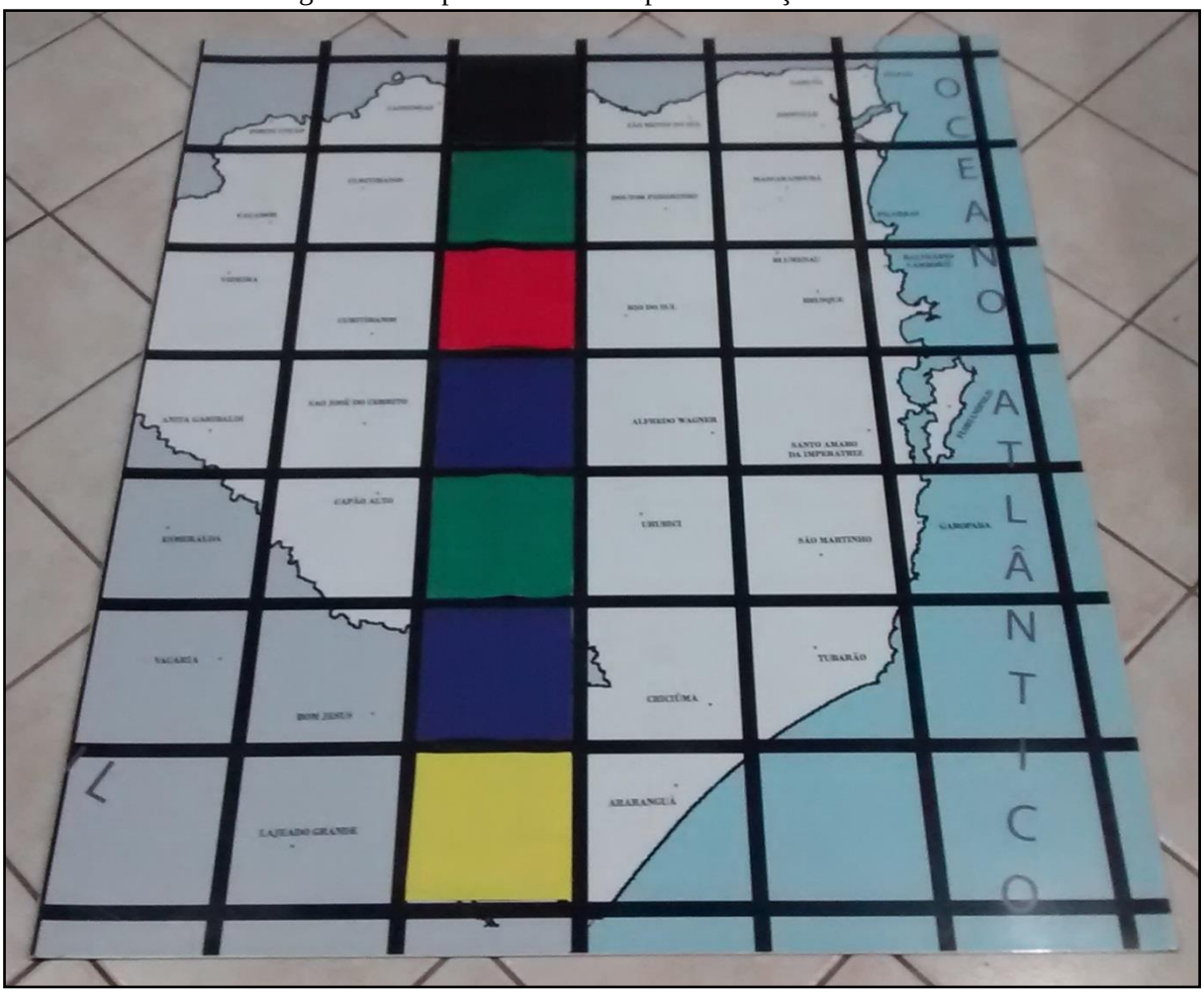
3.6 RESULTADOS E DISCUSSÕES

Para validar o funcionamento do robô, quanto às instruções disponibilizadas pela linguagem Robotoy, foi necessário definir uma superfície plana para servir como cenário para movimentação do robô. Sendo assim, o cenário ideal para validar o funcionamento do robô deve conter linhas e colunas formando um tabuleiro, onde cada célula do tabuleiro representa um passo do robô.

Portanto, inicialmente foi construído um modelo de cenário com papel e fita isolante, para demarcar as células. Porém, não foi possível deixar o papel devidamente esticado, o que formava ondulações no cenário dificultando a movimentação do robô. Então, foi feita a construção de um cenário utilizando-se como base um tecido do tipo *blackout* e fita isolante, também para demarcar as células. Mas assim como o papel, o tecido também não permanecia devidamente esticado, comprometendo a movimentação do robô. Desta forma, foi decidido utilizar uma das placas do Robolab FURB, que consiste em uma superfície lisa, ideal para a movimentação do robô. A placa utilizada pode ser vista na Figura 16, ela possui sete linhas e seis colunas, formando um tabuleiro com células de 18cm² cada. Além disso, foram fixados cartões de cores reconhecidas pela linguagem Robotoy, em algumas células do tabuleiro, de modo a disponibilizar uma área para realização de testes quanto ao reconhecimento de cores.

Destaca-se que a medida do passo do robô, bem como, a tonalidade das cores que o robô reconhece, se deu com base nas características da superfície adotada para a realização dos testes. Assim, caso o robô seja inserido em outro cenário, pode haver variações no seu comportamento.

Figura 16 - Superfície utilizada para realização de testes

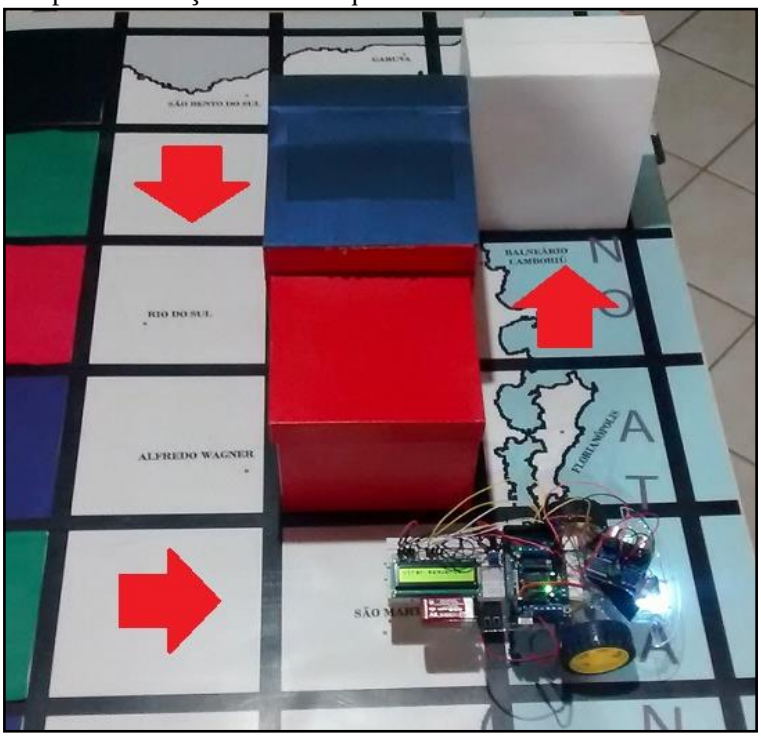


Quanto à montagem do robô, a ideia inicial era construí-lo com base na placa Arduino Leonardo, porém este modelo não possui compatibilidade com a motor *shield* utilizada para controlar os motores esquerdo e direito do robô e o micro servo motor. Também tentou-se fazer a montagem do robô com a placa Arduino Uno. Porém apesar deste modelo ser compatível com a motor *shield* utilizada, ele não apresenta um número de portas digitais de entrada e saída suficientes para conectar todos os componentes do robô simultaneamente. Assim, optou-se por utilizar a placa Arduino Mega 2560, modelo ideal para a construção de projetos de robótica, devido ao grande número de portas digitais de entrada e saída que possui, 54 no total.

Com relação aos testes realizados, um deles teve como objetivo verificar a capacidade do robô de emitir sons e desviar de obstáculos. Para isto, foram inseridas caixas no cenário,

representando os obstáculos, como mostra a Figura 17. A Figura 17 mostra também, através de setas, o percurso que o robô percorreu neste exercício, sendo que o programa desenvolvido na linguagem Robotoy para que o robô fosse capaz de realizar o percurso proposto, bem como a sua correspondência na linguagem do Arduino podem ser vistos no Quadro 19.

Figura 17 - Cenário para realização de testes quanto ao desvio de obstáculos e a emissão de sons



Quadro 19 - Programa desenvolvido para testar o desvio de obstáculos e a emissão de som

Programa em Robotoy	
1	virar motor multiuso para a esquerda 1
2	andarEnquantoTemObstaculo
3	andar para frente
4	andarEnquantoTemObstaculo
5	virar motor multiuso para a direita 1
6	andarEnquantoNaoTemObstaculo
7	emitir som
8	
9	rotina andarEnquantoTemObstaculo
10	enquanto tem obstáculo
11	andar para frente
12	fim do enquanto
13	virar para a esquerda
14	fim da rotina
15	
16	rotina andarEnquantoNaoTemObstaculo
17	enquanto não tem obstáculo
18	andar para frente
19	fim do enquanto
20	fim da rotina

Programa na linguagem do Arduino

```
1   #include <Ultrasonic.h>
2   #include <Servo.h>
3   #include <AFMotor.h>
4   #include <LiquidCrystal.h>
5   const int _s0 = 47;
6   const int _s1 = 48;
7   const int _s2 = 50;
8   const int _s3 = 51;
9   const int _out = 49;
10  const String vermelho = "vermelho";
11  const String verde = "verde";
12  const String azul = "azul";
13  const String amarelo = "amarelo";
14  const String branco = "branco";
15  const String preto = "preto";
16  #define _HC_SR04_TRIGGER A2
17  #define _HC_SR04_ECHO A3
18  #define _BUZZER A0
19  Servo _motorMultiuso;
20  int _anguloSensor = 90;
21  AF_DCMotor _motorEsquerdo(4);
22  AF_DCMotor _motorDireito(1);
23  LiquidCrystal _lcd(43, 44, 45, 42, 46, 41);
24  boolean _ligado = true;
25
26  void setup() {
27    pinMode(_s0, OUTPUT);
28    pinMode(_s1, OUTPUT);
29    pinMode(_s2, OUTPUT);
30    pinMode(_s3, OUTPUT);
31    pinMode(_out, INPUT);
32    digitalWrite(_s0, HIGH);
33    digitalWrite(_s1, HIGH);
34    pinMode(_HC_SR04_TRIGGER, OUTPUT);
35    pinMode(_HC_SR04_ECHO, INPUT);
36    pinMode(_BUZZER, OUTPUT);
37    _motorMultiuso.attach(10);
38    _motorMultiuso.write(_anguloSensor);
39    _setarVelocidadeDeslocamento();
40    _lcd.begin(16, 2);
41    delay(1000);
42  }
43  void loop() {
44    while (_ligado){
45      _virarMotorMultiusoEsquerda(1);
46      andarEnquantoTemObstaculo();
47      _andarFrente();
48      andarEnquantoTemObstaculo();
49      _virarMotorMultiusoDireita(1);
50      andarEnquantoNaoTemObstaculo();
51      _emitirSom();
52      _ligado = false;
53    }
54  }
55  void andarEnquantoTemObstaculo() {
56    while (_temObstaculo()) {
57      _andarFrente();
58    }
59    _virarEsquerda();
60  }
61  void andarEnquantoNaoTemObstaculo() {
62    while (!_temObstaculo()) {
63      _andarFrente();
64    }
65  }
```

Como mostra o Quadro 19, o primeiro comando do programa, vira o sensor de distância para o lado esquerdo (linha 1). Depois disto, é feita a chamada da função `andarEnquantoTemObstaculo` (linha 2), que faz com que o robô ande para frente enquanto houver obstáculos e, quando não houver mais obstáculos, vira o robô para o lado esquerdo. Feito isto, o robô é deslocado uma vez para frente (linha 3) e novamente é chamada a função `andarEnquantoTemObstaculo` (linha 4), de modo a fazer com que o robô contorne os obstáculos existentes. Em seguida, o sensor de distância é virado para o lado direito (linha 5) e a função `andarEnquantoNaoTemObstaculo` é chamada (linha 6), fazendo com que o robô ande para frente até encontrar um obstáculo. E, por último, é chamada a função responsável pela emissão de som (linha 7).

O teste com este programa foi executado mais de uma vez, sendo que o robô conseguiu detectar os objetos e realizar a emissão de som. Apesar disso, algumas vezes foi necessário alinhar o robô no cenário, quanto este pendia para o lado ao realizar a curva para a esquerda. Pelo que foi verificado, esta situação pode ocorrer por dois motivos. Um deles diz respeito ao estado da bateria, quando está totalmente carregada, o robô se movimenta de forma mais precisa. No entanto, quando à bateria começa a perder carga, a velocidade aplicada aos motores, bem como o tempo em que os mesmos permanecem ligados, formando um passo ou uma curva, deixam de ser suficientes para que o robô execute a ação em questão. Outro ponto a ser destacado, quanto à movimentação do robô, diz respeito ao próprio modelo de motor utilizado, pois os motores DC só podem ser controlados através de velocidade e tempo de rotação. Assim, uma alternativa para realizar a movimentação de forma mais precisa seria a utilização de um modelo de motor que permitisse realizar as rotações por meio de ângulos, como os servo motores de 360 graus ou os motores de passo.

Outro teste realizado teve o intuito de validar o funcionamento do *display* LCD e do sensor de cores RGB. Para este teste foi utilizada a área do cenário composta pelos cartões das cores reconhecidas pela linguagem. No Quadro 20 pode ser visto o programa elaborado para execução deste teste, tanto na linguagem Robotoy quanto o programa gerado na linguagem do Arduino.

Quadro 20 - Programa elaborado para realização de testes do *display* LCD e do sensor de cores RGB

Programa em Robotoy

```
cor corIdentificada <- preto

enquanto corIdentificada =/ branco
  corIdentificada <- cor identificada
  escrever corIdentificada
fim do enquanto
```

Programa na linguagem do Arduino

```
#include <Ultrasonic.h>
#include <Servo.h>
#include <AFMotor.h>
#include <LiquidCrystal.h>
const int _s0 = 47;
const int _s1 = 48;
const int _s2 = 50;
const int _s3 = 51;
const int _out = 49;
const String vermelho = "vermelho";
const String verde = "verde";
const String azul = "azul";
const String amarelo = "amarelo";
const String branco = "branco";
const String preto = "preto";
#define _HC_SR04_TRIGGER A2
#define _HC_SR04_ECHO A3
#define _BUZZER A0
Servo _motorMultiuso;
int _anguloSensor = 90;
AF_DCMotor _motorEsquerdo(4);
AF_DCMotor _motorDireito(1);
LiquidCrystal _lcd(43, 44, 45, 42, 46, 41);
boolean _ligado = true;

String corIdentificada;

void setup() {
  pinMode(_s0, OUTPUT);
  pinMode(_s1, OUTPUT);
  pinMode(_s2, OUTPUT);
  pinMode(_s3, OUTPUT);
  pinMode(_out, INPUT);
  digitalWrite(_s0, HIGH);
  digitalWrite(_s1, HIGH);
  pinMode(_HC_SR04_TRIGGER, OUTPUT);
  pinMode(_HC_SR04_ECHO, INPUT);
  pinMode(_BUZZER, OUTPUT);
  _motorMultiuso.attach(10);
  _motorMultiuso.write(_anguloSensor);
  _setarVelocidadeDeslocamento();
  _lcd.begin(16, 2);
  delay(1000);
}

void loop() {
  while (_ligado){
    corIdentificada = "preto";
    while (!corIdentificada.equals(branco)) {
       corIdentificada = _identificarCor();
       _escrever(corIdentificada);
    }
    _ligado = false;
  }
}
```

Como pode ser visto no Quadro 20, no programa desenvolvido para testar a identificação de cores e o *display* LCD, tem-se primeiramente a inicialização de uma variável do tipo `cor`, que recebe como valor inicial `preto` (linha 1). Depois, têm-se os comandos de detecção (linha 4) e impressão das cores (linha 5), que são executados até que o `branco` seja identificado.

Este teste também foi realizado várias vezes e o robô foi capaz de fazer a identificação das cores, assim como exibir a mensagem da cor identificada no *display* LCD. No entanto, percebeu-se que pode haver variação na identificação das cores dependendo da iluminação do ambiente onde for posicionado o cenário.

No Quadro 21 pode ser observado um comparativo entre os trabalhos correlatos e a ferramenta Robotoy, estendida neste trabalho.

Quadro 21 - Comparativo entre os trabalhos correlatos e a ferramenta Robotoy

características / trabalhos	FURBOT	RoboMind FURB	Cubetto	Robotoy
permite execução em robô físico		X	X	X
permite a customização do robô				X
linguagem de programação utilizada	Java	RoboMind	não contém	Robotoy
possui ambiente virtual de simulação integrado	X	X		
tipo de programação	textual	textual	encaixe de blocos	textual

Conforme mostra o Quadro 21, além da Robotoy, o Cubetto e o RoboMind FURB possibilitam a execução dos comandos disponíveis em robôs físicos, sendo que apenas a Robotoy permite a customização do robô. Com relação à linguagem de programação, a Robotoy e o RoboMind FURB se destacam pois permitem que a programação dos robôs seja feita através de uma linguagem de programação inteiramente em português, mais especificamente um português estruturado. Quanto ao ambiente virtual de simulação, apenas o FURBOT e o RoboMind FURB apresentam esta característica. No entanto, já existe um ambiente virtual desenvolvido para a ferramenta Robotoy (SILVA, 2016), porém atualmente este ainda não é integrado à ferramenta. E por fim, tanto a Robotoy como o FURBOT e o RoboMind FURB dispõem de tipo de programação textual, apenas o Cubetto difere nesta característica já que a programação é feita através do encaixe de blocos, que representam as ações do robô, em um tabuleiro.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de uma extensão da ferramenta Robotoy que teve como objetivo adicionar suporte ao Arduino, como plataforma de programação, à ferramenta.

Neste sentido, foram feitas as adaptações necessárias para a ferramenta possibilitar a configuração de um robô do tipo Arduino, bem como, a geração de código para a linguagem do Arduino a partir da linguagem Robotoy. Também foi feita a montagem de um robô com base na placa Arduino Mega 2560, sendo que o modelo de robô montado possui todos os componentes necessários para executar os comandos disponíveis na linguagem Robotoy. Apesar disso, é possível fazer a montagem de robôs mais simples, com menos componentes. No entanto, se os programas elaborados contiverem ações que dependam de componentes que não estiverem presentes na montagem do robô, estes não serão executados.

Além disso, também foi feita a implementação de um mecanismo para implantar o código gerado no robô, que é feito através da ferramenta Robotoy com o auxílio da IDE do Arduino.

Com relação aos experimentos realizados, no que diz respeito à geração, implantação e execução dos programas no robô, concluiu-se que os objetivos do trabalho foram alcançados. Embora tenham sido constatadas algumas limitações quanto à mobilidade do robô e o ambiente ao qual ele é inserido, o mesmo foi capaz de executar todos os comandos disponíveis na linguagem Robotoy. Além disso, como era esperado, a montagem de um robô com base na placa Arduino se mostrou uma alternativa mais barata para construção de robôs, se comparada a montagem de robôs utilizando-se a plataforma Lego Mindstorms NXT.

4.1 EXTENSÕES

Como sugestões para trabalhos futuros propõem-se as seguintes extensões para a ferramenta:

a) utilizar servo motores ou motores de passos para movimentar o robô, ao invés de motores DC, já que estes modelos de motor podem ser controlados a partir de ângulos, o que resultaria em movimentos mais precisos para o robô;
b) utilizar baterias em paralelo para alimentar a placa Arduino;
c) possibilitar a configuração do tamanho do passo do robô, dando a opção do programador informar, por exemplo, a velocidade dos motores e o tempo em que os mesmos permanecem ligados;

d) possibilitar a inclusão de mais componentes na montagem do robô, tornando a tela de configurações dinâmica, de modo que o programador consiga configurar um número de componentes variável;
e) disponibilizar uma diretiva para que o usuário possa adicionar código nativo do Arduino na construção dos programas da linguagem Robotoy;
f) integrar à ferramenta o simulador desenvolvido por Silva (2016);
g) elaborar uma linguagem gráfica, onde o usuário possa arrastar e soltar peças para elaborar programas, disponibilizando assim outra maneira de fazer a programação do robô, além da textual.

REFERÊNCIAS

AHO, Alfred V. et al. **Compiladores**: princípios, técnicas e ferramentas. 2. ed. São Paulo: Pearson Addison Wesley, 2008.

ARAÚJO, Daniel P.; HILDEBRAND, Hermes R. Processing e Arduino. In: SIMPÓSIO INTERNACIONAL DE GAMES, MUNDOS VIRTUAIS E TECNOLOGIAS NA EDUCAÇÃO SIMPÓSIO DE ARTES, MÍDIAS LOCATIVAS E TECNOLOGIAS NA EDUCAÇÃO, 1., 2014, Santa Maria. **Anais...** Santa Maria: UFSM, 2014. Não paginado. Disponível em: <http://pt.slideshare.net/danielpazaraujo/processing-e-arduino>. Acesso em: 29 maio 2016.

ARDUINO E CIA. **Sensor de reconhecimento de cor TCS230 / TCS3200**. [S.l.], 2014. Disponível em: <http://www.arduinoecia.com.br/2014/02/sensor-de-reconhecimento-de-cor-tcs230.html>. Acesso em: 27 maio 2016.

BANZI, Massimo et al. **Arduino Mega 2560 & Genuino Mega 2560**. [S.l.], 2016a. Disponível em: <https://www.arduino.cc/en/Main/arduinoBoardMega2560>. Acesso em: 11 maio 2016.

_____. **“Hello world”**. [S.l.], 2015. Disponível em: <https://www.arduino.cc/en/Tutorial/HelloWorld>. Acesso em: 29 maio 2016.

_____. **Motor shield**. [S.l.], 2016b. Disponível em: <http://playground.arduino.cc/Main/AdafruitMotorShield>. Acesso em: 18 maio 2016.

_____. **What is Arduino?** [S.l.], 2016c. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 23 jun. 2016.

BENITTI , Fabiane B. V. et al. **Bem vindo ao RoboLAB**: robótica educativa na FURB. [Blumenau?], [2008]. Disponível em: <http://robolab.inf.furb.br/>. Acesso em: 12 set. 2015.

BOXELECTRÓNICA. **Aprender a controlar um LCD**. [S.l.], 2015. Disponível em: <http://blog.boxelectronica.com/2015/08/29/aprender-a-controlar-um-lcd/>. Acesso em: 28 maio 2016.

BUILDBOT. **Como utilizar o sensor ultrassônico HC-SR04**. [S.l.], 2015. Disponível em: <http://buildbot.com.br/blog/como-utilizar-o-sensor-ultrasonico-hc-sr04>. Acesso em: 26 maio 2016.

CASA DAS ANTENAS. **Bateria MO-9V450 recarregável 9V 450 mAh "RTU"**. [S.l.], 2015. Disponível em: <http://www.casadaantena.com.br/bateria-mo-9v450-recarregavel-9v-450-mah-rtu.html>. Acesso em: 28 maio 2016.

DATASHEETCAFE. **SG90 Datasheet PDF:** 9g micro servo – TowerPro. [S.l.], 2015. Disponível em: <http://www.datasheetcafe.com/sg90-datasheet-pdf-9-g-micro-servo>. Acesso em: 26 maio 2016.

DELAMARO, Márcio E. **Como construir um compilador utilizando ferramentas Java**. São Paulo: Novatec, 2004.

ELETROGATE. **Jumpers**: macho/femea - 40 unidades de 20cm. [S.l.], 2016. Disponível em: <http://www.eletrogate.com/pd-11906c-jumpers-macho-femea-40-unidades-de-20-cm.html?ct=ae596&p=1&s=1>. Acesso em: 28 maio 2016.

EVANS, Martin; HOCHENBAUM, Jordan; NOBLE, Joshua. **Arduino em ação.** São Paulo: Novatec, 2013.

FREIRE, Paulo; FAUNDEZ, Antonio; COSTA, Heitor F. **Por uma pedagogia da pergunta**. 2. ed. Rio de Janeiro: Paz e Terra, 1986.

GOMES, Cristiane G. et al. A robótica como facilitadora do processo de ensino-aprendizagem de matemática no ensino fundamental. In: PIROLA, Nelson A. (Org). **Ensino de ciências e matemática IV**: temas de investigação. São Paulo: UNESP, 2010. p. 205-221. Disponível em: <http://books.scielo.org/id/bpkng/pdf/pirola-9788579830815-11.pdf>. Acesso em: 28 ago. 2015.

HERNANDEZ, Sebastian. **Chassis of a 2WD autonomous robot**. [S.l.], 2015. Disponível em: <https://grabcad.com/library/chassis-of-a-2wd-autonomous-robot-1>. Acesso em: 26 maio 2016.

MCROBERTS, Michael. **Arduino básico**. São Paulo: Novatec, 2011.

MECÂNICA INDUSTRIAL. **O que é um sensor ultrassônico**. [S.l.], 2015. Disponível em: <http://www.mecanicaindustrial.com.br/598-o-que-e-um-sensor-ultrassonico>. Acesso em: 23 jun. 2016.

MEDEIROS FILHO, Daniel A; GONÇALVES, Paulo C. Robótica educacional de baixo custo: uma realidade para as escolas brasileiras. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 28., 2008, Belém. **Anais...** Belém: SBC, 2008. p. 264-273. Disponível em: <http://www.br-ie.org/pub/index.php/wie/article/view/985/971>. Acesso em: 23 ago. 2015.

MERCADO LIVRE. **Kit jumpers macho macho Arduino pic protoboard prototipos**. [S.l.], 2016. Disponível em: <http://produto.mercadolivre.com.br/MLB-696813946-kit-jumpers-macho-macho-arduino-pic-protoboard-prototipos-_JM>. Acesso em: 28 maio 2016.

MONK, Simon. **Programação com Arduino**: começando com sketches. Porto Alegre: Bookman, 2013.

MOTA, Allan. **Usando o buzzer com Arduino**: transdutor piezo elétrico. [S.l.], 2015. Disponível em: <http://blog.vidadesilicio.com.br/arduino/basico/usando-o-buzzer-com-arduino-transdutor-piezo-eletrico>. Acesso em: 28 maio 2016.

NYPLATFORM. **L293D motor drive shield expansion board for Arduino duemilanove Mega UNO**. [S.l.], 2016. Disponível em: <http://www.nyplatform.com/index.php?route=product/product&product_id=662>. Acesso em: 28 maio 2016.

PAPERT, Seymour. **A máquina das crianças**: repensando a escola na era da informática. Porto Alegre: Artes Médicas, 1994.

PRICE, Ana M. A.; TOSCANI, Simão S. **Implementação de linguagens de programação**: compiladores. 2. ed. Porto Alegre: Luzzatto, 2001.

PRIMO. **Cubetto playset manual**. [S.l.], [2015a?]. Disponível em: <http://www.primotoys.com/docs/docs/cubetto-manual.html>. Acesso em: 12 set. 2015.

_____. **Make your Cubetto prototype**. [S.l.], [2015b?]. Disponível em: <http://www.primotoys.com/docs/docs/make.html>. Acesso em: 18 set. 2015.

SEEED DEVELOPMENT. **Arduino and color recognition sensor TCS230 TCS320**. [S.l.], 2015. Disponível em: <http://www.seeedstudio.com/recipe/209-arduino-and-color-recognition-sensor-tcs230-tcs320.html>. Acesso em: 27 maio 2016.

SILVA, Alessandro. **Conheça o Arduino Mega 2560, a mais poderosa placa Arduino.** [S.l.], 2013. Disponível em: <http://www.oarduino.com/conheca-o-arduino-mega-2560>. Acesso em: 26 maio 2016.

SILVA, Diogo. **Um simulador 2D para a linguagem Robotoy**. 2016, 54 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TECHMOUNT. **Conector de bateria de 9V com o plug 5.5mm/2.1mm**. [S.l.], 2016. Disponível em: <http://www.techmount.com.br/conector-de-bateria-de-9v-com-o-plugue-5-5mm-2-1mm>. Acesso em: 28 maio 2016.

THOMSEN, Adilson. **Controle motor DC 12V com Motor Shield L293D**. [S.l.], 2013. Disponível em: <http://blog.filipeflop.com/motores-e-servos/controle-motor-dc-arduino-motor-shield.html>. Acesso em: 28 maio 2016.

TORRENS, Maria G. **Robotoy**: ferramenta para uso de robótica no ensino de programação para crianças. 2014, 71 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://dsc.inf.furb.br/arquivos/tccs/monografias/2014_2_maria-gabriela-torrens_monografia.pdf>. Acesso em: 20 set. 2015.

VAHLDICK, Adilson et al. **Projeto FURBOT**: introdução. [Blumenau], 2008. Disponível em: <http://www.inf.furb.br/poo/furbot/introducao.html>. Acesso em: 10 set. 2015.

VALENTE, José A. **O computador na sociedade do conhecimento**. Campinas: UNICAMP-NIED, 1999.

VENTURA, Felipe. **Ensine crianças a programar usando este brinquedo**. [S.l.], [2014]. Disponível em: <http://gizmodo.uol.com.br/primo-programacao/>. Acesso em: 12 set. 2015.

APÊNDICE A – Montagem do robô

Este apêndice apresenta como funciona cada componente utilizado na montagem do robô, bem como o esquema com as ligações realizadas para conectá-los ao Arduino. Para formar a base do robô foi usado um *kit* contendo: um chassi de acrílico; dois motores DC (3~6V) com caixa de redução (1:48); duas rodas de borracha e uma roda (giratória ou boba), que foi substituída, visto que a mesma fazia com que o robô desviasse muito da sua trajetória ao andar em linha reta. Em substituição a roda boba foi utilizado um fecho de pressão para armário parafusado em dois pedaços de madeira que, por sua vez, foram fixados na base do robô de forma a dar sustentação ao mesmo, conforme mostra a Figura 9f. A Figura 18 ilustra os componentes originais do *kit* já montados.

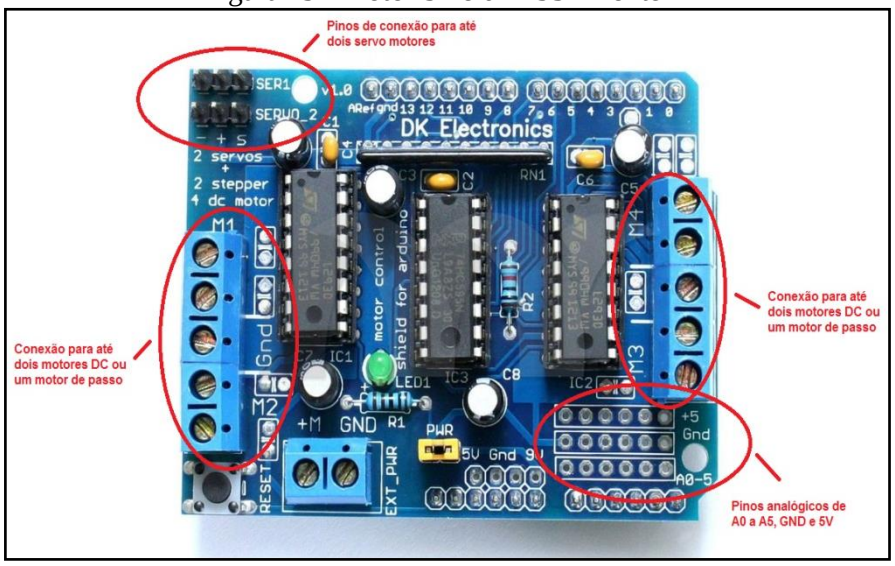
Figura 18 - Base do robô

Fonte: Hernandez (2015).

O modelo de placa Arduino utilizado para construção do robô foi o Mega 2560. Este modelo foi projetado especialmente para projetos mais complexos, como os de robótica (BANZI et al., 2016a). Por possuir uma grande quantidade de memória e de pinos de entrada e saída, se comparado às outras variações da placa, o Arduino Mega 2560 possibilita o trabalho com um maior número de componentes.

Para controlar os motores foi utilizada a Motor Shield L293D Ponte H, compatível com o Arduino Uno e Arduino Mega. Como mostra a Figura 19, com ela é possível controlar até quatro motores DC e dois servos motores ou dois motores de passo (BANZI et al., 2016b). Porém, para este trabalho foram utilizados apenas dois motores DC, um para mover a roda direita e outro para mover a roda esquerda do robô. Além disso, foi utilizado também um micro servo motor modelo 9g SG90 TowerPro para movimentar o sensor de distância ultrassônico.

Figura 19 - Motor Shield L293D Ponte H



Fonte: adaptado de NYPlatform (2016).

Os pinos utilizados para controlar os motores DC ou motores de passo são: 11, 3, 5 e 6, além dos pinos 4, 7, 8 e 12. Já os servos motores utilizam os pinos 9 (servo 1) e 10 (servo 2). Os pinos que sobram são os pinos analógicos de A0 a A5, que também podem ser utilizados como pinos analógicos de 14 a 19 (BANZI et al., 2016b) e os pinos digitais 2 e 13. Para utilizar os pinos que sobram pode-se soldar uma barra de pinos nos furos correspondentes, como mostra a Figura 19. Para o desenvolvimento deste trabalho foi utilizada uma barra de pinos fêmea cortada em três partes de seis pinos cada, soldada nos furos correspondentes aos pinos analógicos de A0 a A5, GND e 5V.

O micro servo motor 9g SG90 TowerPro, utilizado para mover o sensor de distância ultrassônico, possui ângulo de rotação de 180 graus e acompanha: um cabo de alimentação/controle, três modelos de hélices e três parafusos. Na Figura 20 pode-se observar o micro servo motor e o modelo da hélice utilizado neste trabalho.

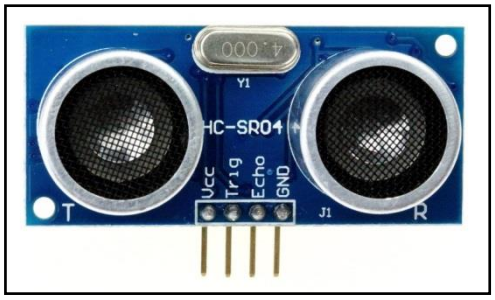
Figura 20 - Micro servo motor 9g SG90 TowerPro

Fonte: adaptado de DataSheetCafe (2015).

O sensor de distância ultrassônico utilizado foi o HC-SR04. Este sensor é capaz de medir distâncias de dois centímetros a quatro metros com precisão de três milímetros

(BUILDBOT, 2015). O HC-SR04 é composto por um circuito com emissor e receptor acoplados e quatro pinos, são eles: VCC, Trigger, Echo e GND, conforme mostra a Figura 21.

Figura 21 - Sensor de distância ultrassônico HC-SR04

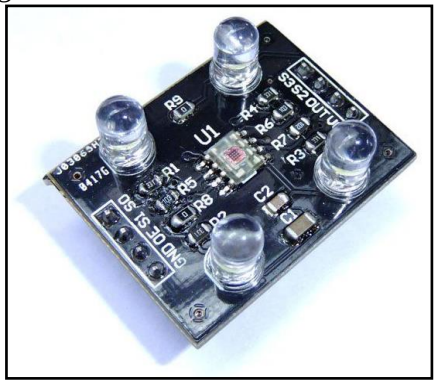


Fonte: Buildbot (2015).

O funcionamento do sensor consiste basicamente no envio e recebimento de sinais ultrassônicos. Para começar a medição, o HC-SR04 emite uma onda sonora (Trigger) que ao encontrar um objeto a rebaterá de volta ao módulo. Com base no tempo de envio e retorno do sinal, é calculada a distância entre o sensor e o obstáculo (BUILDBOT, 2015). Destaca-se que alguns tipos de materiais, como tecidos de pelúcia, por exemplo, podem absorver as ondas sonoras dificultando a identificação do obstáculo pelo sensor (MECÂNICA INDUSTRIAL, 2015).

Já para a detecção de cores foi utilizado o sensor TCS230. Este sensor é composto por 64 fotodiodos, dos quais 16 têm filtros para a cor vermelha, 16 para a cor verde, 16 para a cor azul e outros 16 não possuem filtro (ARDUINO E CIA, 2014). Conforme ilustrado na Figura 22, o TCS230 possui quatro LED*s* brancos para iluminação e 8 pinos para conexão, são eles: S0, S1, S2 e S3, que são pinos de controle, e o pino OUT que é responsável pelo envio de informações. Além disso, também possui o pino OE (*Output Enable*) e os pinos VCC e GND.

Figura 22 - Sensor de cores RGB TCS230



Fonte: Seeed Development (2015).

Para realizar a detecção de cores os pinos S0 e S1 devem ser colocados em nível alto e o estado dos pinos S2 e S3 são alterados conforme o fotodiodo que se deseja ativar

(ARDUINO E CIA, 2014). O Quadro 22 mostra as possíveis combinações de estados dos pinos S2 e S3 e os respectivos fotodiodos ativos para cada combinação.

Quadro 22 - Fotodiodos ativos conforme combinação dos pinos S2 e S3

PINO		FOTODIODO
S2	S3	
low	low	vermelho
low	high	azul
high	low	sem filtro
high	high	verde

Para a emissão de som foi utilizado um *buzzer* em conjunto com um resistor de 220 ohms. Conforme mostra a Figura 23, em um dos lados do *buzzer* existe um sinal de "+", neste lado deve ser conectado o resistor para então conectar o pino ao 5V do Arduino. O outro pino do *buzzer* deve ser conectado ao GND.

Figura 23 - Buzzer

Fonte: Mota (2015).

Outro componente utilizado na montagem do robô foi o *display* LCD 16x2 em conjunto com um potenciômetro de 10K, utilizado para o controle de brilho do *display*. Conforme mostra a Figura 24, o LCD possui 16 furos, os quais foram soldados a uma barra de pinos para possibilitar a conexão com a *protoboard*.

Figura 24 - *Display* LCD 16x2

Fonte: Banzi et al. (2015).

Os pinos do *display* LCD são numerados de 1 até 16. O último pino, à direita do componente, é o pino de número 1 e o primeiro pino, à esquerda do componente, é o de número 16. No Quadro 23 é possível observar a sigla correspondente a cada pino e a sua respectiva função.

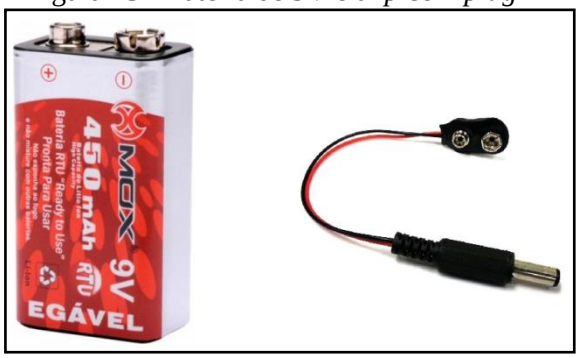
Quadro 23 - Pinos do *display* LCD 16x2

PINO	SÍMBOLO	FUNÇÃO
1	Vss	GND (alimentação)
2	Vdd	5V (alimentação)
3	V0	ajuste de contraste
4	Rs	habilita/desabilita seletor
5	R/W	leitura/escrita
6	E	habilita escrita no LCD
7	DB0	dados
8	DB1	dados
9	DB2	dados
10	DB3	dados
11	DB4	dados
12	DB5	dados
13	DB6	dados
14	DB7	dados
15	A	5V (*backlight*)
16	K	GND (*backlight*)

Fonte: BoxElectrónica (2015).

Como fonte de alimentação externa, foi utilizada uma bateria de 9V recarregável (Figura 25 – à esquerda), conectada a placa através de um *clip* para bateria de 9V com *plug* P4 (Figura 25 – à direita).

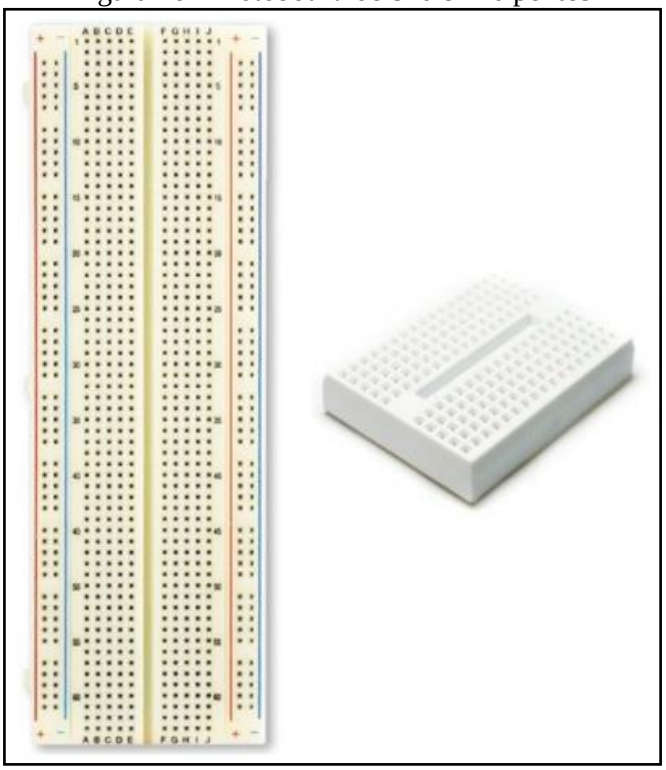
Figura 25 - Bateria de 9V e *clip* com *plug* P4



Fonte: adaptado de Casa das Antenas (2015) e Techmount (2016).

Ainda foram utilizadas na montagem do robô uma *protoboard* de 840 pontos para elaboração do circuito do *display* LCD e uma mini *protoboard* de 170 pontos para elaboração dos circuitos envolvendo o *buzzer* e o sensor de distância ultrassônico. Os dois modelos de *protoboard* utilizados podem ser vistos na Figura 26.

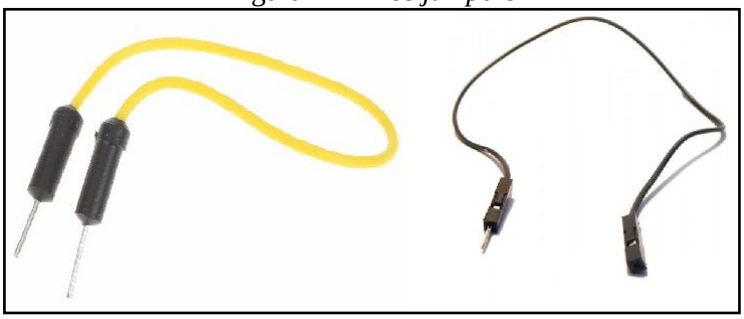
Figura 26 - *Protoboard* de 840 e 170 pontos



Fonte: adaptado de Araújo e Hildebrand (2014).

Foram utilizados também 25 fios *jumpers* macho/macho (Figura 27 – à esquerda) e 8 fios *jumpers* macho/fêmea (Figura 27 – à direita).

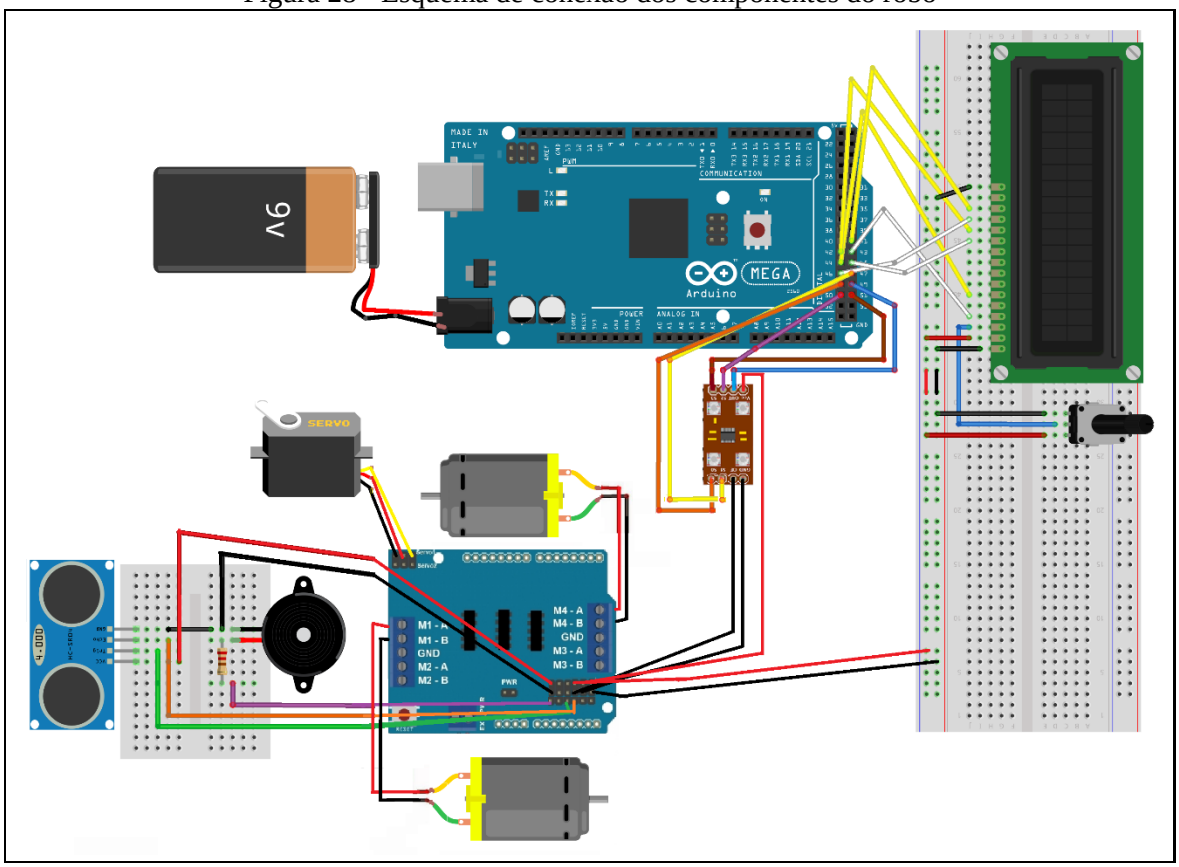
Figura 27 - Fios *jumpers*



Fonte: adaptado de Eletrogate (2016) e Mercado Livre (2016).

Na Figura 28 é possível observar o esquema com as ligações que foram feitas para conectar os componentes do robô ao Arduino Mega 2560 e a Motor Shield L293D Ponte H. Parte do esquema apresentado foi desenhado no Fritzing, ferramenta que permite a construção de protótipos de circuitos elétricos.

Figura 28 - Esquema de conexão dos componentes do robô



Fonte: adaptado de Arduino e Cia (2014) e Thomsen (2013).

O primeiro passo da montagem do robô é conectar a Motor Shield L293D Ponte H ao Arduino Mega 2560. Depois disto, pode-se acomodar a placa Arduino sob a base do robô, mostrada na Figura 18, e continuar a montagem adicionando os demais componentes.

Na Motor Shield L293D Ponte H, como mostra a Figura 28, foram conectados os seguintes componentes:

a) o micro servo motor, que antes, foi fixado na parte da frente da base do robô, como mostra a Figura 9, para depois ser conectado no pino Servo 1;

b) o motor que gira a roda esquerda, conectado no pino M4;

c) o motor que gira a roda direita, conectado no pino M1;

d) o *buzzer*, conectado primeiro a uma mini *protoboard*, sendo que o pino negativo do *buzzer* foi conectado a um dos pinos GND da *shield* (fio preto) e o pino positivo do *buzzer* foi conectado a uma das pontas do resistor de 220 ohms. Já a outra ponta do resistor foi conectada ao pino A0 da *shield* (fio roxo);

e) o sensor de distância ultrassônico HC-SR04, também foi conectado a mini *protoboard* e através de fios *jumpers* conectou-se:

- o pino VCC do sensor a um dos pinos de 5V da *shield* (fio vermelho),
- o pino Trigger do sensor ao pino A2 da *shield* (fio verde),

- o pino Echo do sensor ao pino A3 da *shield* (fio laranja),
- o pino GND do sensor a um dos pinos GND da *shield* (fio preto).

Por fim, foi retirado o adesivo da parte debaixo da mini *protoboard* e esta foi fixada a hélice de duas pontas do micro servo motor, mostrada na Figura 20. E, desta forma, é possível fazer a movimentação do sensor ultrassônico para detecção dos obstáculos a sua volta.

Já no Arduino Mega, como mostra a Figura 28, foram conectados os seguintes componentes:

a) o *display* LCD 16x2, que foi conectado primeiro a uma *protoboard* de 480 pontos, sendo que:
   - o pino 1, VSS, foi conectado ao GND do Arduino (fio preto),
   - o pino 2, VDD, foi conectado ao 5V do Arduino (fio vermelho),
   - o pino 3, V0, foi conectado ao pólo do meio do potenciômetro de 10K (fio azul),
   - o pino 4, RS, foi conectado ao pino 43 do Arduino (fio branco),
   - o pino 5, R/W, foi conectado ao GND do Arduino (fio preto),
   - o pino 6, E, foi conectado ao pino 44 do Arduino (fio amarelo),
   - o pino 11, DB4, foi conectado ao pino 45 do Arduino (fio branco),
   - o pino 12, DB5, foi conectado ao pino 42 do Arduino (fio amarelo),
   - o pino 13, DB6, foi conectado ao pino 46 do Arduino (fio branco),
   - o pino 14, DB7, foi conectado ao pino 41 do Arduino (fio amarelo),
   - o pino 16, K, foi conectado ao GND do Arduino (fio preto);
b) o potenciômetro de 10K, utilizado para controlar o brilho do *display* de LCD também foi colocado primeiro na *protoboard* de 480 pontos, sendo que:
   - o pólo do meio foi conectado ao pino 3 do *display* LCD,
   - um dos pólos das extremidades ao 5V do Arduino,
   - o outro pólo no GND do Arduino;
c) o sensor de cores RGB TCS230, sendo que:
   - o pino S0 foi conectado ao pino 47 (fio amarelo),
   - o pino S1 foi conectado ao pino 48 (fio laranja),
   - o pino S2 foi conectado ao pino 50 (fio roxo),
   - o pino S3 foi conectado ao pino 51 (fio marrom),
   - o pino OE foi conectado ao GND (fio preto),
   - o pino OUT foi conectado ao pino 49 (fio azul),
   - o pino VCC foi conectado ao pino 5V (fio vermelho),

- o pino GND foi conectado ao GND do Arduino (fio preto).

Vale ressaltar que a *protoboard*, onde foi conectado o *display* LCD e o potenciômetro, foi fixada na parte de cima da base do robô. Já o sensor de cores RGB foi posicionado na parte de baixo da estrutura, apontado para o chão, de modo que consiga identificar as cores no cenário.

APÊNDICE B – Programa escrito na linguagem Robotoy e sua correspondência na linguagem do Arduino

Este apêndice apresenta um programa escrito na linguagem Robotoy e a sua correspondência na linguagem do Arduino. No Quadro 24 pode ser observado o programa, onde destaca-se o código adicionado pelo integrador da plataforma Arduino. O integrador adiciona ao programa traduzido todas as variáveis globais que representam os componentes do robô, mesmo as não configuradas, bem como, as configurações necessárias para cada componente na função `setup()`.

Quadro 24 - Programa escrito na linguagem Robotoy e sua correspondência na linguagem do Arduino

Programa em Robotoy

```
1  número passos <- 0
2  enquanto não tem obstáculo
3    andar para frente
4    passos <- passos + 1
5  fim do enquanto
6  texto mensagem <- "Nro de passos: " . passos
7  escrever mensagem
```

Programa na linguagem do Arduino

```
1  #include <Ultrasonic.h>
2  #include <Servo.h>
3  #include <AFMotor.h>
4  #include <LiquidCrystal.h>
5  const int _s0 = 47;
6  const int _s1 = 48;
7  const int _s2 = 50;
8  const int _s3 = 51;
9  const int _out = 49;
10 const String vermelho = "vermelho";
11 const String verde = "verde";
12 const String azul = "azul";
13 const String amarelo = "amarelo";
14 const String branco = "branco";
15 const String preto = "preto";
16 #define _HC_SR04_TRIGGER A2
17 #define _HC_SR04_ECHO A3
18 #define _BUZZER A0
19 Servo _motorMultiuso;
20 int _anguloSensor = 90;
21 AF_DCMotor _motorEsquerdo(4);
22 AF_DCMotor _motorDireito(1);
23 LiquidCrystal _lcd(43, 44, 45, 42, 46, 41);
24 boolean _ligado = true;
25 double passos;
26 String mensagem;
27 void setup() {
28   pinMode(_s0, OUTPUT);
29   pinMode(_s1, OUTPUT);
30   pinMode(_s2, OUTPUT);
31   pinMode(_s3, OUTPUT);
32   pinMode(_out, INPUT);
33   digitalWrite(_s0, HIGH);
34   digitalWrite(_s1, HIGH);
35   pinMode(_HC_SR04_TRIGGER, OUTPUT);
36   pinMode(_HC_SR04_ECHO, INPUT);
37   pinMode(_BUZZER, OUTPUT);
38   _motorMultiuso.attach(10);
39   _motorMultiuso.write(_anguloSensor);
```

```
40    _setarVelocidadeDeslocamento();
41    _lcd.begin(16, 2);
42    delay(1000);
43  }
44  void loop() {
45    while (_ligado){
46      passos = 0;
47      while (!_temObstaculo()) {
48         _andarFrente();
49         passos = passos + 1;
50      }
51      mensagem = "Nro de passos: " + (String) passos;
52      _escrever(mensagem);
53      _ligado = false;
54    }
55  }
```