UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

SÉRGIO TORRES GUIDA

ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS UTILIZANDO ROBÔS PROGRAMÁVEIS

BAURU

2021

SÉRGIO TORRES GUIDA

ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS UTILIZANDO ROBÔS PROGRAMÁVEIS

Trabalho de Conclusão de Curso do curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, campus Bauru.

Orientador: Prof. Dr. Renê Pegoraro

BAURU

2021

SÉRGIO TORRES GUIDA

ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS UTILIZANDO ROBÔS PROGRAMÁVEIS

Trabalho de Conclusão de Curso do curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, campus Bauru.

Banca Examinadora

Prof. Dr. Renê Pegoraro
Orientador
Departamento de computação
Faculdade de Ciências
UNESP – Bauru

Profa. Dra. Simone das Graças Domingues Prado
Departamento de computação
Faculdade de Ciências
UNESP - Bauru

Prof. Dr. Wilson Massashiro Yonezawa
Departamento de computação
Faculdade de Ciências
UNESP - Bauru

BAURU
2021

Guida, Sérgio
G946e Ensino de programação para crianças utilizando robôs programáveis / Sérgio Guida. -- Bauru, 2021
67 p. : il.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru
Orientador: Renê Pegoraro

1. Ensino. 2. Programação. 3. Robôs. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Resumo

Com os avanços da tecnologia, a área da educação vem adotando cada vez mais equipamentos tecnológicos como lousas digitais e dispositivos móveis para auxiliar no processo de ensino. O uso de robôs em sala de aula pode prover ao professor uma variedade de possibilidades de assuntos para ensinar, como circuitos elétricos, física e programação. Esse projeto visa construir um robô programável que pode ser utilizado para ensinar conceitos básicos de programação. Desenvolveu-se também o aplicativo que controla o robô disponível para os sistemas operacionais *Android* e *iOS*. O aplicativo permite ao usuário utilizar comandos diferentes para que o robô conclua tarefas específicas, como percorrer um percurso específico ou fazer formas geométricas. A comunicação entre o robô e o aplicativo é feita utilizando *bluetooth*. Foram utilizadas as seguintes tecnologias para a criação do robô e do aplicativo: *Arduino, Dart* e *Flutter.*

Palavras-chave: Educação. Robôs. Conceitos Básicos de Programação. Bluetooth. Arduino. Dart. Flutter.

Abstract

With the advancement of technology, the field of education is increasingly using technological equipment such as digital whiteboards and mobile devices to help on the learning process. The use of robots in class can provide teachers a variety of possibilities of subjects to teach such as electronic circuits, physics and programming. This project aims to construct a robot that can be used to teach basic concepts of programming. The app which controls the robot was developed with availability to Android and iOS devices. The app lets the user utilize different commands to make the robot accomplish specific tasks such as walking through a specific path or drawing geometric shapes. The communication between the robot and the app is made using bluetooth. The following technologies were used to create the robot and the app: Arduino, Dart and Flutter.

Keywords: Education. Robots. Basic Concepts of Programming. Bluetooth. Arduino. Dart. Flutter.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

A tecnologia está cada vez mais sendo reconhecida como uma importante ferramenta de ensino para ajudar crianças no desenvolvimento de suas habilidades cognitivas, sociais e de aprendizado (Tahir et al., 2015).

O uso dela em sala de aula é cada vez mais desejado por crianças e jovens, pois eles cresceram em um ambiente repleto de *smartphones*, assistentes virtuais, e diversos outros tipos de tecnologias e acabam não se identificando com o método tradicional de ensino utilizado. Segundo Junior (2017), o uso de celulares e suas aplicações, permite aos alunos o desenvolvimento de habilidades cognitivas e uma aprendizagem mais lúdica, significativa e atraente. Junior continua dizendo que esses recurso ajudam os professores e permitem aos alunos a prática e o estudo fora da escola.

Segundo Ruiz-del-Solar e Avilés (2004), a robótica é uma atividade altamente motivadora para crianças. Ela as permite abordar a tecnologia de forma divertida e intuitiva, enquanto descobrem os princípios básicos da ciência.

O uso de robôs para ensino data desde os arredores de 1940 após a Segunda Guerra Mundial. O Lilliput, apresentado na Figura 1, era um robô de dar corda, japonês, bem simples, mas sua produção foi interrompida devido ao foco das indústrias japonesas de reconstruir o país.

Figura 1 – Robô Lilliput

Fonte: *Junk42*[1]

1 – Disponível em: < https://www.junk42.com/products/tin-toy-robot-lilliput>. Acesso em: 6 jun. 2021.

Uma grande inspiração para esse projeto foi o LOGO de Papert (1980). Seu projeto consistia em um robô chamado de “tartaruga” que recebe comandos de crianças e cria um desenho seguindo esses comandos. Segundo Papert (1980), as crianças conseguem se identificar com a tartaruga, se colocando no lugar dela e experimentando diferentes combinações de comandos para tentar chegar em algum resultado desejado. Outra influência é o jogo LightBot[2], que tem como ideia base a utilização de lógica computacional para programar um robô dentro do jogo para cumprir os objetivos de cada fase. Baseado nas ideias de Papert e nas do LightBot, é proposto um ambiente formado por um robô, um aplicativo celular e uma área quadrada de 2m x 2m dividida em 25 células onde o robô ou um aluno possa se locomover seguindo comandos para realizar uma tarefa. O robô será controlado por uma sequência de comandos pré-escolhidos pelos alunos através do celular com o intuito de resolver atividades predefinidas.

1.1 Problema

Muitas instituições de ensino não ensinam programação para as crianças, principalmente por falta de recursos humanos e tecnológicos, o que é um problema, devido à importância da área da computação. Uma pesquisa feita por Lima, Vieira e Brandão (2019) mostra que entre dezoito municípios com mais de um milhão de habitantes, apenas dois possuíam itens relacionados ao ensino de programação em suas orientações curriculares. O estudo mostra também que muitas escolas só possuem algum tipo de ensino de computação através de trabalhos realizados por pesquisadores e alunos de graduação, porém esses projetos são limitados em número de vagas, não conseguindo abranger a todos os alunos e são limitados pela infraestrutura local, as vezes, até inviabilizando o projeto.

1.2 Justificativa

Dado o contínuo crescimento da área da computação e suas aplicações atualmente e a comum falta de recursos das escolas, esse projeto visa projetar um robô de custo baixo que possa ser recriado facilmente e um aplicativo simples para a execução de atividades planejadas pelo professor. A programação do robô seria uma atividade mais engajante como mostra um estudo feito por Voštinár e Klimová (2019)

2 – Disponível em: < https://lightbot.com/>. Acesso em: 18 jun. 2021.

onde crianças ficaram bastante interessadas e gostaram de aprender utilizando robôs programáveis e que o trabalho dos professores também foi facilitado. Esse estudo mostra que esse tipo de ensino pode funcionar bem e que pode ser um possível futuro para o ensino da programação.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um robô programável e um aplicativo conectado a esse robô que possa ser utilizado para ensinar a crianças conceitos básicos de programação como o encadeamento de comandos para resolver um problema, permitindo que elas deem comandos para o robô realizar tarefas simples.

1.3.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

a) Construir um robô móvel com funções básicas que possa ser controlado via comunicação sem fio através de um aplicativo celular.

b) Criar um aplicativo que permita a programação o robô.

c) Criar algumas atividades a serem realizadas pelo robô e pelos alunos.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados e explicados conceitos abordados ao longo do trabalho.

2.1 Robótica na Educação

A robótica é amplamente utilizada em diversos setores. É possível ver robôs em chão de fábrica produzindo produtos com extrema precisão e rapidez, em casa, onde robôs aspiradores tornam o processo de limpeza quase que automático, ou até mesmo em serviços de entrega de mercadorias com os drones.

O uso da robótica na educação data desde a década de 1960 com Seymour Papert surgindo com o termo robótica educacional e defendendo o uso de computadores em escolas por serem atrativos para crianças. Em seu livro *Mindstorms: Computers, Children and Powerful Ideas* (1980), Papert diz que no sistema de ensino, é geralmente assumido que crianças não conseguem aprender geometria até estarem a um bom tempo na escola e alguns alunos mesmo após anos de ensino não aprendem muito bem, mas ele diz que essa assunção possui provas muito fracas, pois, segundo ele, qualquer criança consegue aprender algo vivenciando aquilo. Para exemplificar isso ele utiliza o ensino de francês. Muitas crianças na escola não conseguem aprender e utilizar completamente a língua, porém se essas mesmas crianças vivessem na França, elas com certeza seriam fluentes na língua. Sua proposta então, é que as crianças vivam no mundo da matemática, ou, como ele chama '*Mathland*' utilizando robôs e computadores como o meio de comunicação entra a criança e o ensino de matemática no caso.

Para colocar sua proposta em prática, Papert desenvolveu um robô chamado '*The Turtle*' ou a tartaruga. Ele chamava a tartaruga de um '*object-to-think-with*' que em uma tradução direta para o português seria objeto para pensar com, ou seja, algo que te faça pensar e aprender.

A tartaruga era um robô programável que tinha um formato semelhante a uma tartaruga e que riscava o chão por onde passasse. Ela fazia uso da linguagem LOGO, que foi feita para projetos educacionais. A linguagem era bem simples e parecida com nossa linguagem natural, para facilitar a comunicação da pessoa com o robô. A Figura 2 apresenta um exemplo de código em LOGO para programar a tartaruga.

Figura 2 – Exemplo de código em LOGO

```
TO SQUARE
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
END
```

Fonte: *Papert (1980)*

O ensino através do LOGO para Papert tinha o aluno como o principal construtor de seus conhecimentos. O aluno deveria errar, refletir sobre seu erro e tentar novamente, assim desenvolvendo sua capacidade intelectual. O papel do professor seria permitir os erros e somente guiar o estudante.

Com o tempo os avanços tecnológicos permitiram a criação de robôs mais complexos e com usos diferentes. Atualmente diversos tipos de robôs programáveis utilizados para a educação estão disponíveis. Segundo O`Brien (2019), é possível dividi-los em quatro categorias: robôs programáveis fisicamente, robôs introdutórios à programação, robôs programáveis por computador e robôs de kit. A Figura 3 mostra um exemplo de cada tipo de robô respectivamente.

Figura 3 – Exemplos de robôs programáveis

Fontes: *Terrapin*[3]*, Learning Resources*[4]*, Sphero*[5] *e Lego*[6]*.*

3 - Disponível em: <https://www.terrapinlogo.com/products/robots/bee/bee-bot-family.html>. Acesso em: 18 jun. 2021.
4 - Disponível em: <https://www.learningresources.com/shop/collections/botley>. Acesso em: 18 jun. 2021.
5 - Disponível em: <https://sphero.com/products/sphero-bolt>. Acesso em: 18 jun. 2021.
6 - Disponível em: <https://www.lego.com/pt-br/product/lego-mindstorms-ev3-31313>. Acesso em: 18 jun. 2021.

2.2 Dispositivos Móveis na Educação

Outro tipo de tecnologia que avançou e continua avançando muito são os dispositivos móveis. A possibilidade de acessar qualquer tipo de informação rapidamente e em qualquer lugar, poder ter conversas a distância instantaneamente ou poder assistir a um filme enquanto não está em casa são apenas algumas das muitas utilidades destes aparelhos. Além disso, existe a possibilidade de uso deles para a educação.

> Crianças estão sempre animadas para utilizar dispositivos móveis, então aprender é muito engajante e providencia para as crianças um novo meio de relacionar suas experiências com o conhecimento abstrato. (Tahir e Arif, 2015)

Esses jovens cresceram rodeados dessas tecnologias e elas fazem parte de sua rotina, tanto que segundo Couse e Chen (2010), a questão deixou de ser: “até onde a tecnologia deveria ser utilizada em salas de aula?” para: “como a tecnologia deveria ser utilizada?”. A gama de aplicativos disponíveis é enorme. Desde agendas à aplicativos que ensinam a programar, as possibilidades de o que utilizar são muitas.

Devido à natureza interdisciplinar dessas tecnologias, juntar o uso da robótica com aplicativos móveis pode ser benéfico para o desenvolvimento dos alunos em diversas áreas do ensino, como matemática, física e a programação e podendo até mesmo utilizar os robôs para fixar conceitos de outras áreas como a geografia e a biologia.

> Ao trabalhar em um ambiente de robótica educativa, o protótipo construído pelos alunos passa a ser um artefato cultural que os alunos utilizam para explorar e expressar suas próprias ideias. (Gomes, Silva, Botelho e Souza, 2010)

2.3 Dispositivos Semelhantes

A seguir serão apresentados os tipos de dispositivos que influenciaram a criação desse trabalho e operam de maneira semelhante e alguns exemplos de cada tipo de robô.

2.3.1 LOGO

Primeiramente, o principal influenciador e um dos pioneiros da robótica educacional o ambiente LOGO consiste em uma tartaruga gráfica e um robô que responde aos comandos dados à tartaruga. Desenvolvido por Seymour Papert, utilizava um conceito de aprendizado construtivista, onde seu objetivo era auxiliar no processo de aprendizado dos alunos. Os alunos passavam os comandos desejados para a tartaruga gráfica que em seguida fazia com que o robô executasse esses comandos. Assim, era possível ver quase que instantaneamente o resultado do seu programa, permitindo a reflexão e o aprendizado. A Figura 4 apresenta uma imagem da tartaruga sendo programada por crianças.

Figura 4 – Foto do ambiente LOGO

Fonte: *Papert (1980)*

2.3.2 Bee-Bot

Esse robô, apresentado na Figura 5, é um exemplo de um robô programável fisicamente. Ele possui sete botões de programação: ir para frente, ir para trás, virar para direita, virar para esquerda, começar, pausar e o botão de começar para que o robô execute os comandos desejados. Ao completar cada comando uma luz pisca e o robô faz um barulho para a criança saber qual passo está sendo executado.

Um estudo feito por Yu e Roque (2018) mostra que robôs com interfaces físicas influenciam fortemente crianças a perceber e usar esses tipos de robôs.

Figura 5 – Imagem do Bee-Bot

Fonte: *Terrapin*

2.3.3 Botley

Botley, apresentado na Figura 6, entra na categoria de robôs introdutórios à programação. Esses robôs são fisicamente parecidos com os robôs fisicamente programáveis, porém a maneira que são programados é diferente. Segundo O`Brien todos esses robôs utilizam algum dispositivo de programação separado do corpo principal do robô. Alguns utilizam controles remotos, mas as aplicações de programação mais comuns rodam através de *tablets* e *smartphones*.

O`Brien continua dizendo que esses tipos de robôs definem a separação de *hardware* e *software* no funcionamento de um robô. Esses tipos de robôs já passam a possuir sensores, permitindo a eles movimentos mais complexos e com isso o aluno pode criar programas que vão além de simples estruturas sequenciais de código.

O robô Botley utiliza um controle remoto para sua programação e como diferencial possui um sensor de ultrassom para não colidir com objetos. Ele permite também a criação de *loops* para a criação de programas mais complexos.

Figura 6 – Imagem do Botley

Fonte: *Learning Resources*

2.3.4 Sphero Bolt

Esse robô se encaixa na categoria de robôs programáveis por computador. Eles são mais avançados do que os robôs introdutórios à programação em relação aos seus sensores e passam a parecer menos com brinquedos, tomando formas mais funcionais para se adequarem às funções para as quais foram projetados.

A programação deles, como o nome da categoria mostra, necessita de um computador para ser feita. Geralmente eles possuem linguagens de programação e ambientes próprios para cada robô, assim criando a necessidade de aprender novas sintaxes e *software* para aproveitar todas suas funcionalidades. Porém a maioria desses ambientes de programação não utilizam linguagens baseadas em texto, tornando a programação algo ainda visual e de fácil compreensão.

O Sphero Bolt, apresentado na Figura 7, possui diversos sensores que o tornam bastante complexo e aumenta sua gama de opções quando programado. Ele possui sensores de comunicação infravermelhos, que possibilitam a comunicação entre robôs, uma matriz de 8x8 LEDs que podem ser programados para mostrar diversas formas diferentes, sensores de claridade e uma bússola interna que pode ser utilizada para mudar as direções do robô via código.

Figura 7 – Imagem do Sphero Bolt

Fonte: *Sphero*

Com ele vem um ambiente de programação chamado Sphero Edu, que permite a programação do robô de três maneiras de dificuldades crescentes. A primeira maneira, apresentada na Figura 8, permite que o robô seja programado ao desenhar na tela o caminho que o usuário deseja que o robô percorra.

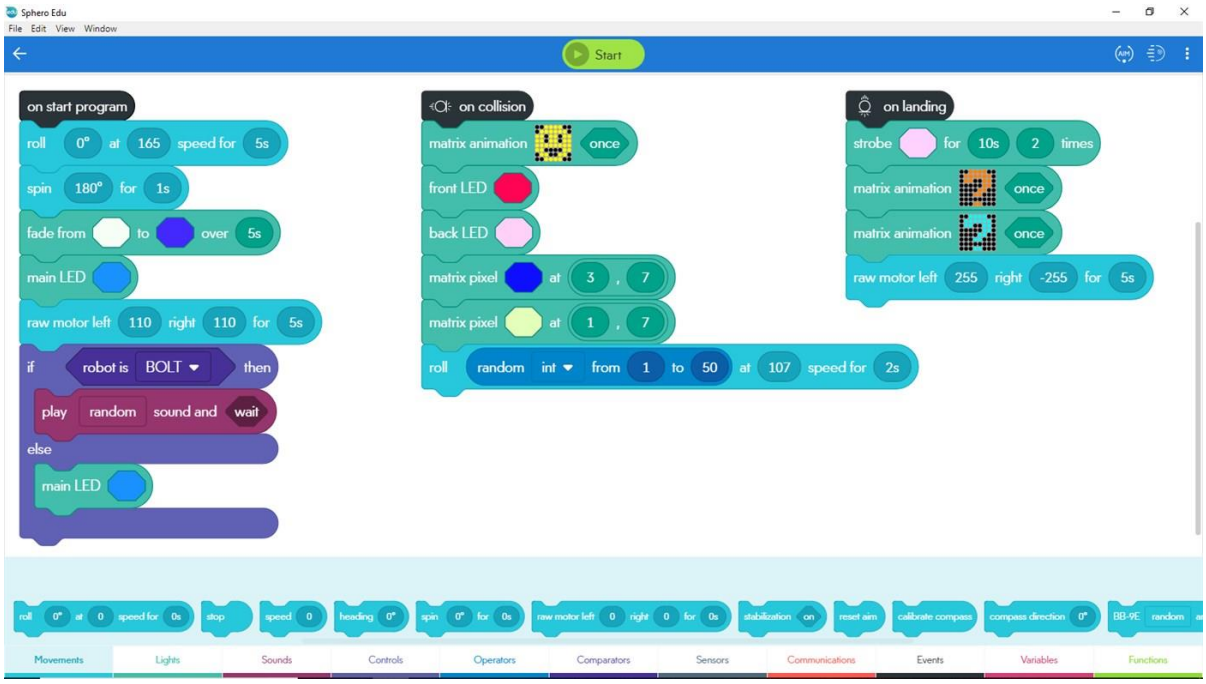
Figura 8 – Ambiente de programação Sphero Edu utilizando desenhos

Fonte: *Microsoft App Store*[7]

Sua segunda maneira de programação, apresentada na Figura 9, utiliza blocos. Cada bloco realiza uma certa tarefa e podem ser encadeados para formar um código. Essa forma já aumenta o grau de complexidade dos programas feitos pelo robô, pois permitem a criação de loops e o uso de condicionais, além da ativação dos sensores e da matriz de LEDs.

7 – Disponível em: <https://www.microsoft.com/en-us/p/sphero-edu/9n2796r62xlz?rtc=1&activetab=pivot:overviewtab>. Acesso em: 10 jun. 2021.

Figura 9 – Ambiente de programação Sphero Edu utilizando blocos



Fonte: *Microsoft App Store*

A última e mais complexa forma de programação, apresentada na Figura 10, utiliza JavaScript e permite integração com outros aparelhos, e uma enorme gama de novas opções.

Figura 10 – Ambiente de programação Sphero Edu utilizando JavaScript

Fonte: *Microsoft App Store*

2.3.5 LEGO Mindstorms EV3

Como O`Brien apresenta, o tipo mais complexo de robôs são os de kit. Eles vêm desmontados, necessitando tanto a programação quanto a montagem para funcionarem. Isso proporciona um desafio interessante de criatividade e engenharia ao usuário. Eles possuem diversos tipos de sensores e motores para possibilitar muitos resultados diferentes.

Sua programação é similar aos robôs programados por computadores, porém o ambiente tende a não ser tão amigável, normalmente utilizando programação por texto.

A empresa LEGO é famosa por oferecer kits de blocos para montar desde bonecos a réplicas de edifícios. O LEGO Mindstorms EV3, apresentado na Figura 11, fornece a mesma experiência de montar um robô, mas inclui motores e sensores para possibilitar a criação de robôs customizados ao gosto do usuário.

Figura 11 – Kit LEGO Mindstorms EV3

Fonte: *Robot Education*[8]

8 – Disponível em: <https://roboteducation.com.br/loja/lego-education-mindstorms-ev3-45544/>. Acesso em: 10 jun. 2021.

2.3.6 LightBot

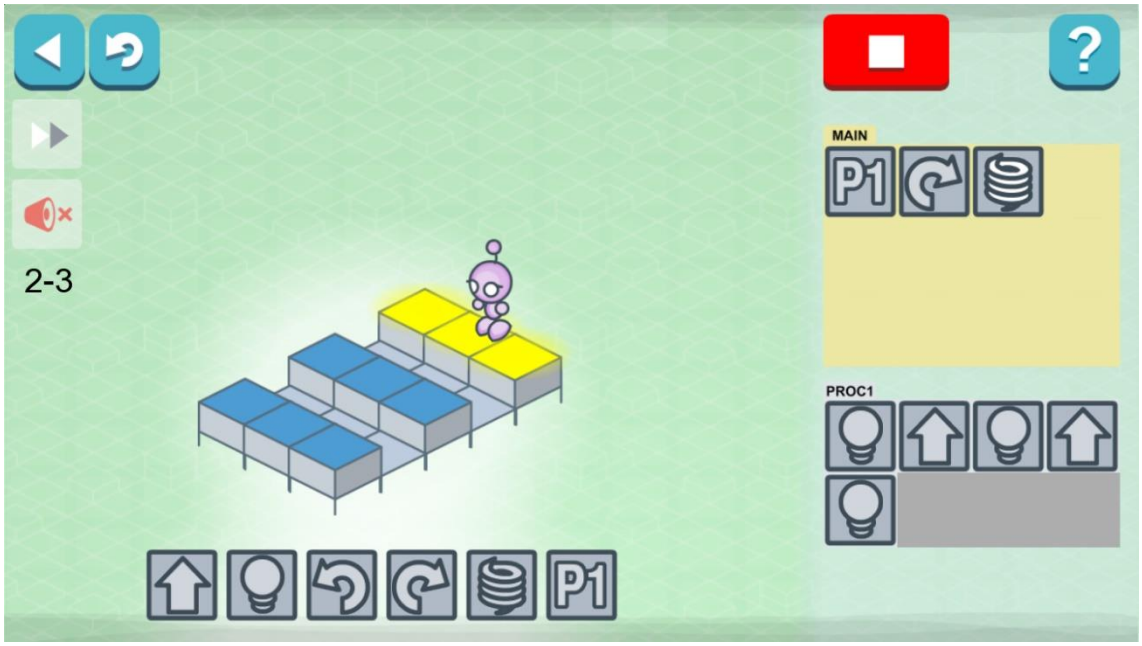
Diferentemente dos outros itens desse capítulo, o LightBot é um jogo para navegador de internet e dispositivos móveis, mas também serviu de grande inspiração para esse trabalho.

Publicado em 2008 por Danny Yaroslavski, o LightBot tem a premissa de ensinar o pensamento lógico através de um jogo onde o usuário programa um pequeno robô através de blocos com símbolos.

O jogo fez enorme sucesso tanto entre usuários comuns como entre alunos e professores em salas de aula. Uma reportagem do site americano TechCrunch mostra que o jogo já foi jogado mais de sete milhões de vezes entre as plataformas que está disponível. Nessa mesma reportagem, Danny, o criador do jogo, diz que professores dos Estados Unidos, da Rússia e de muitos outros países utilizam o jogo em sala de aula para introduzir conceitos de computação.

Alguns conceitos abordados no jogo são: criação de funções, utilização de loops e a resolução de problemas utilizando comandos básicos como andar para a frente, virar para os lados e saltar. A Figura 12 apresenta uma fase do jogo e os comandos sendo utilizados para mover o personagem.

Figura 12 – Captura da interface do jogo



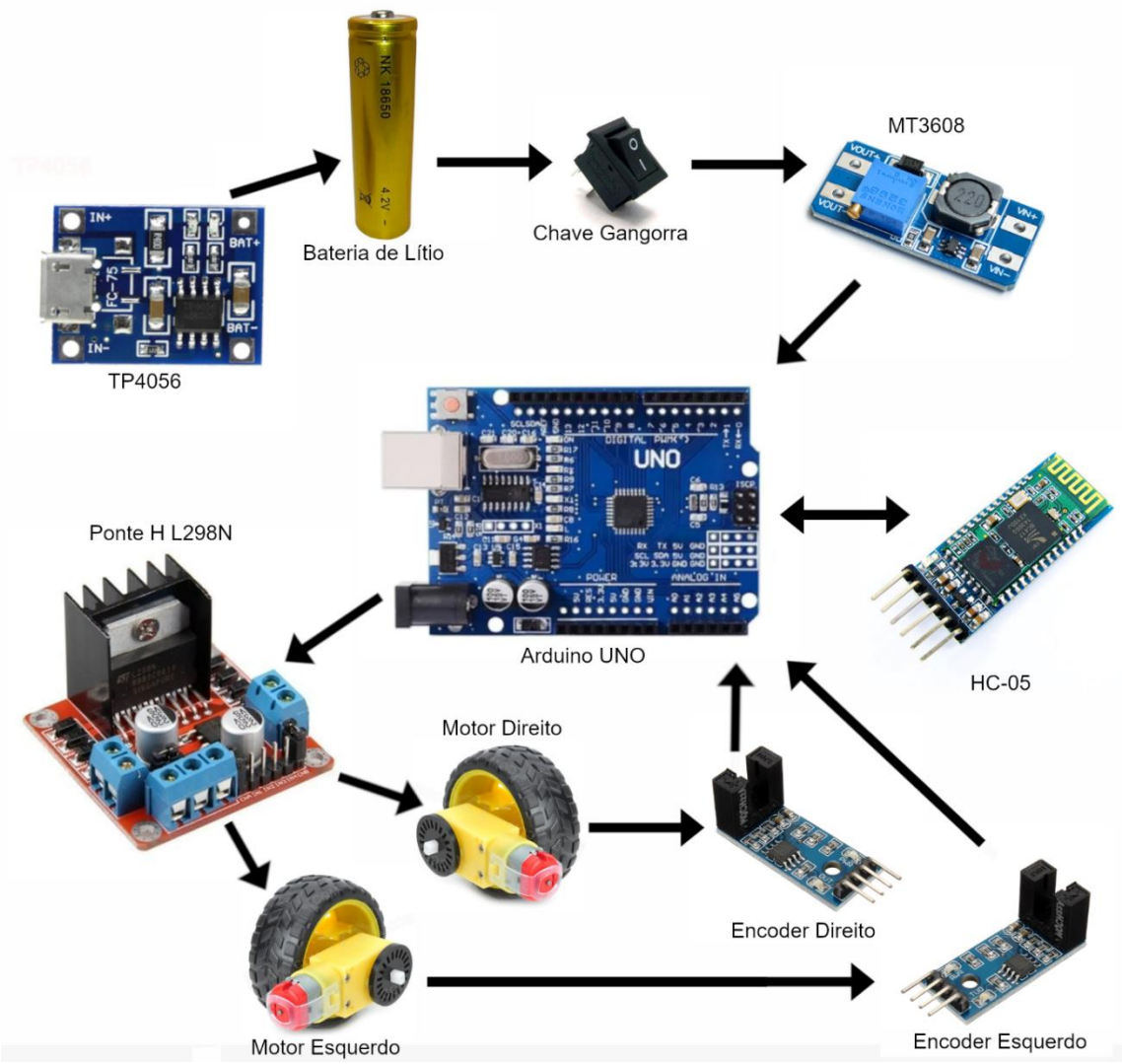
Fonte: *LightBot*

3 TECNOLOGIAS UTILIZADAS

Nesta seção serão descritas as tecnologias utilizadas para a criação do robô e do aplicativo que compõem esse trabalho.

O diagrama apresentado na Figura 13 mostra todos os componentes do robô e como interagem entre si.

Figura 13 – Diagrama de interação dos componentes do robô



Fonte: *Elaborado pelo autor*

3.1 Partes do Robô

Aqui serão descritos todos os componentes utilizados na construção do robô.

3.1.1 Kit Chassi

A base do robô onde todos os outros componentes são colocados é um Kit Chassi 2WD como visto na Figura 14. Esse kit é composto de uma base em acrílico, dois motores DC que funcionam separadamente com tensões de 3 a 6 volts, duas rodas de borracha, uma roda boba, dois discos de *encoder*, um suporte para quatro pilhas alcalinas tipo AA, um *switch* de liga e desliga e um jogo de parafusos.

Figura 14 – Kit Chassi montado

Fonte: *Filipeflop*[9]

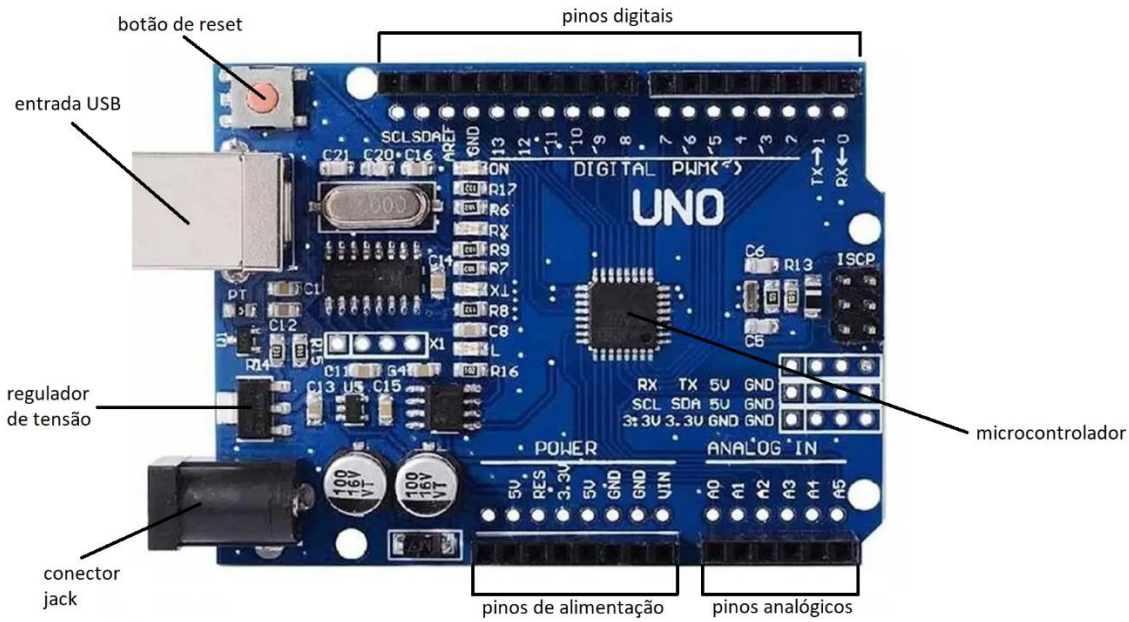
3.1.2 Arduino

A parte mais importante do robô, o Arduino é uma plataforma que tem como objetivo a facilitação da prototipagem ou implementação de sistemas interativos, no caso deste projeto, o robô e sua comunicação com um aplicativo móvel. O Arduino é uma plataforma completa que só requer as conexões de entrada e saída, a sua programação e alimentação para funcionar.

Neste desenvolvimento foi utilizado o Arduino Uno Ref 3. As partes principais que o compõem são: os conectores fêmea de entrada e saída, uma entrada USB, um conector *jack* de entrada, dois reguladores de tensão, um botão de *reset* e um microcontrolador ATmega328.

9 – Disponível em: <https://www.filipeflop.com/produto/kit-chassi-2wd-robo-para-arduino/>. Acesso em: 13 jun. 2021.

Figura 15 – Arduino e seus componentes



Fonte: *Elaborado pelo autor*

A alimentação do Arduino pode ser feita de três maneiras. O *jack* de entrada que recebe uma tensão de entrada entre 6V e 20V, mas a recomendação é que a fonte externa possua entre 7V e 12V para não alimentar o dispositivo nos limites da operação, no limite máximo pode resultar em sobreaquecimento do regulador interno e no limite inferior o regulador pode não conseguir fornecer os 5V indispensáveis ao funcionamento da placa. A utilização do *jack* utiliza o regulador de tensão interno presente no Arduino (Figura 15), já que a tensão de funcionamento da placa é de 5V. Outra forma de alimentação é utilizando o conector USB. A última forma de alimentação é externa utilizando o pino Vin.

A comunicação do Arduino com outros componentes é feita através de vinte pinos de entrada e saída, sendo que 14 utilizam sinais digitais e 6 utilizam sinais analógicos. Todos os pinos operam em 5V. Importante mencionar que os pinos 0 e 1 fazem a comunicação serial e são ligados ao microcontrolador que faz a comunicação da USB com o computador. Estes pinos devem estar desconectados no momento da programação do Arduino. O Arduino possui também sete conectores de alimentação, entre eles: dois pinos GND, um pino que fornece 5V, um pino que fornece 3,3V, o pino Vin para alimentação externa do Arduino, um pino de *RESET* que possibilita o reset externo da placa.

O componente mais importante da placa é o ATmega328. Esse componente é um microcontrolador que é responsável por realizar os comandos de seu programa, enviando e recebendo sinais dos pinos de entrada e saída.

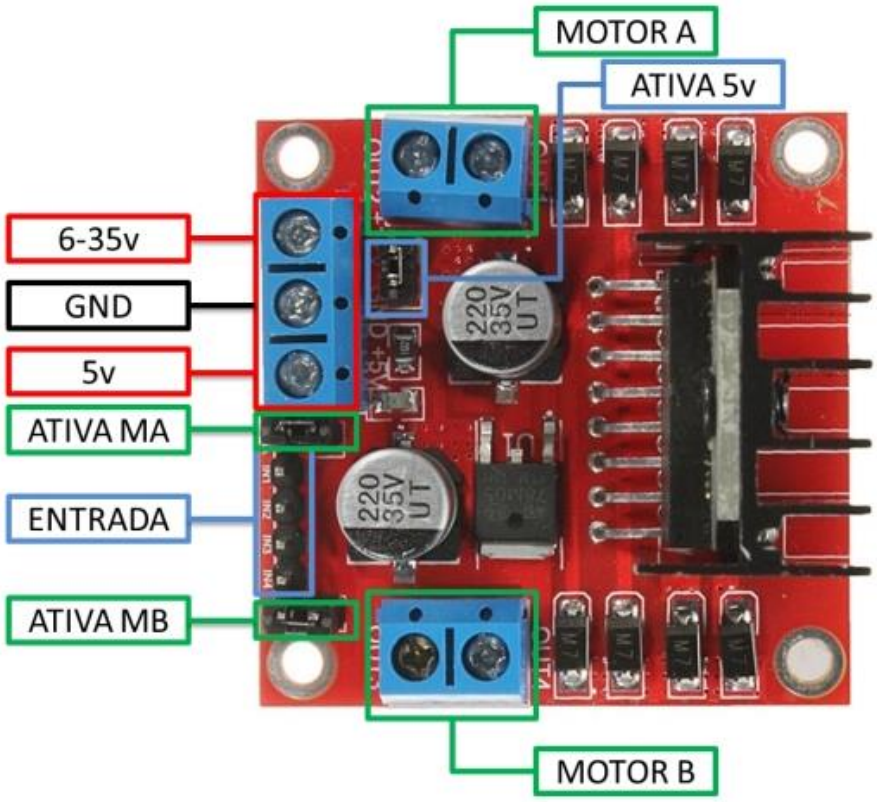
Sua programação é feita utilizando a linguagem C++ com algumas pequenas modificações. Seu código não necessita de uma função *main( )* para funcionar, mas necessita das funções *setup( )* e *loop( )*. Na função *setup( )* ocorre a iniciação dos pinos do Arduino onde o programador pode escolher se um pino vai ser de entrada ou saída, os valores iniciais de cada pino e a utilização de outras funções como a iniciação de variáveis e a abertura da comunicação serial. Já a função *loop( )*, que a tradução significa “laço”, é um laço eterno que geralmente é utilizada para realizar as funções principais do projeto.

3.1.3 Ponte H L298N

A Ponte H é o componente responsável pelo acionamento dos motores DC. Como as correntes fornecidas nas saídas do Arduino não são suficientes para acionar os motores, a ponte H é necessária para fornecer tais correntes protegendo o Arduino e permitindo que os motores operem com maiores tensões e correntes além das fornecidas pelo Arduino. Ela possui quatro saídas que enviam sinais para os motores. Esse sinal vem de quatro pinos de entrada que fazem a comunicação com o Arduino.

Ela possui também um modo de controle de velocidade dos motores. Para ativar esse modo, basta remover o *jumper* que fica sobre os pinos *ENABLE*, possibilitando assim que o Arduino controle a velocidade dos motores. Com o *jumper* conectado, a velocidade é sempre a máxima. Há outro *jumper* presente na placa que altera o estado de um pino entre entrada e saída de 5V. Ao retirar esse *jumper* é possível usar esse pino como saída para alimentar algum outro componente como o próprio Arduino. A Figura 16 apresenta a Ponte H, seus pinos e *jumpers*.

Figura 16 – Ponte H L298N e seus componentes



Fonte: *Filipeflop*[10]

A alimentação da placa se dá através de uma entrada GND e uma entrada que opera entre 4V e 35V. O uso de voltagens entre 4V e 5,5V necessita do uso do pino de 5V como entrada.

Nesse projeto, a ponte H está sendo alimentada pela bateria de lítio NK 18950 com sua tensão ajustada pelo MT3608, totalizando 6V de tensão.

3.1.4 HC-05

Este componente é responsável pela comunicação *Bluetooth* entre o robô e o aplicativo móvel. Ele possui 6 pinos, como mostrado na Figura 17, sendo eles: ENABLE, +5V, GND, TX, RX e STATE.

Sua alimentação é simples, necessitando apenas uma entrada de 5V e um GND.

O pino ENABLE permite que o HC-05 seja configurado como mestre ou escravo ou habilitar o modo AT. Esse modo permite a configuração e conversa com o componente por meio de comandos em um terminal serial.

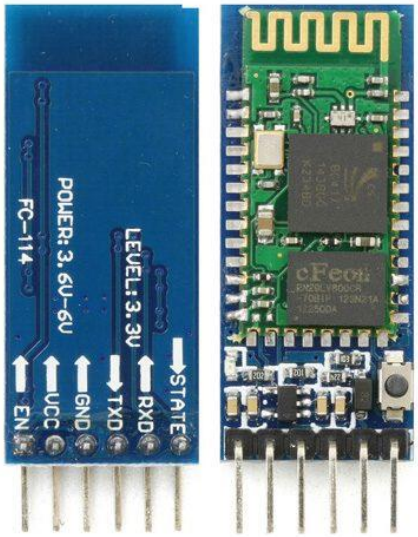
O pino STATE envia um sinal quando o HC-05 está conectado com algum dispositivo e um sinal baixo quando desconectado.

10 – Disponível em: <https://www.filipeflop.com/blog/motor-dc-arduino-ponte-h-l298n/>. Acesso em: 13 jun. 2021.

Os pinos TX e RX fazem a comunicação com o Arduino. Para funcionar o TX do módulo deve ser ligado ao RX do Arduino e o RX do módulo ao TX do Arduino. É importante lembrar que o RX e o TX do HC-05 funcionam com sinais de 3,3V e o RX e TX do Arduino operam com 5V, para evitar danos ao HC-05 é necessário que a entrada RX receba valores não maiores que 3,3V, para ajustar estes valores, usa-se dois resistores em série formando um divisor de tensão

Ele possui também um LED que sinaliza os estados do componente. Quando ele está pronto para parear[11], o LED pisca continuamente e quando conectado com um dispositivo, ele pisca duas vezes a cada 5 segundos.

Figura 17 – HC-05 e seus pinos



Fonte: *Arduino Eletrônica*[12]

3.1.5 Bateria Li-Ion

A Figura 18 uma bateria de lítio que fornece 4,2V com capacidade de até 9800mAh. Esse tipo de bateria é recarregável. Neste projeto ela está sendo utilizada como fonte de alimentação do Arduino e da Ponte H.

11 – Parear é o ato de conectar um dispositivo ao outro via *bluetooth*.
12 – Disponível em: <https://www.arduinoeeletronica.com.br/produto/modulo-bt-bluetooth-master-slave-hc-05/>. Acesso em: 13 jun. 2021.

Figura 18 – Bateria NK 18650

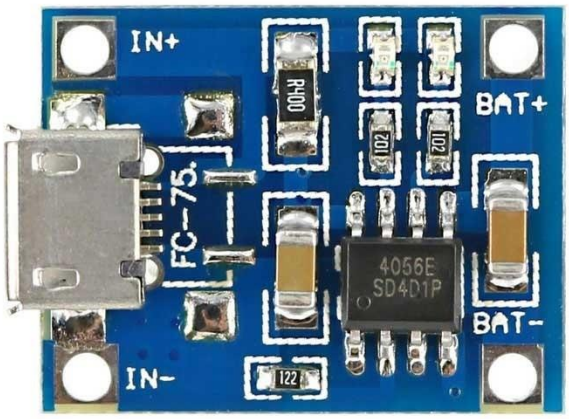
Fonte: *Eletrônica Castro*[13]

3.1.6 Carregador de Bateria TP4056

Esse componente, apresentado na Figura 19, é responsável por recarregar a bateria de lítio. Ele possui uma entrada micro USB e 4 pinos, sendo eles: BAT+, BAT-, IN+ e IN-.

Os pinos BAT+ e BAT- são ligados aos polos positivo e negativo da bateria, respectivamente. Já os pinos IN+ e IN- servem como entrada da fonte de energia para carregar a bateria caso não seja utilizada a entrada micro USB.

Figura 19 – TP4056



Fonte: *Smart Kits*[14]

13 – Disponível em: <https://www.eletronicacastro.com.br/pilhas/15091-bateria-recarregavel-42v-88000mah.html>. Acesso em: 14 jun. 2021.
14 – Disponível em: <https://www.smartkits.com.br/modulo-tp4056-carregador-de-baterias-de-litio>. Acesso em: 15 jun. 2021.

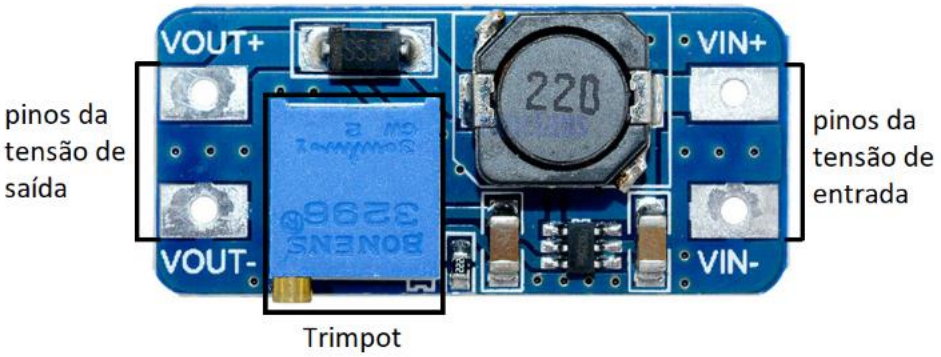
3.1.7 Regulador de Tensão Ajustável MT3608

Este componente permite a regulação da tensão que entra nele para uma de valor diferente. Neste projeto ele está recebendo os 4,2V da bateria de lítio e os transformando em 6V para a alimentação da Ponte H e do Arduino.

Ele funciona com tensões de entrada de 2V a 24V e oferece em suas saídas tensões de 2,5V a 28V, necessitando no mínimo uma diferença de 0,5V entre entrada e saída.

Ele possui 4 pinos, como mostra a Figura 20, dois para a tensão de entrada e dois para a tensão de saída e um *trimpot*. O *trimpot* funciona de maneira similar a um potenciômetro. Ele varia sua resistência conforme o pequeno parafuso é girado, assim variando a tensão de saída.

Figura 20 – MT3608 e seus pinos



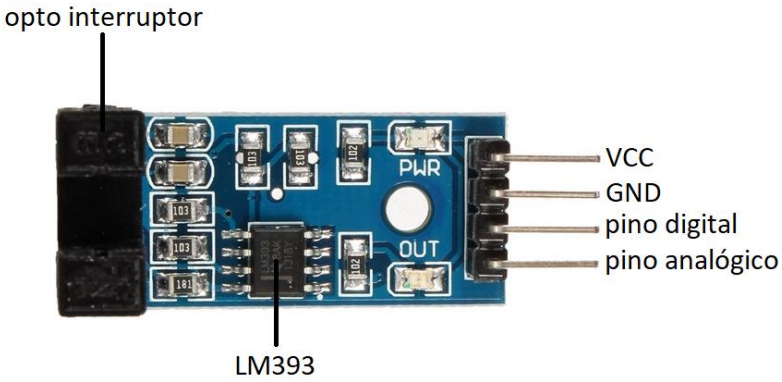
Fonte: *Elaborado pelo autor*

3.1.8 Sensor de Velocidade Encoder LM393

Este componente serve para medir a rotação dos motores. Para isso ele utiliza um opto interruptor, que de um lado possui um LED infravermelho e de outro um foto transistor. Quando o feixe de luz é interrompido pela rotação dos discos de *encoder* o pino de saída digital envia o sinal 1, caso contrário o sinal permanece 0.

Ele possui 4 pinos, como mostra a Figura 21, sendo dois deles de alimentação (3V à 5V e GND), uma saída digital e uma saída analógica.

Figura 21 – Sensor de Velocidade Encoder LM393 e seus pinos



Fonte: *Elaborado pelo autor*

3.2 Arduino IDE

O Arduino IDE, apresentado na Figura 22, é um ambiente de desenvolvimento integrado, ou do inglês, *Integrated Development Environment*. Ele é um *software open-source*, ou seja, seu código fonte é disponível para acesso por todos.

Esse ambiente é executado em um computador pessoal, ele possibilita a escrita de programas para uma placa do Arduino e a programação dela de maneira simples. Ele possui também um terminal que recebe os dados seriais enviados do Arduino ao computador que pode ser utilizado para imprimir dados e encontrar possíveis erros no código. Este IDE possui vários outros recursos como fazer o *upload* de seu código diretamente para o fórum do Arduino, ou mudanças mais visuais como aumentar o tamanho da fonte ou mudar a indentação do código.

Figura 22 – Imagem do Arduino IDE

Fonte: *Elaborado pelo autor*

3.3 Dart

Dart é uma linguagem de programação criada pela *Google* em 2011. Ela é uma linguagem orientada a objeto e projetada para desenvolver aplicativos rápidos para qualquer plataforma. Sua sintaxe tem um estilo baseado em C, logo, sua sintaxe também é muito similar a linguagens populares como Java e C#. Um trecho de código em *Dart* pode ser visto na Figura 23.

O código *Dart* pode ser compilado de duas maneiras diferentes: utilizando a *DartVM,* uma máquina virtual do *Dart,* ou transformando o código para *JavaScript* através da ferramenta dart2js, que otimiza o código *JavaScript*, podendo as vezes, rodar mais rápido que um código escrito inicialmente em *JavaScript* (DART, 2019).

A máquina virtual pode ser executada de duas maneiras: JIT (*just in time*) ou AOT (*ahead of time*). A compilação JIT ocorre no momento da execução de um trecho de código, onde ele é convertido para código de máquina à medida que ele é executado. Já na compilação AOT o código é convertido em código de máquina antes.

Figura 23 – Exemplo de trecho de código Dart

```dart
import 'dart:math' show Random;

void main() async {
  print('Compute π using the Monte Carlo method.');
  await for (final estimate in computePi().take(100)) {
    print('π ≅ $estimate');
  }
}

/// Generates a stream of increasingly accurate estimates of π.
Stream<double> computePi({int batch = 100000}) async* {
  var total = 0; // Inferred to be of type int
  var count = 0;
  while (true) {
    final points = generateRandom().take(batch);
    final inside = points.where((p) => p.isInsideUnitCircle);

    total += batch;
    count += inside.length;
    final ratio = count / total;

```

Fonte: *Dart*[15]

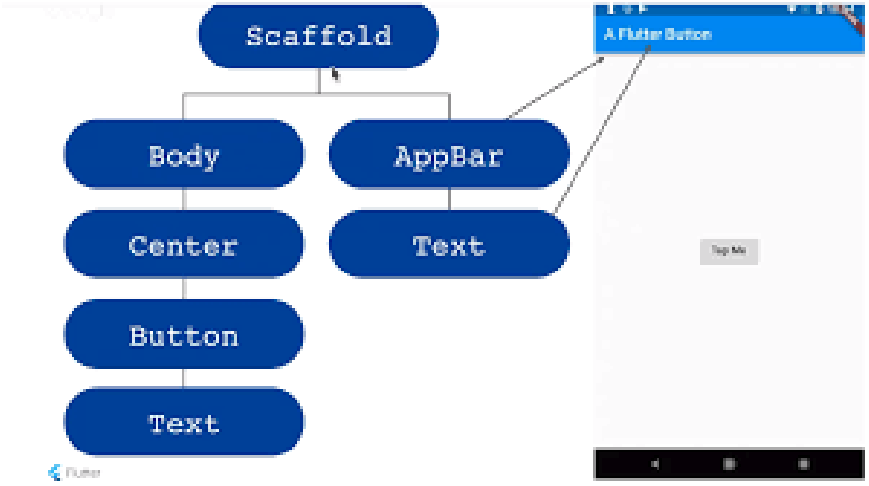
3.4 Flutter

Flutter é um *framework* da *Google* que usa *Dart* como sua linguagem base e que serve para o desenvolvimento de aplicativos para *Android* e *iOS*. Ele é multiplataforma por poder ser utilizado em qualquer sistema operacional.

O *framework* utiliza a árvores de *widgets* (Figura 24) como principal forma de estruturar o código. Um *widget* é como se fosse uma peça. Cada um já possui suas características básicas e ao adicioná-los um após o outro é formada uma árvore de *widgets* que compõem o programa. Isso facilita a visualização do código e suas conexões.

15 - Disponível em: <https://dart.dev/overview>. Acesso em: 18 jun. 2021.

Figura 24 – Árvore de *widgets* e seu funcionamento em um aplicativo



Fonte: *Antonio Leiva*[16]

O grande diferencial do *Flutter* é que ao compilar um código, ele é transformado para a linguagem base do dispositivo, tornando as aplicações realmente nativas ao dispositivo, assim, facilitando o acesso aos recursos do dispositivo e deixando o código com maior desempenho.

Outra funcionalidade importante é o *Hot Reload* que permite ao programador visualizar rapidamente as atualizações feitas no código do aplicativo. O *Hot Reload* funciona da seguinte maneira: o *framework* injeta arquivos atualizados do código fonte na máquina virtual do *Dart*. Após a máquina virtual atualizar as classes com as novas versões dos campos e funções, o *framework* do *Flutter* remonta a árvore de *widgets* automaticamente (*Flutter*, 2019).

É possível utilizar diversos pacotes desenvolvidos pela comunidade para auxiliar no desenvolvimento do aplicativo. Neste projeto foi feito o uso do pacote de *bluetooth* para o *Flutter* com o objetivo de facilitar o processo de comunicação do aplicativo com o robô.

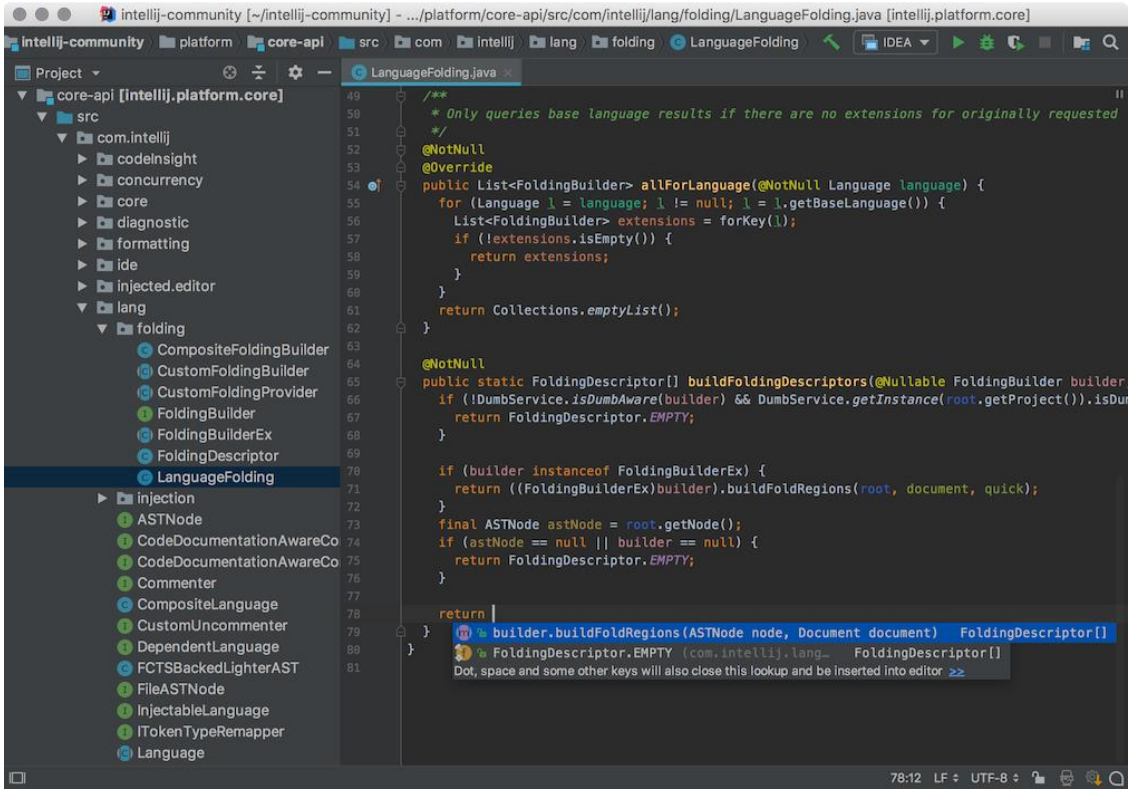
16 - Disponível em: <https://antonioleiva.com/flutter-mobile-development-for-busy-people-droidkast-05/>. Acesso em: 18 jun. 2021.

3.5 IntelliJ Idea

IntelliJ Idea (Figura 25) é uma IDE como a Arduino IDE no sentido de ser um ambiente de desenvolvimento de código. Desenvolvida pela *JetBrains* em 2001, foi uma das primeiras IDEs para programação de Java e serviu de base para a criação de outras IDEs como o *Android Studio* da *Google*.

Ela possui uma grande variedade de linguagens de programação suportadas e *frameworks*, entre eles o *Dart* e o *Flutter*. Além disso possui várias ferramentas para facilitar o uso do software, entre elas o preenchimento automático de código e assistência para o código sendo desenvolvido, explicando o que funções fazem, como utilizar seus construtores.

Figura 25 – Página principal da IDE



Fonte: *JetBrains*[17]

17 - Disponível em: <https://www.jetbrains.com/pt-br/idea/>. Acesso em: 19 jun. 2021.

4 DESCRIÇÃO DO SISTEMA

Nesta seção são abordados os detalhes para a montagem do robô, a programação do Arduino e a criação do aplicativo móvel.

4.1 Construção do Robô

Para iniciar o projeto, foi feito um levantamento das partes necessárias para a criação do robô utilizando como base questões sobre o funcionamento do robô.

Essas questões eram:

a) Como será feita a base do robô?
b) Como será feita a alimentação do robô?
c) Como o robô se comunicará com o aplicativo móvel?
d) Como será feito o controle dos motores?
e) Como a distância percorrida pelo robô será medida?
f) Como tudo será controlado?

Ao analisar essas questões, buscou-se em diversos sites as partes necessárias para montar um robô, resultando nas seguintes escolhas.

O robô é construído em um chassi que já fornece todos os componentes necessários para a base do robô, incluindo rodas, os motores, discos de *encoder*, um botão de liga e desliga e uma base de acrílico.

Para a alimentação dos componentes do robô, usa-se uma bateria de lítio para alimentar a Ponte H e o Arduino, pela sua autonomia e por ser recarregável, mas como sua tensão não seria suficiente, utiliza-se um regulador de tensão MT3608 para obter o valor desejado de 6V. Os demais componentes são alimentados utilizando a saída de 5V do Arduino.

A comunicação do robô emprega o módulo de *bluetooth* HC-05 pela facilidade de uso e configuração, além de sua alta disponibilidade no mercado.

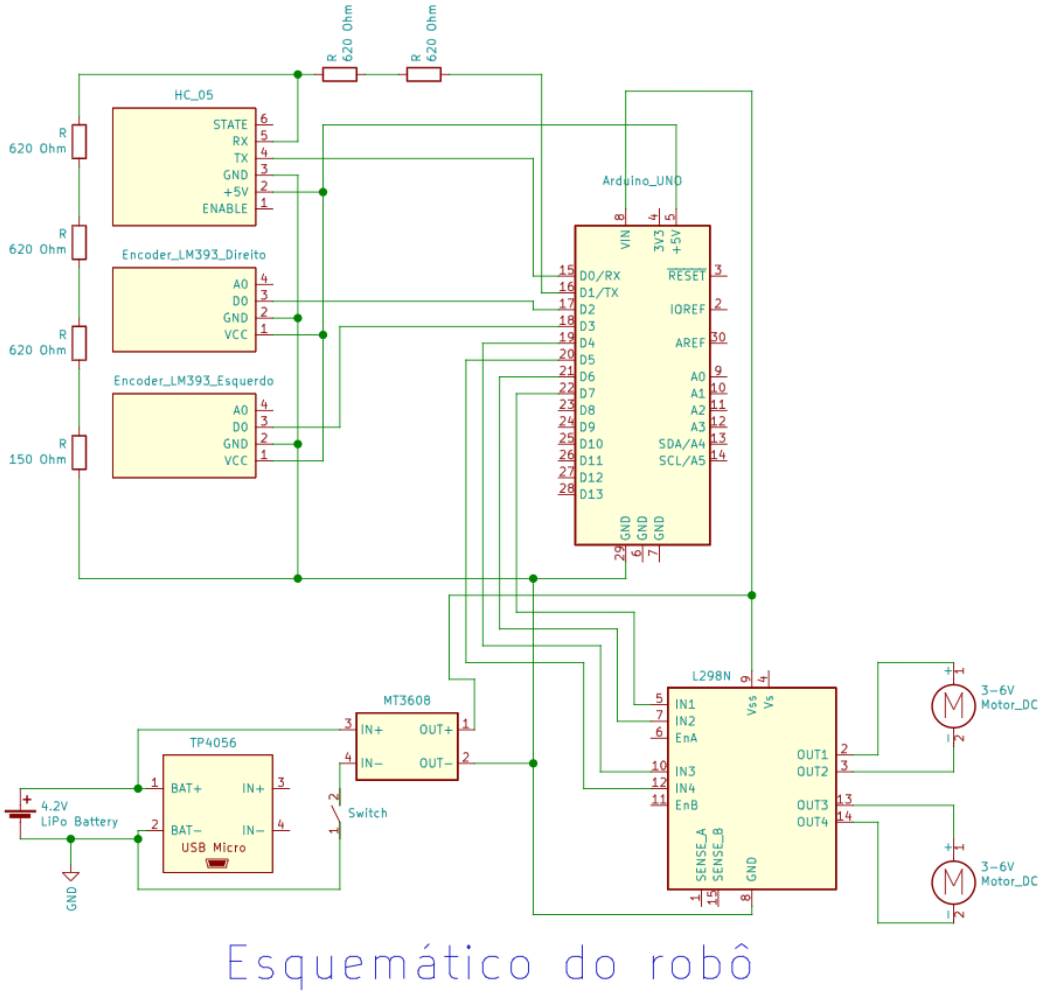
O acionamento dos motores é feito por uma Ponte H L298N. Ela possibilita o controle de dois motores simultaneamente além de oferecer uma possível fonte de alimentação para outros componentes.

Como é preciso saber a distância percorrida pelo robô para controlá-la e possibilitar movimentações exatas do robô, como virar exatamente 90 graus ou andar exatamente 40 centímetros para frente. São usados dois sensores de velocidade *encoder* LM393, que através de um opto interruptor consegue medir a rotação dos discos de *encoder* presentes com o kit chassi do robô.

É utilizado o Arduino Uno devido a quantidade de entradas digitais, a utilização dos pinos TX e RX para comunicação com o módulo *bluetooth* e a possibilidade dele de gerenciar interrupções nos pinos 2 e 3 para medir a rotação das rodas sem interferir na comunicação do robô com o aplicativo.

A Figura 26 demonstra como são as conexões dos componentes do robô.

Figura 26 – Esquemático das conexões entre os componentes do robô

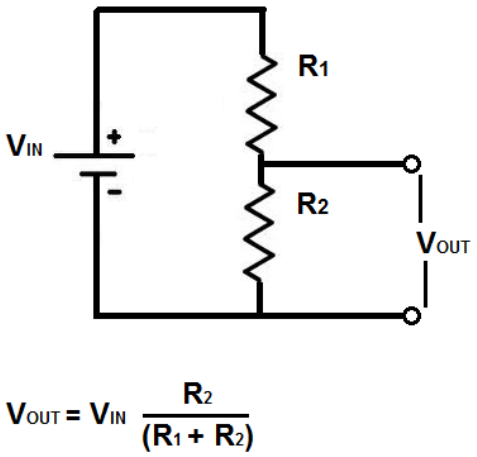


Fonte: *Elaborado pelo autor*

Como pode ser visto, são utilizados oito pinos digitais do Arduino. Para fazer a comunicação serial com o módulo *bluetooth*, utiliza-se os pinos 0 e 1 (RX e TX, respectivamente), os pinos 2 e 3 estão conectados aos sensores de velocidade já que esses pinos permitem o uso da função *attachInterrupt*( ). Essa função permite que o programa obtenha as informações dos sensores de velocidade sem interferir em outras funções do código. Já os pinos 4, 5, 6 e 7 são utilizados para enviar sinais para a ponte H controlar os motores.

Uma aspecto importante da montagem do robô é o divisor de tensão feito entre o pino TX do Arduino e o RX do módulo *bluetooth*. O pino RX do módulo *bluetooth* trabalha somente com 3,3V e o pino TX do Arduino fornece sinais com 5V, assim, o uso de um divisor de tensão se faz necessário. A Figura 27, mostra como funciona o sistema do divisor de tensão e a fórmula utilizada para calcular a tensão de saída.

Figura 27 – Divisor de tensão e fórmula para o cálculo da tensão de saída



Fonte: *Aprendendo sobre a eletrônica*[18]

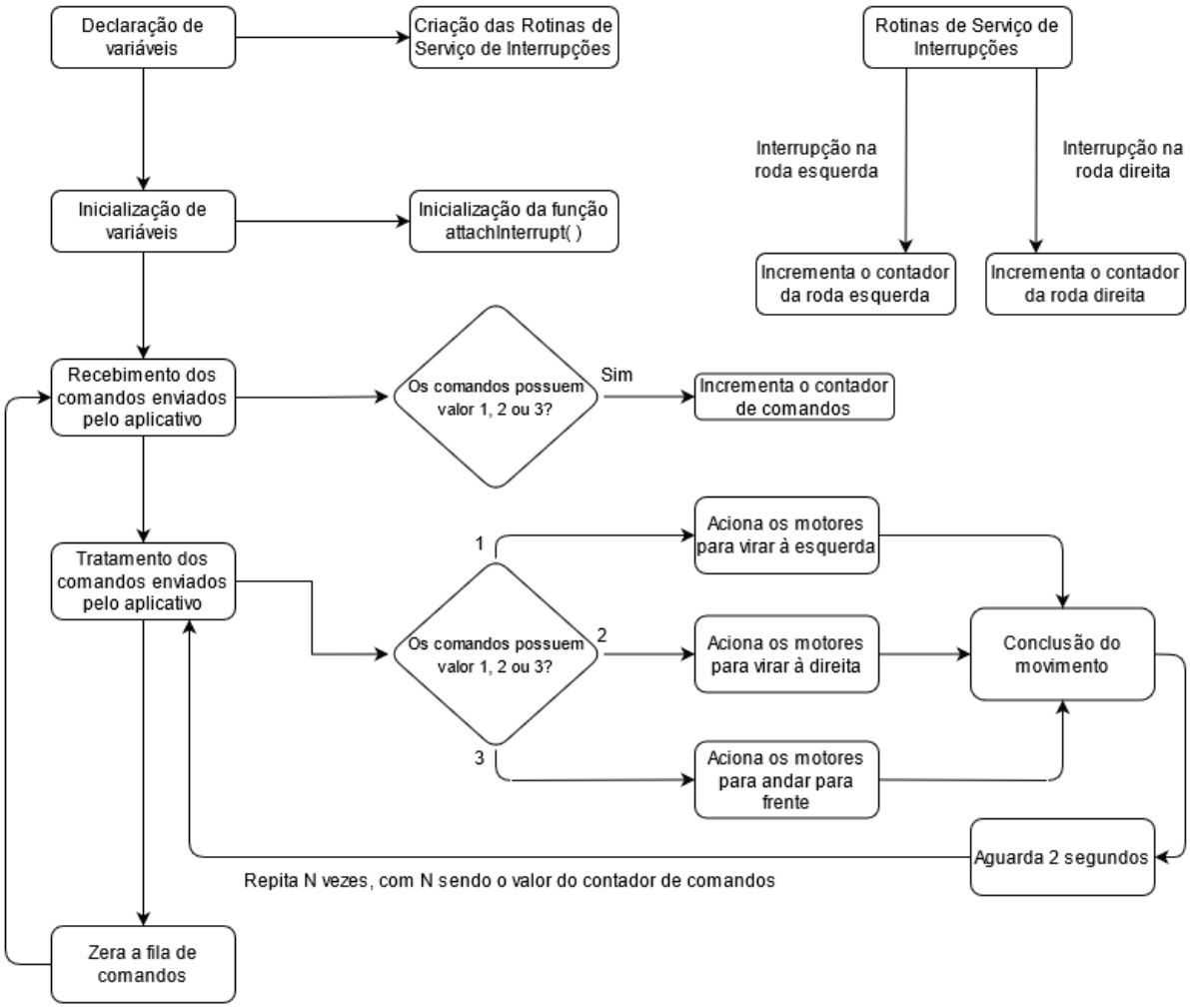
Planejando uma tensão de saída um pouco menor que 3,3V, são utilizados dois resistores de 620 ohms em série como R1 e três resistores de 620 ohms mais um de 150 ohms em série como R2, assim obtendo uma tensão de saída de 3,09V.

18 - Disponível em: <http://www.learningaboutelectronics.com/Artigos/Calculadora-divisor-de-tensao.php>. Acesso em: 22 jun. 2021.

4.2 Código do Arduino

O código do Arduino funciona como uma ponte de comunicação entre o aplicativo e o robô, nele ocorre a recepção das informações enviadas pelo aplicativo, a interpretação dessas informações e então a execução dos comandos como apresenta a Figura 28. Nessa seção serão explicadas cada uma dessas etapas e seu código.

Figura 28 – Diagrama de funcionamento do código do Arduino



Fonte: *Elaborado pelo autor*

4.2.1 Função Setup

Nessa parte do programa é onde ocorre as inicializações de variáveis, a configuração dos pinos, a abertura da porta serial e a chamada da função *attachInterrupt( )*.

4.2.1.1 Função attachInterrupt

Essa função tem grande importância no funcionamento do robô. Ela permite que o Arduino faça outras ações como controlar os motores sem que seja necessário verificar constantemente os sensores de velocidade. Uma vez chamada, a função *attachInterrupt* se utiliza de uma *ISR* (*Interrupt Service Routine*), ou em português, Rotina de Serviço de Interrupção para realizar uma ação toda vez que uma interrupção ocorre no pino indicado. No caso deste projeto, a interrupção seria quando o feixe de luz infravermelho do sensor de velocidade é interrompido devido aos intervalos entre os furos do disco de odometria, possibilitando medir o deslocamento de cada roda do robô.

Para chamar essa função são necessários três parâmetros. O primeiro é a indicação de qual pino ocorrerá a interrupção. O segundo parâmetro é a *ISR* que deverá ser chamada quando houver a interrupção. O último parâmetro é o modo, que define quando a interrupção deve ser ativada. Existem quatro constantes para esse modo:

a) *LOW*: aciona a interrupção quando o sinal do pino for *LOW*;
b) *CHANGE*: aciona a interrupção quando o sinal do pino muda;
c) *RISING*: aciona a interrupção quando o sinal do pino muda de *LOW* para *HIGH*;
d) *FALLING*: aciona a interrupção quando o sinal do pino muda de *HIGH* para *LOW*.

A Figura 29 mostra como essa função é chamada neste projeto.

Figura 29 – Função *attachInterrupt*

```
attachInterrupt(digitalPinToInterrupt(encoderdir), Contador, FALLING);
```

Fonte: *Elaborado pelo autor*

No caso, na primeira linha, o primeiro parâmetro está apontando para o pino 2, que é definido como *encoderdir* que se refere ao disco codificador da roda direita, utilizando *digitalPinToInterrupt*, por ser o modo recomendado de apontar o pino. Contador é a *ISR* utilizada quando a interrupção ocorre e é chamada somente quando o sinal do pino muda de *HIGH* para *LOW*.

4.2.1.2 Rotinas de serviço de interrupções

Esse tipo de função possui limitações que outras funções não possuem. Primeiramente, elas não recebem argumentos e não devem retornar nada. Uma *ISR* deve ser curta e rápida e não pode ser executada em conjunto com outra *ISR*. O programa só pode executar uma *ISR* de cada vez em uma ordem de prioridade escolhida pelo programador.

Como essas funções dependem de interrupções para serem chamadas, funções como *millis( )* e *delay( ),* que também dependem de interrupções, não funcionam se forem chamadas dentro das *ISR*s.

Na Figura 30 pode ser vista a *ISR* utilizada neste projeto. Note que ela é simples, apenas atualizando um contador, não retorna nada e não possui parâmetros.

Figura 30 – Função de Serviço de Interrupções: Contador

```
/*Rotina de Serviço de Interrupções*/
void Contador(){
  pulsos++;
}
```

Fonte: *Elaborado pelo autor*

4.2.2 Função Loop

Essa parte do programa cuida do recebimento e tratamento dos comandos enviados pelo aplicativo móvel e controle dos motores.

4.2.2.1 Recebimento dos dados do aplicativo

Ao ser utilizado, o aplicativo deste projeto envia dados para o robô. Esses dados correspondem aos comandos a serem executados por ele.

Para o robô executar todos os comandos em sequência, é utilizada uma fila que armazena os valores na sequência que forem enviados pelo aplicativo.

Utiliza-se também um contador de comandos que é incrementado a cada comando recebido, para que quando os dados forem tratados, o número de repetições de execuções da função *loop( )* seja o mesmo que o número de comandos.

A Figura 31 que mostra como ocorre o recebimento dos dados no código.

Figura 31 – Recebimento dos comandos enviados pelo aplicativo móvel

```
//Recebe os comandos do aplicativo um por um e os adiciona na fila estados
if(Serial.available() > 0){
  for(int i = 0; i < 100; i++){
    estados[i] = Serial.read();
    if(estados[i] == '1' || estados[i] == '2' || estados[i] == '3'){
      comandos++;
    }
    delay(50);
  }
}
```

Fonte: *Elaborado pelo autor*

Explicando o trecho de código linha por linha, nele é verificado se o Arduino recebeu dados e eles estão disponíveis. Se estiverem, adiciona-se ao elemento *i* do vetor o dado que foi enviado pelo aplicativo. Se esse dado for um comando válido, sendo ‘1’: (virar para esquerda), ‘2’ (virar para a direita) e ‘3’ (seguir em frente), o contador de comandos é aumentado em um e é esperado 50 milissegundos para que o aplicativo envie o próximo dado.

Quando não há mais dados enviados pelo aplicativo celular, a função *Serial.available( )* retorna zero e a recepção dos comandos é encerrada.

4.2.2.2 Tratamento dos dados

A parte de tratamento dos dados segue um caminho simples. A leitura dos dados armazenados no vetor é feita e para cada dado ativam-se os motores de modo que o movimento do robô corresponda ao comando enviado pelo aplicativo. Uma vez em movimento, entra-se em um laço para verificar a quantidade de pulsos do sensor de velocidade utilizando o contador que está sendo incrementado pela *ISR* da função *attachInterrupt*. Quando a quantidade de pulsos chega no valor que corresponde a distância necessária para o movimento, o laço termina e ocorre a espera de 2 segundos para ler o próximo comando.

A Figura 32 apresenta o código para o tratamento dos dados enviados pelo aplicativo móvel e armazenados no vetor *estados*.

Figura 32 – Tratamento dos dados armazenados no vetor *estados*

```
for (int i = 0; i < comandos; i++) {
  pulsos = 0;
  if (estados[i] == '1') { // vira para a esquerda
    do {
      digitalWrite(motor1pin1, HIGH);
      digitalWrite(motor1pin2, LOW);
      digitalWrite(motor2pin1, LOW);
      digitalWrite(motor2pin2, HIGH);
      if (pulsos >= 30) {
        digitalWrite(motor1pin1, LOW);
        digitalWrite(motor1pin2, LOW);
        digitalWrite(motor2pin1, LOW);
        digitalWrite(motor2pin2, LOW);
      }
    } while (pulsos < 30);
  }
  if (estados[i] == '2') { // vira para a direita
    do {
      digitalWrite(motor1pin1, LOW);
      digitalWrite(motor1pin2, HIGH);
      digitalWrite(motor2pin1, HIGH);
      digitalWrite(motor2pin2, LOW);
      if (pulsos >= 30) {
        digitalWrite(motor1pin1, LOW);
        digitalWrite(motor1pin2, LOW);
        digitalWrite(motor2pin1, LOW);
        digitalWrite(motor2pin2, LOW);
      }
    } while (pulsos < 30);
  }
  if (estados[i] == '3') { // anda
    do {
      digitalWrite(motor1pin1, LOW);
      digitalWrite(motor1pin2, HIGH);
      digitalWrite(motor2pin1, LOW);
      digitalWrite(motor2pin2, HIGH);
      if (pulsos >= 150) {
        digitalWrite(motor1pin1, LOW);
        digitalWrite(motor1pin2, LOW);
        digitalWrite(motor2pin1, LOW);
        digitalWrite(motor2pin2, LOW);
      }
    } while (pulsos < 150);
  }
  delay(2000);
```

Fonte: *Elaborado pelo autor*

Utiliza-se o contador de comandos para limitar a quantidade de repetições do *for*. Outro aspecto importante é a função *digitalWrite*, que permite enviar um sinal digital para um pino digital que foi iniciado como pino de saída. Neste caso essa função é utilizada para controlar os motores. Enviar um sinal alto (*HIGH*) para o pino 2 do motor e um sinal baixo (*LOW*) para o pino 1 do motor, faz com que a roda gire para frente e invertendo esses sinais, a roda gira para trás. Assim, é possível fazer com que o robô vire para a esquerda e direita e ande para frente. Quando sinais baixos são enviados para os dois pinos do motor, coloca-se o motor em ponto morto para parar a movimentação do robô e esperar o próximo comando.

Utilizam-se *ifs* para checar se uma roda andou mais que outra. Os contadores de pulsos de cada roda são comparados dentro desses *ifs* e se houver alguma diferença entre eles, a roda que mais se deslocou (com maior valor em seu contador) é parada e a outra roda é acionada para compensar a diferença de distância.

Após a realização de todos os comandos na fila, utiliza-se outro *for* que atualiza o vetor de comandos com valores 0, para que não ocorra a repetição de comandos antigos.

A Figura 33 mostra esse trecho de código.

Figura 33 – Reiniciar o vetor de comandos para evitar a repetição de comandos antigos

```
//Resetar a fila de comandos
for(int i = 0; i < comandos; i++){
  estados[i] = 0;
}
```

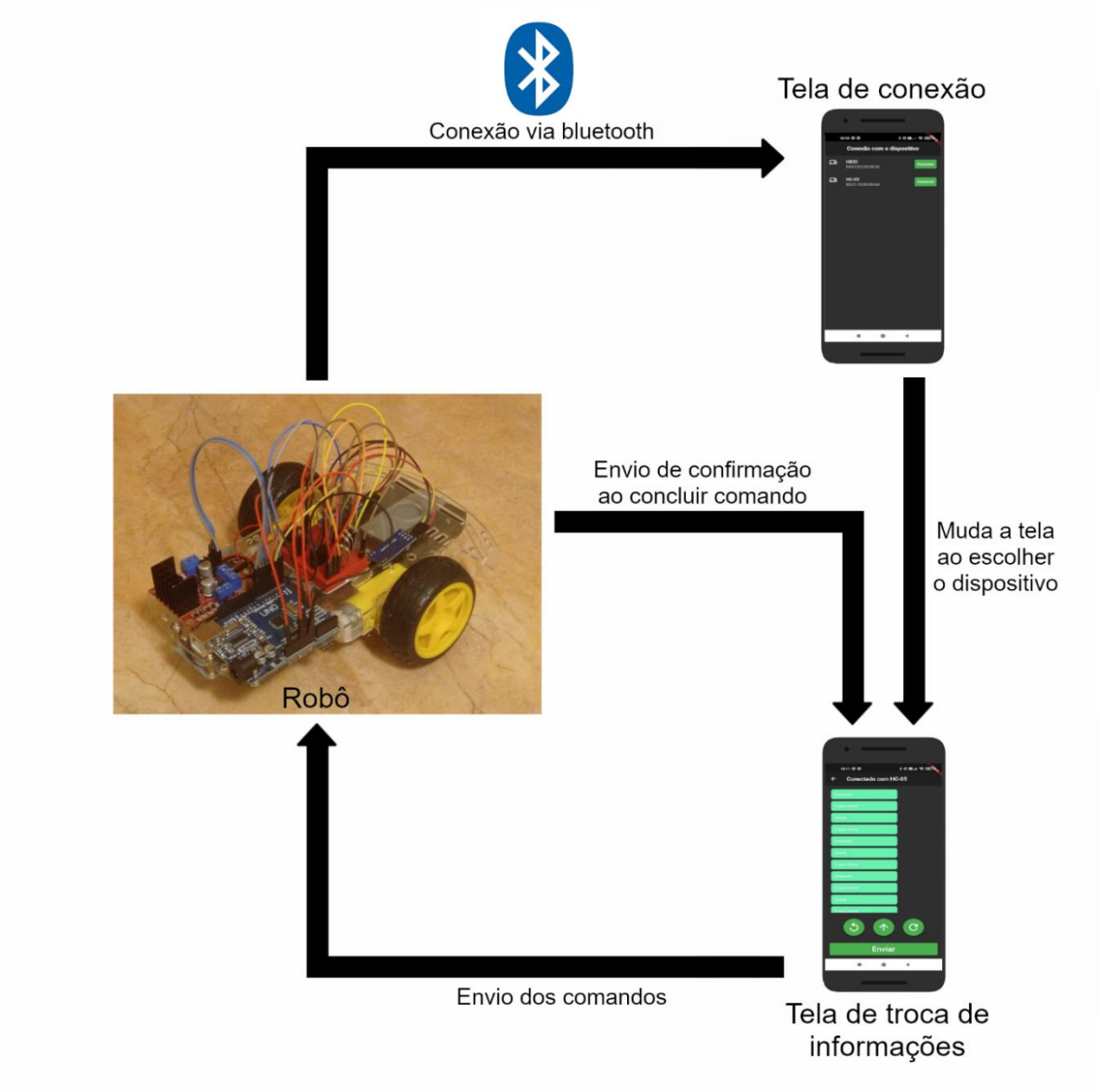
Fonte: *Elaborado pelo autor*

4.3 Aplicativo Móvel

Nessa seção será explicado o código do aplicativo e apresentadas as telas geradas por eles e suas funcionalidades.

O Arduino que equipa o robô se conecta ao aplicativo celular através de *bluetooth*. Uma representação do fluxo de dados trocados entre o Arduino e o aplicativo do celular pode ser apreciada na Figura 34.

Figura 34 – Diagrama do fluxo de dados trocados entre o Arduino e o aplicativo celular



Fonte: *Elaborado pelo autor*

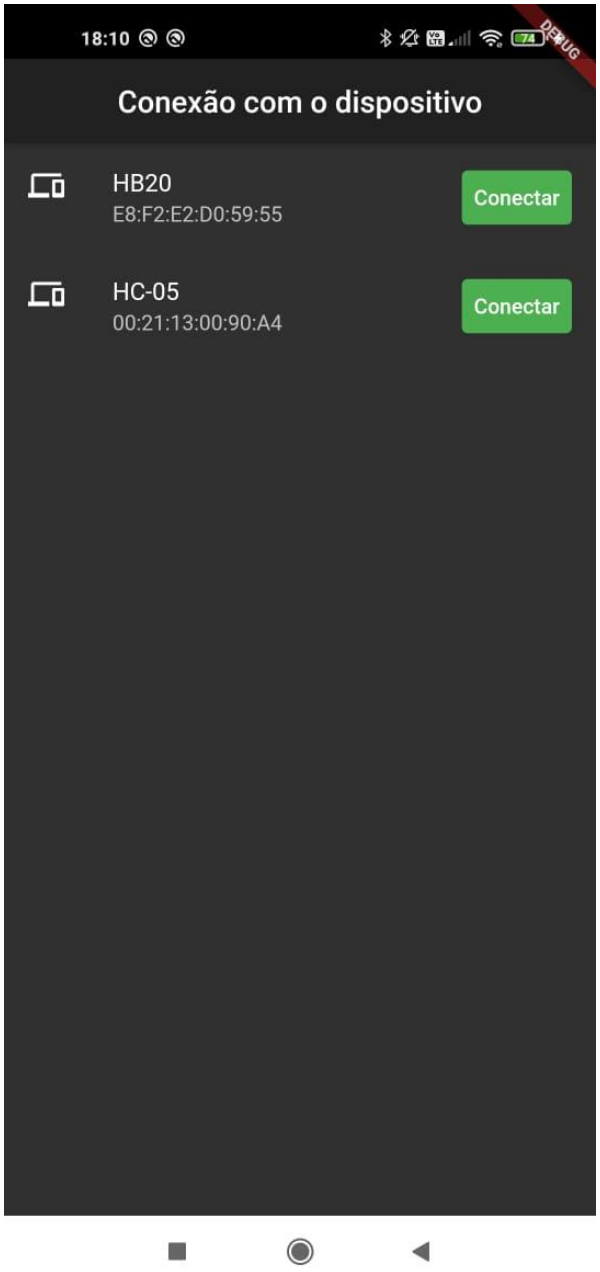
4.3.1 Funcionamento do Aplicativo

Construído utilizando *Flutter*, a proposta desse aplicativo é oferecer um ambiente simples e amigável onde o usuário consiga conectar seu dispositivo móvel a um robô controlado por um Arduino através de uma conexão *bluetooth* e enviar comandos em sequência para serem executados pelo robô. Sua composição consiste em duas telas. A primeira lista todos os dispositivos disponíveis para a conexão *bluetooth* e a segunda apresenta três botões de comando, uma lista que apresenta os comandos selecionados e um botão para enviar os dados de quais comandos foram escolhidos para o robô.

4.3.2 Tela de Conexão

Ao abrir o aplicativo o usuário se depara com a tela de conexão, Figura 35. Nela estão listados todos os dispositivos que foram pareados previamente com o dispositivo. Para realizar o pareamento com o robô, é necessário que o *bluetooth* de seu dispositivo esteja ligado e que o LED presente no módulo de *bluetooth* do robô esteja piscando rapidamente.

Figura 35 – Página de conexão com outro dispositivo via *bluetooth*



Fonte: *Elaborado pelo autor*

4.3.3 Criação da Lista de Dispositivos Emparelhados

Para haver a conexão entre dois dispositivos, primeiro é necessário verificar se o *bluetooth* está disponível. Ao construir a página inicial, o programa verifica se o *bluetooth* do dispositivo está disponível. Caso não esteja, pergunta se o usuário deseja ativar o *bluetooth* e se for negado, a página fica sem ser carregada impossibilitando a conexão com outros dispositivos. No caso do *bluetooth* estar ligado, a lista de dispositivos pareados aparece automaticamente.

Para criar a lista de dispositivos pareados o programa busca todos os aparelhos pareados do dispositivo. Essa busca começa no momento que a tela do aplicativo, apresentada na Figura 35, é aberta. Então é verificado se as informações de cada dispositivo estão corretas. Se sim, o dispositivo é tido como disponível e esse processo é repetido para todos os dispositivos pareados. Ao finalizar, uma variável é alterada para *false* para parar as buscas. A Figura 36 apresenta o trecho de código onde a busca é iniciada e a Figura 37 apresenta a função que realiza a busca.

Figura 36 – Iniciação da busca por dispositivos emparelhados

```
@override
void initState() {
  super.initState();

  _isDiscovering = widget.checkAvailability;

  if (_isDiscovering) {
    _startDiscovery();
  }
```

Fonte: *Elaborado pelo autor*

Figura 37 – Função que realiza a busca de dispositivos emparelhados

```
//Encontra dispositivos
void _startDiscovery() {
  _discoveryStreamSubscription =
      FlutterBluetoothSerial.instance.startDiscovery().listen((r) {
        setState(() {
          Iterator i = devices.iterator;
          //Passa por todos os dispositivos pareados
          while (i.moveNext()) {
            var _device = i.current;
            //Checa se as informações dos dispositivos estão corretas
            if (_device.device == r.device) {
              //Modifica a disponibilidade desse dispositivo para sim
              _device.availability = _DeviceAvailability.yes;
              _device.rssi = r.rssi;
            }
          }
        });
      });

  //Quando acabar, parar de procurar por dispositivos
  _discoveryStreamSubscription.onDone(() {
    setState(() {
      _isDiscovering = false;
    });
  });
}
```

Fonte: *Elaborado pelo autor*

A função initState é chamada sempre que um objeto é colocado na tela. No caso, o objeto é a lista de dispositivos. Ela inicia o estado do *widget*, permitindo mudanças visuais na aparência do aplicativo. Como ela acabou de ser chamada, a variável *_isDiscovering* recebe o valor *true* ao verificar a disponibilidade do *widget* e com isso ocorre a entrada no *if* e a função *_startDiscovery é chamada*.

A função procura por todos os dispositivos pareados utilizando o comando *FlutterBluetoothSerial.instance.startDiscovery().listen((r)*. Agora todos os dados dos dispositivos estão armazenados nessa variável *r*. Então passa-se por toda a lista de dispositivos utilizando o *while(i.moveNext( ))*. No *if* comparam-se as informações para

ver se estão corretas. Se sim, a disponibilidade e a potência do sinal do dispositivo são atualizadas.

Ao terminar de atualizar a disponibilidade de todos os dispositivos, a variável *_isDiscovering* é alterada para *false*, assim, a procura por dispositivos para.

Uma vez que os dispositivos que estão disponíveis são reconhecidos, é criada a lista com eles como mostra a Figura 38.

Figura 38 – Criação da lista de dispositivos pareados

```
// Prepara a lista de dispositivos pareados
//Busca os aparelhos pareados na lista bondedDevices
FlutterBluetoothSerial.instance.getBondedDevices()
    .then((List<BluetoothDevice> bondedDevices) {
  setState(() {
    devices = bondedDevices
        .map(
          (device) => _DeviceWithAvailability(
        device,
        widget.checkAvailability
            ? _DeviceAvailability.maybe
            : _DeviceAvailability.yes,
      ),
    //Cria a lista com os dispositivos disponíveis
    ).toList();
  });
});
```

Fonte: *Elaborado pelo autor*

Primeiro ocorre a busca dos dispositivos emparelhados com a função *FlutterBluetoothSerial.instance.getBondedDevices( ),* que são adicionados a uma lista com *.then((List<BluetoothDevice> bondedDevices)*. Então, adicionam-se à lista *devices* todos os dispositivos que possuem sua disponibilidade como *yes*. A função *.map( )* retorna os dados que coincidem com as condições impostas e, no caso, *widget.checkAvailability* é utilizada para procurar dispositivos que estão disponíveis. O item é adicionado à lista com a função *toList( )*.

Com a lista preparada, é feita a impressão dela na tela com o trecho de código mostrado na Figura 39.

Figura 39 – Impressão da lista de dispositivos pareados

```
@override
//Contrução da lista com os aparelhos disponíveis
Widget build(BuildContext context) {
  //Monta o layout com o nome e botão conectar de cada dispositivo
  List<BluetoothDeviceListEntry> list = devices.map(
        (_device) => BluetoothDeviceListEntry(
      device: _device.device,
      //Quando tocar no dispositivo ir para a página de conversa
      onTap: () {
        widget.onChatPage(_device.device);
      },
    ),
  //Adiciona a lista
  ).toList();
  //Coloca a lista na tela
  return ListView(
    children: list,
  );
}
```

Fonte: *Elaborado pelo autor*

A função *Build* tem o papel de adicionar o *widget* na tela, no caso, adiciona-se o *ListView* que cria uma lista com *scroll* de *widgets*. Utiliza-se novamente a função *.map( )* para obter os dados desejados, no caso, os dados da classe *BluetoothDeviceListEntry*, que cria o formato visual de como os dados estarão disponíveis na tela e o que ocorre quando a tela é tocada. Essas informações estão representadas respectivamente pelos trechos: *device: _device.device* e *onTap: ( ) { widget.onChatPage(_device.device);}*. A Figura 40 apresenta o trecho de código que mostra a classe *BluetoothDeviceListEntry*.

Figura 40 – Classe *BluetoothDeviceListEntry*

```
@override
Widget build(BuildContext context) {
  return ListTile(
    onTap: onTap,
    leading: Icon(Icons.devices),
    title: Text(device.name ?? "Dispositivo desconhecido"),
    subtitle: Text(device.address.toString()),
    trailing: TextButton(
      child: Text(
        'Conectar',
        style: TextStyle(color: Colors.white),
      ),  // Text
      onPressed: onTap,
      style: ButtonStyle(
          backgroundColor: MaterialStateProperty.all<Color>(Colors.green),
      ),  // ButtonStyle
    ),  // TextButton
  );  // ListTile
}
```

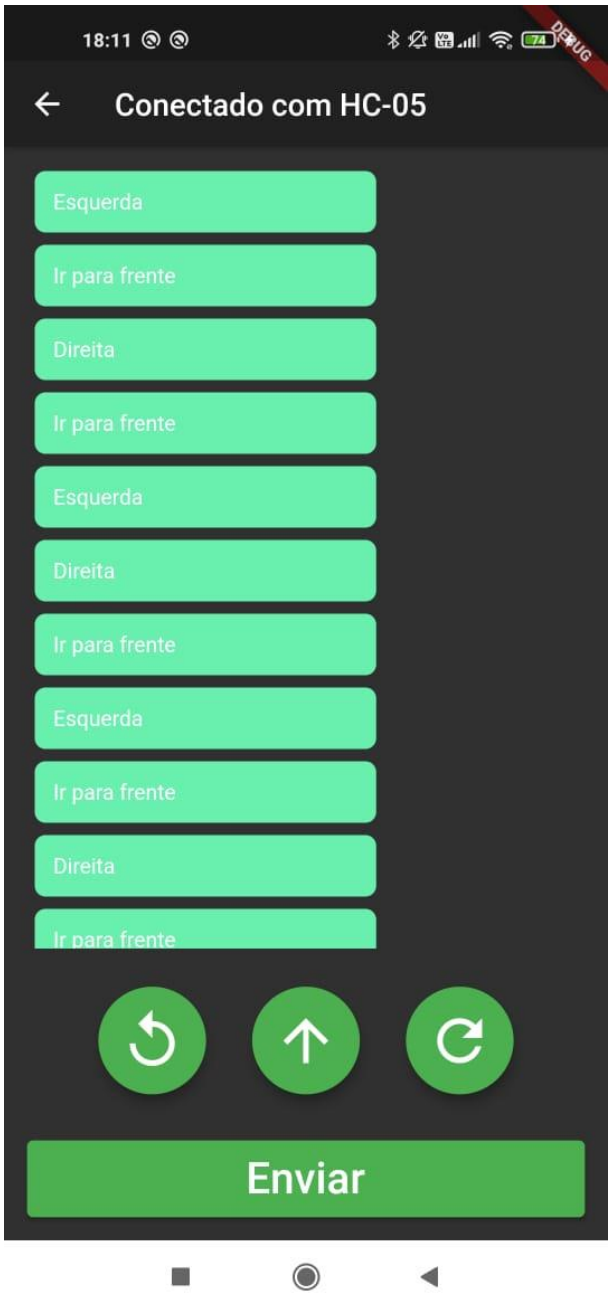
Fonte: *Elaborado pelo autor*

A classe *BluetoothDeviceListEntry* cria uma lista de *widgets* em sequência. Primeiramente, cria uma função *onTap* permitindo que o usuário toque na tela para executar uma ação. Então é utilizado *leading* onde adiciona-se um ícone, seguido de *title* que contém o nome do aparelho se esse nome existir ou “Dispositivo desconhecido” caso contrário e *subtitle* que contém o endereço do dispositivo e se localiza abaixo do nome do aparelho. E por último cria-se um botão de fundo verde com letras brancas escrito “Conectar” que quando apertado executa a mesma ação *onTap* de quando o usuário toca na tela. Essa ação é levar o usuário para a próxima página do aplicativo, que é a página de conversa com o dispositivo selecionado.

4.3.4 Tela de Troca de Informações com o Dispositivo

Após selecionar o dispositivo que deseja se conectar, o usuário é direcionado para a página de conversa com o dispositivo. Seu *layout* consiste em uma barra que notifica o usuário o estado de conexão com o dispositivo (se está conectando ou se não está conectado), um espaço onde os comandos aparecem ao serem pressionados, três botões de comando e um botão de enviar. A Figura 41 mostra esse *layout*.

Figura 41 – Página de conversa com o dispositivo.



Fonte: *Elaborado pelo autor*

4.3.5 Conexão com o Dispositivo

Antes de enviar os comandos para o Arduino, é preciso realizar a conexão via *bluetooth* com o HC-05. Para isso o endereço do dispositivo é procurado e a conexão é gerada. A Figura 42 apresenta o trecho de código que faz essa conexão.

Figura 42 – Conexão com o dispositivo

```
//Conecta com o dispositivo
BluetoothConnection.toAddress(widget.server.address).then((_connection) {
  connection = _connection;
```

Fonte: *Elaborado pelo autor*

A função *BluetoothConnection.toAddress* cria uma conexão *bluetooth* com o dispositivo de endereço encontrado por *widget.server.address*. Então, os dados dessa conexão são armazenados na variável *connection*.

4.3.6 Construção do *Layout* da Página

O *layout* da página é divido em três partes: uma lista que mostra quais foram os comandos selecionados, três botões de comando e um botão de enviar. Nessa seção será explicada uma parte de cada vez.

4.3.6.1 Lista de comandos

Antes de adicionar a lista à tela, é preciso criá-la. A Figura 43 mostra o trecho de código utilizado para esse processo.

Figura 43 – Criação da lista de comandos

```
final List<Row> list = messages.map((_message) {
  return Row(
    children: <Widget>[
      Container(
        child: Text(
            (_message.text.trim()),
            style: TextStyle(color: Colors.white),
        ),  // Text
        padding: EdgeInsets.all(12.0),
        margin: EdgeInsets.only(bottom: 8.0, left: 8.0, right: 8.0),
        width: 222.0,
        decoration: BoxDecoration(
            color: Colors.greenAccent,
            borderRadius: BorderRadius.circular(7.0)
        ),  // BoxDecoration
      ),  // Container
    ],  // <Widget>[]
    mainAxisAlignment: MainAxisAlignment.start,
  );  // Row
}).toList();
```

Fonte: *Elaborado pelo autor*

Cria-se uma lista do tipo *Row*, que é um *widget* que disponibiliza seus *widgets* filhos em ordem horizontal. A função *.map( )* é utilizada para obter os textos armazenados na lista *_message*. Esses textos são os comandos que foram utilizados e armazenados em sequência na lista *_message*. O texto obtido e os *widgets* de decoração são colocados dentro de um *Container* para produzir uma caixa como a exemplificada na Figura 44.

Figura 44 – Caixa com texto gerada ao selecionar um comando

Fonte: *Elaborado pelo autor*

Então utiliza-se a função *MainAxisAlignment.start* para alinhar todas as caixas ao lado esquerdo da tela e adicionam-se esses dados à lista com *toList( )*. Com a lista pronta, ocorre a adição dela à tela com o trecho de código da Figura 45.

Figura 45 – *Widget* que adiciona a lista de comandos à tela

```
Flexible(
  child: ListView(
      padding: const EdgeInsets.all(12.0),
      controller: listScrollController,
      children: list,
    ),  // ListView
),  // Flexible
```

Fonte: *Elaborado pelo autor*

Utiliza-se o *widget Flexible* para fazer com que a lista ocupe todo o espaço não preenchido pelos botões na parte inferior. O *widget ListView* é utilizado como filho de *Flexible* para adotar a característica de ocupar todo espaço vazio. A lista *list* é adicionada com todos os componentes ao *ListView* e adiciona-se um *listScrollController* que permite o *scroll* da lista quando o espaço acabar.

4.3.6.2 Botões de comando

Os botões de comando são adicionados em sequência, horizontalmente e quando clicados, adicionam à lista de comando o texto que representa aquele botão.

O trecho de código a seguir, apresentado na Figura 46, mostra como é feita a adição dos botões de comando.

Figura 46 – Botão de comando “virar à esquerda”

```
Container(
  width: 100,
  height: 100,
  padding: const EdgeInsets.all(15),
  child: FittedBox(
    child: FloatingActionButton(
      onPressed: () {
        _printMessage('1');
      },
      child: const Icon(
        Icons.replay,
        color: Colors.white,
        size: 35,
      ),  // Icon
      backgroundColor: Colors.green,
    ),  // FloatingActionButton
  ),  // FittedBox
),  // Container
```

Fonte: *Elaborado pelo autor*

Container é um *widget* que armazena outros *widgets* dentro de tamanhos específicos, no caso, *width* e *height* com valores de 100. Um *FittedBox* é utilizado para redimensionar o tamanho do botão *FloatingActionButton* para o tamanho do *Container*.

FloatingActionButton é um *widget* que cria um botão circular que executa uma ação quando pressionado pelo usuário e permite a adição de um ícone para representá-lo. No caso dos botões de comando, quando apertados, chamam a função *_printMessage* que trata o texto em seu argumento e o envia para a lista de comandos. A Figura 47 apresenta o trecho de código desta função.

Figura 47 – Função *_printMessage*

```
//Recebe o texto dos botões e passa para a lista de comandos
void _printMessage(String text) async {
  text = text.trim();
  sentMessages.length = position + 1;
  sentMessages[position] = text;
  position ++;
  if(text == '1'){
    text = 'Esquerda';
  }
  if(text == '2'){
    text = 'Direita';
  }
  if(text == '3'){
    text = 'Ir para frente';
  }
  //Adiciona o texto à lista de comandos
  if (text.length > 0) {
    setState(() {
      messages.add(_Message(clientID, text));
    });
```

Fonte: *Elaborado pelo autor*

Ao receber o texto, utiliza-se a função *trim( )* para ter certeza de que não há nenhum caractere indesejado e adiciona-se esse texto à lista de mensagens que serão enviadas ao Arduino. Os *ifs* são utilizados para transformar os valores de cada botão em palavras que são adicionadas à lista de comandos utilizando *messages.add( )*. É importante lembrar que *setState()* deve ser chamado para informar o aplicativo que houve uma mudança interna nos dados e que o aplicativo deve remontar a tela com as novas informações.

4.3.6.3 Botão de enviar

Quando o usuário termina de selecionar todos os comandos que ele deseja enviar ao robô, basta ele apertar o botão “Enviar” para que a comunicação com o módulo de *bluetooth* seja feita e os dados sejam enviados ao Arduino.

A construção do botão de enviar é similar à construção dos botões de comando. A única diferença é que um *ElevatedButton* é utilizado ao invés de um *FloatingActionButton*. O *ElevatedButton* possui um formato retangular e permite somente a utilização de textos como identificação para o botão. Ao ser pressionado, ele chama a função *_sendMessage* que faz o envio dos comandos em ordem para o Arduino. A Figura 48 apresenta essa função.

Figura 48 – Função *_sendMessage*

```
void _sendMessage() async {
  try {
    for(int i = 0; i < position; i++){
      connection.output.add(utf8.encode(sentMessages[i]));
      await connection.output.allSent;
    }
  } catch (e) {
    setState(() {});
  }
}
```

Fonte: *Elaborado pelo autor*

Nessa função utiliza-se um *for* para enviar um comando de cada vez para o Arduino. Utilizam-se os dados da conexão *bluetooth* armazenados na variável *connection* junto com *output.add( )* para enviar o dado na posição atual da lista *sentMessages* dentro do *for.*

Utiliza-se *utf8.encode* para transformar o texto em um inteiro simplificando o recebimento pelo Arduino. Depois de enviar o comando, *await* é utilizado em conjunto com *connection.output.allSent* para ter certeza de que tudo foi enviado corretamente e poder prosseguir para o envio do próximo comando. Esse processo é repetido até que todos os comandos tenham sido enviados para o Arduino.

4.4 Ambiente de Locomoção do Robô

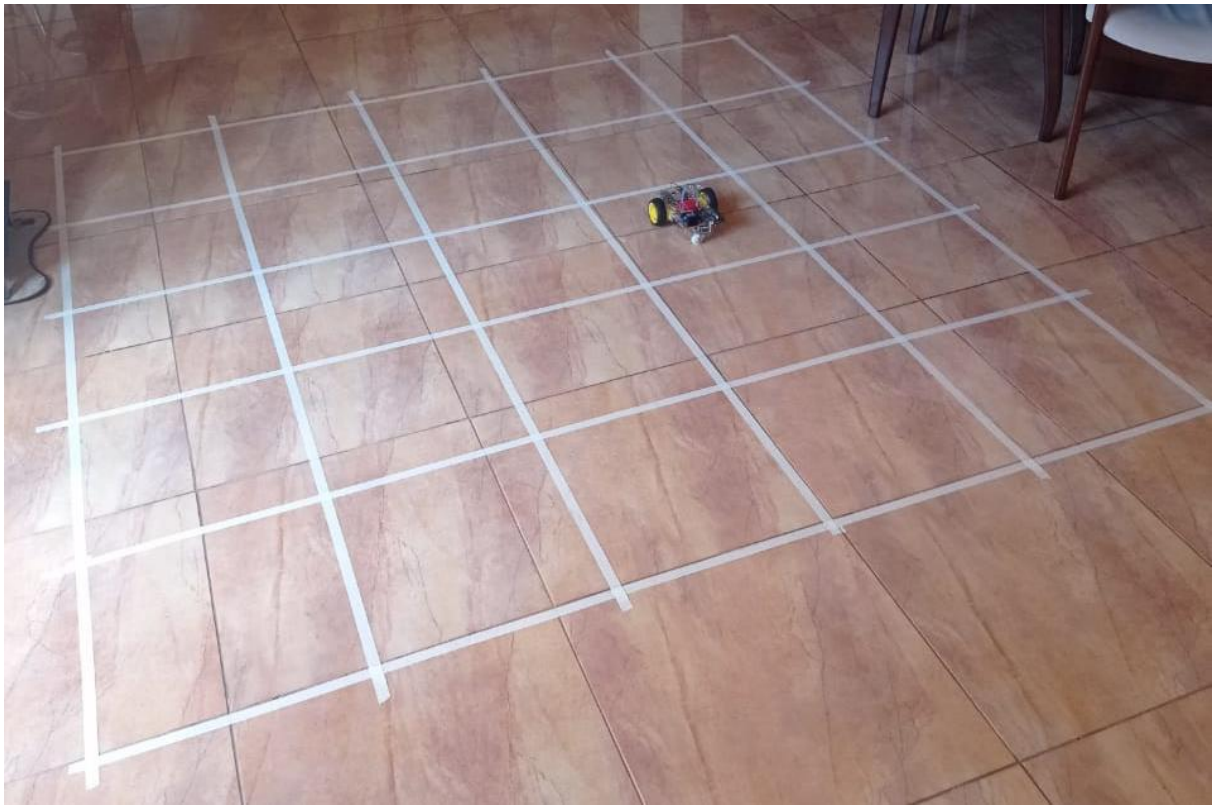
Com o intuito de realizar atividades de programação, é definido um ambiente onde o robô deve operar.

O ambiente de locomoção do robô, apresentado na Figura 49, consiste em uma área de 2 metros quadrados dividida em 25 células de 40 centímetros quadrados.

A intensão deste ambiente quadriculado é fornecer referências para a locomoção do robô e ajudar na compreensão espacial das atividades pelos alunos.

Neste ambiente, uma marcação de destino final e alguns obstáculos podem ser inseridos para tornar as atividades mais complexas.

Figura 49 – Ambiente de locomoção do robô com o robô para noção de escala



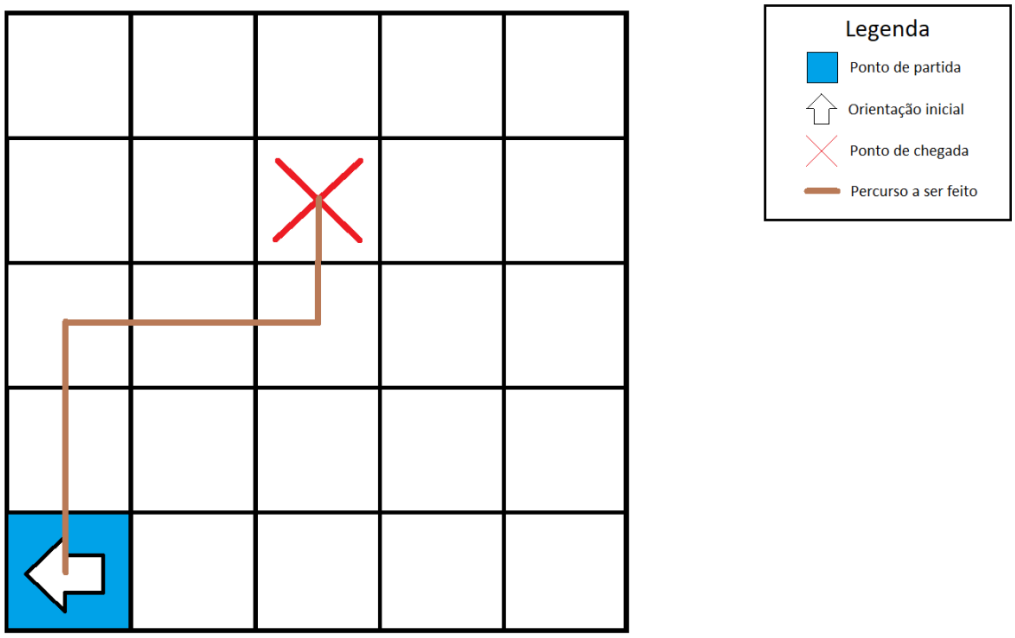
Fonte: *Elaborado pelo autor*

4.5 Atividades

Nessa seção serão explicadas algumas atividades que podem ser realizadas com os alunos e com o robô.

Para a realização dessas atividades, propõem-se a definição de um ponto de partida, uma orientação inicial do robô ou, em algumas atividades, do aluno e um destino. A Figura 50 apresenta um exemplo de como é montado o ambiente para o exercício.

Figura 50 – Exemplo de configuração do exercício com o robô



Fonte: *Elaborado pelo autor*

4.5.1 Atividade com um Aluno Vendado

Para realizar essa atividade, dois alunos são escolhidos. Um é vendado e o outro tem a função de guiar o aluno vendado pelo ambiente de locomoção do robô usando apenas os comandos “dê um passo para frente”, “gire à direita 90 graus” e “gire à esquerda 90 graus”, seguindo um percurso predefinido.

A utilização da venda é comparada com um robô sem sensores para mostrar a importância desses componentes estarem presentes em um robô, assim, espera-se que neste experimento o aluno vendado não alcance o destino com precisão.

4.5.2 Atividade com os Alunos sem Venda

Essa atividade funciona de maneira similar à atividade com um aluno vendado, mas no caso dessa atividade, nenhum aluno necessita utilizar uma venda.

Para realizar essa atividade, dois alunos são escolhidos. Um assume o papel do robô e o outro assume o papel do programador. A função do programador é dar comandos para o aluno com papel de robô. A função do robô é seguir os comandos dados pelo programador e chegar a uma célula específica do ambiente demarcada por um “X” para representar o destino.

O programador pode utilizar comandos como “vire 90 graus para direita” ou “dê um passo para frente” para que o aluno no papel do robô alcançar o objetivo do percurso.

O aluno no papel de robô sem o uso da venda, pode ser comparado a um robô que possui sensores, mostrando como um robô consegue, através desses componentes, se mover de maneira precisa dentro das células do ambiente de locomoção.

4.5.3 Atividade em que os Alunos Controlam o Robô

Nessa atividade os alunos são responsáveis pela programação do robô utilizando o aplicativo móvel para que o robô faça um percurso predefinido.

Objetos podem ser colocados em algumas células com a proposta de aumentar o nível de dificuldade do exercício proposto.

Com a movimentação do robô, os alunos poderão observar a importância dos sensores que medem a rotação da roda, visto que eles permitem a movimentação precisa do robô entre células e ao girar em seu eixo.

4.5.4 Objetivo das Atividades

Estudos da utilização deste robô e sua programação devem ser realizados com crianças para verificar os resultados. As atividades citadas anteriormente podem ser aplicadas em alunos do ensino fundamental, com o objetivo de ensiná-los conceitos básicos de programação como o encadeamento de comandos para resolver um problema. Conceitos de robótica, como o uso de sensores em robôs, alimentação de componentes e a conexão entre eles, também podem ser aprendidos ao estudar o robô.

A realização de um percurso com o robô depende do pensamento lógico do usuário para realizar o encadeamento de comandos necessários para cumprir o objetivo do exercício. A dificuldade do exercício aumentava com o tamanho e complexidade do percurso. Por exemplo, se o percurso do robô é um quadrado, é necessário que o usuário pense no encadeamento de comandos para o robô andar e virar 90 graus para o lado e repetir esses comandos mais três vezes.

O robô e o aplicativo celular que o controla são funcionais e podem ser utilizados como estão. Os percursos ensinam que um programa é composto por uma sequência de passos, iniciando o usuário na programação através de um robô que é um objeto concreto e atraente às crianças, porém, existem trabalhos futuros a ser em realizados, o aplicativo ainda precisa ser desenvolvido para abordar outros conceitos de programação como o uso de funções, condições e laços para que atividades mais complexas possam ser desenvolvidas.

5 CONCLUSÃO

Foi apresentado um robô móvel e uma aplicação celular que permite a programação simples do robô. Através dessa aplicação, alguns conceitos básicos de programação podem ser apresentados, compreendidos e exercitados de forma simples.

Os testes feitos para verificar o funcionamento do robô em conjunto com o aplicativo mostram que é possível criar uma variedade de atividades que visam trabalhar conceitos básicos de programação, e que com a adição de alguns recursos a mais no aplicativo, esses exercícios podem se tornar mais complexos e abranger mais conteúdos da área da computação.

A construção e programação do robô se provaram tarefas interessantes de realizar e que poderiam ser utilizadas como método de ensino por abranger desde conteúdos simples, como lidar com as conexões entre os componentes do robô, até conteúdos mais complexos como fazer a comunicação *bluetooth* com o aplicativo celular.

Mais estudos onde alunos utilizam o robô seriam necessários para comprovar a efetividade do uso deste projeto em sala de aula para auxiliar no ensino de computação.

REFERÊNCIAS

ARDUINO. **Arduino Software**. 2021. Disponível em: https://www.arduino.cc/en/software. Acesso em: 17 jun. 2021.

BIGGS, John. Light-Bot Teaches Computer Science with a Cute Little Robot and Some Symbol-Based Programming. **Tech Crunch**, 2013. Disponível em: <https://techcrunch.com/2013/06/26/light-bot-teaches-computer-science-with-a-cute-little-robot-and-some-symbol-based-programming/?guccounter=1>. Acesso em: 11 jun. 2021.

COUSE, Leslie J.; CHEN, Dora W. A tablet computer for young children? Exploring its viability for early childhood education. **Journal of research on technology in education**, v. 43, n. 1, p. 75-96, 2010.

DART. **Dart overview**. 2019. Disponível em: https://dart.dev/overview. Acesso em: 15 jun. 2021.

FLUTTER. **Flutter architectural overview**. 2020. Disponível em: https://flutter.dev/docs/resources/architectural-overview. Acesso em: 15 jun. 2021.

GOMES, Cristiane; SILVA, Fernando; BOTELHO, Jaqueline; SOUZA, Aguinaldo. A Robótica como facilitadora do Processo Ensino-aprendizagem de Matemática no ensino Fundamental. **Ensino de Ciências e Matemática IV-Temas e Investigações. São Paulo: Editora UNESP Cultura Acadêmica. Disponível em http://books. scielo. org/id/bpkng/pdf/pirola-9788579830815-11. pdf [GS Search]**, 2010.

JETBRAINS. **IntelliJ IDEA**. 2001. Disponível em: https://www.jetbrains.com/pt-br/idea/. Acesso em: 23 jun. 2021.

JUNIOR, João Batista Bottentuit. O aplicativo Kahoot na educação: verificando os conhecimentos dos alunos em tempo real. In: **Livro de atas X Conferência Internacional de TIC na Educação–Clallenges**. 2017. p. 1587-1602.

LEARNING RESOURCES. **Botley the Coding Robot**: Coding for Kids. 2018. Disponível em: https://www.learningresources.com/shop/collections/botley. Acesso em: 18 jun. 2021.

LEGO. **LEGO Mindstorms EV3**. 2013. Disponível em: https://www.lego.com/pt-br/product/lego-mindstorms-ev3-31313. Acesso em: 18 jun. 2021.

LIGHTBOT INC. **Página Inicial**. 2017. Disponível em: https://lightbot.com/. Acesso em: 18 jun. 2021.

LIMA, Priscila; VIEIRA, Paulo; BRANDÃO, Leônidas. Ensino de algoritmos, programação e matemática: panorama e estudo de caso para estudantes de escolas públicas brasileiras. In: **Anais do XXV Workshop de Informática na Escola**. SBC, 2019. p. 697-706.

O'BRIEN, Brenton. How to choose the right type of robot for your classroom. **Meet Edison**, 2019. Disponível em < https://meetedison.com/how-to-choose-the-right-robot-for-your-classroom/>. Acesso em: 10 jun. 2021.

PAPERT, S. Mindstorms: Children, Computers, and Powerful Ideas. 1 ed. New York: Basic Books, 1980.

RUIZ-DEL-SOLAR, Javier; AVILÉS, Roberto. Robotics courses for children as a motivation tool: the Chilean experience. **IEEE Transactions on Education**, v. 47, n. 4, p. 474-480, 2004.

SPHERO. **Sphero BOLT**: Coding Robot. 2018. Disponível em: https://sphero.com/products/sphero-bolt. Acesso em: 18 jun. 2021.

TAHIR, Rabail; ARIF, Fahim. Mobile technology in children education: Analyzing parents' attitude towards mobile technology for children. In: **2015 Science and Information Conference (SAI)**. IEEE, 2015. p. 410-420.

TERRAPIN. **Bee-Bot**. 2016. Disponível em: https://www.terrapinlogo.com/products/robots/bee/bee-bot-family.html. Acesso em: 18 jun. 2021.

VOŠTINÁR, Patrik; KLIMOVÁ, Nika. Experience with Using Robots for Teaching Programming. In: **2019 International Conference on Information and Digital Technologies (IDT)**. IEEE, 2019. p. 536-542.

YU, Junnan; ROQUE, Ricarose. A survey of computational kits for young children. In: **Proceedings of the 17th ACM conference on interaction design and children**. 2018. p. 289-299.