PLATAFORMA DE PROGRAMAÇÃO VISUAL PARA ESP8266

Roberto Luiz Debarba, Miguel Alexandre Wisintainer – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

rldebarba@furb.br, maw@furb.br

Resumo: *Apesar da computação estar presente em todos os setores da sociedade hoje, existe uma carência de conhecimento e interesse da população nesta área. Uma forma de viabilizar o conhecimento científico-tecnológico e ao mesmo tempo estimular a criatividade e a experimentação é a utilização da robótica educacional. Com o objetivo de possibilitar o acesso à robótica em sala de aula, através de alternativas de hardware em relação aos produtos existentes no mercado, buscando facilitar o ensino de lógica de programação e aumentar o interesse dos alunos pelo tema, este trabalho apresenta o desenvolvimento de uma plataforma para programação visual baseada em blocos para suporte ao ensino de lógica de programação nas escolas com o auxílio de robótica educacional, usando o microcontrolador Esp8266. Este trabalho também apresenta o projeto Otto DIY e sua utilização, adaptando seu funcionamento para o microcontrolador em questão. De modo geral, o protótipo apresentou resultados satisfatórios em relação à eficiência ao facilitar o ensino de lógica de programação, conforme os testes aplicados em sala de aula. O uso da robótica também gerou bons resultados, alcançando total satisfação dos usuários.*

Palavras-chave*: Lógica de programação. Robótica educacional. Linguagem de programação visual. Otto DIY. Esp8266.*

1 INTRODUÇÃO

Atualmente, os estudantes do ensino básico estão imersos em um ambiente em que a tecnologia é facilmente percebida: carros, celulares e computadores são exemplos que todos conhecem e muitos utilizam, no entanto, poucos entendem. Estes mesmos estudantes passam boa parte de seu tempo na escola estudando conteúdos de matemática e física e, paradoxalmente, os conceitos que lhes são apresentados parecem distantes (BENITTI et al., 2009, p. 1811).

Von Wangenheim, Nunes e Santos (2014) afirmam que, apesar da computação estar presente em todos os setores da sociedade hoje, existe uma carência de conhecimento e interesse da população nesta área. Complementam informando que uma das razões é a ausência do ensino de computação no ensino fundamental.

Uma forma de viabilizar o conhecimento científico-tecnológico, segundo Benitti et al. (2009), e, ao mesmo tempo estimular a criatividade e a experimentação com um forte apelo lúdico, pode ser proporcionada através da robótica educacional. Hoje esse recurso ainda é pouco utilizado, principalmente no cenário nacional, porém, conforme afirma Sentance et al. (2017), com os últimos avanços em tecnologia educacional, o preço dos recursos permitem aos educadores utilizarem ambientes de simulação baseados em paradigmas de ensino digital, com baixo custo e que podem ser aplicados diretamente nas salas de aulas em larga escala.

Diante do exposto, este trabalho apresenta o desenvolvimento de uma plataforma para programação visual baseada em blocos para suporte ao ensino de lógica de programação nas escolas com o auxílio de robótica educacional, usando o microcontrolador Espressif Esp8266. Acredita-se que a plataforma irá possibilitar o acesso à robótica em sala de aula, através de alternativas de hardware em relação aos produtos existentes no mercado, com o objetivo de facilitar o ensino de lógica de programação e aumentar o interesse dos alunos pelo tema. Os objetivos específicos são: (i) disponibilizar uma plataforma de programação visual baseada em blocos; (ii) permitir a execução dos programas desenvolvidos no microcontrolador Espressif Esp8266; (iii) garantir a execução dos programas de forma direta, sem a necessidade de o usuário interagir com programas terceiros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo descreve os principais conceitos relacionados à robótica educacional e à linguagem de programação visual. Também será apresentado o projeto Otto DIY, fundamental para o desenvolvimento deste trabalho. Ao final, são descritos os trabalhos correlatos.

2.1 ROBÓBITA EDUCACIONAL

A robótica educacional, segundo Benitti et al. (2009), é uma forma de viabilizar o conhecimento científico-tecnológico e, ao mesmo tempo, estimular a criatividade e a experimentação com um forte apelo lúdico. Hoje esse recurso ainda é pouco utilizado, principalmente no cenário nacional. Porém, conforme afirma Sentance et al. (2017), com os últimos avanços em tecnologia educacional, o preço dos recursos permitem aos educadores utilizarem

ambientes de simulação baseados em paradigmas de ensino digital, com baixo custo e que podem ser aplicados diretamente nas salas de aula em larga escala.

A ideia do uso da robótica em educação se baseia fortemente na Teoria Construcionista de Seymor Papert, que une a Teoria Construtivista de Jean Piaget ao uso do computador na educação, conforme afirma Pinto (2011). Para Queiroz, Sampaio e Santos (2016), a robótica educacional permite à criança manipular e controlar objetos concretos e, através destes, observar a materialização dos comandos dados por ela ao computador, processo a partir do qual, tomando-se como base a teoria de Jean Piaget, se estabelece a construção do seu conhecimento.

Para Campos (2011), deve-se considerar que a robótica pressupõe de três componentes: (i) a utilização de kits de montagem para a construção de dispositivos; (ii) o computador; (iii) uma linguagem de programação que permita "dar movimento" ao dispositivo construído.

Por tratar-se de uma disciplina de caráter multidisciplinar, segundo Campos (2011), a robótica permite que os alunos trabalhem uma grande diversidade de competências e habilidades à medida que engloba, em um único objeto de estudo, diversas áreas do conhecimento, como matemática, eletrônica, mecânica, inteligência artificial, artes e programação.

> Os projetos de robótica na educação básica se configuram em sua maioria como práticas isoladas de desenvolvimento de projetos, pois tais projetos são muitas vezes compreendidos como matéria específica de formação técnica, sendo aplicada no ensino profissionalizante de níveis médio ou superior. Além disso, a robótica ainda é vista por educadores e a população em geral apenas como um brinquedo sofisticado, em que pessoas aficionadas pela robótica se encontram em campeonatos e mostras ao redor do mundo. (CAMPOS, 2011, p. 46).

Apesar do cenário descrito pelo autor, a busca pela robótica vem ampliando o contexto tanto das faculdades quanto da indústria. O interesse pelo assunto vem crescendo, a ponto de despertar a atenção do Estado, que voltou a investir e incentivar mais na educação tecnológica. No entanto, são poucas as instituições educacionais de ensino básico que tratam de incluir tópicos relacionados à educação tecnológica nos seus currículos (CAMPOS, 2011, p. 47).

Não é possível entrar em consenso em relação ao uso do termo robótica educacional. Campos (2011) divide o objetivo da robótica na educação básica em três categorias, sem pretensão de defender nenhum dos conceitos: (i) aprendizado de robótica; (ii) robótica como recurso tecnológico na aprendizagem de conceitos interdisciplinares; (iii) integração das duas categorias anteriores. O autor define a primeira categoria:

> [...] corresponde a projetos que visam a aprendizagem de conceitos de robótica propriamente ditos, ou seja, os alunos desenvolvem projetos que têm como objetivo aprender programação de dispositivos, construção de objetos robóticos e aprendizagem inicial de conceitos de engenharia e tecnologia. Nas escolas de educação infantil até o ensino médio essa categoria aparece com mais evidência em cursos extracurriculares. (CAMPOS, 2011, p. 48).

A segunda categoria pode ser definida por:

> [...] robótica é utilizada no desenvolvimento de projetos que evidenciam a aprendizagem de conceitos diversos como matemática, física, artes, lógica, etc. Assim, esse recurso tecnológico permite à escola criar um ambiente diferenciado para o processo de aprendizagem, em que através da criação e programação do dispositivo robótico o aluno possa aprender conceitos de física, matemática, ciências no ensino médio, por exemplo. Embora a utilização da robótica nessa última categoria tenha relação mais direta com as ciências exatas, os projetos desenvolvidos também integram o conhecimento das humanidades (artes, geografia, história e etc.), e também podem ser interdisciplinares, em especial englobando essas últimas áreas. (CAMPOS, 2011, p. 48).

Por fim, a última categoria – integração –, é descrita pelo por Campos (2011) como uma parte que envolve as duas primeiras, com projetos que englobam tanto a aprendizagem da robótica quando de conceitos específicos e interdisciplinares.

Hoje a robótica educacional é explorada comercialmente por diversas empresas. Segundo Resnick (1993), Produtos como LEGO Dacta e LEGO Cybermaster da empresa LEGO e o CoachLab produzido pela Foundation CMA-Centre for Microcomputer Applications são exemplos destes kits para robótica.

> Com o LEGO Mindstorms, produto mais recente da empresa LEGO é possível criar robôs autônomos. Neste caso elabora-se o programa no computador, que a seguir é transferido para o robô. Softwares como o Robolab e Lego Mindstorms apresentam uma interface gráfica com ícones representando sensores, motores, fios, entre outros. A atividade de programar significa conectar os diversos componentes que formarão o robô, seguindo a lógica estabelecida no projeto. (CHELLA, 2002, p. 2).

Seguindo o mercado, algumas escolas aproveitam para explorar o status que a disponibilidade de robótica nas suas salas de aula oferece. Esse status é responsável por uma vantagem competitiva frente ao “mercado” educacional, como afirmado por Campos (2011). No entanto, a maioria utiliza kits que não contemplam a totalidade que o recurso pode oferecer, ou seja, a construção de um dispositivo, a programação através de uma linguagem e a posterior transmissão e execução da tarefa pré-estabelecida pelo dispositivo (CAMPOS, 2011, p. 52).

2.2 LINGUAGEM DE PROGRAMAÇÃO VISUAL

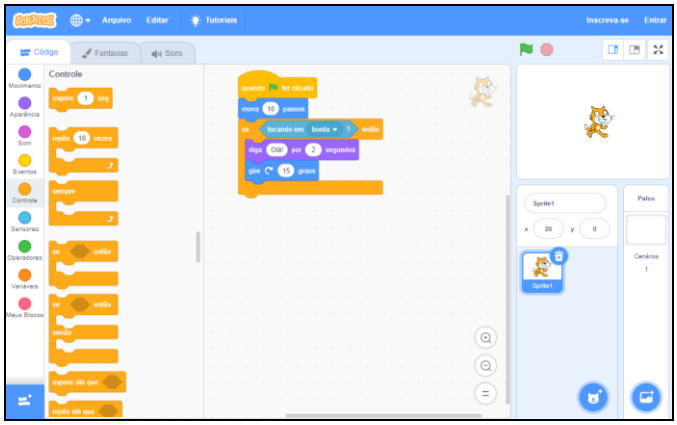
Junto com os primeiros computadores pessoais (PCs), no fim dos anos 70, surgiu o interesse de se utilizar esses equipamentos nas escolas para o aprendizado de programação (QUEIROZ; SAMPAIO; SANTOS, 2016, p. 1170). No entanto, segundo Resnick et al. (2009), a dificuldade de compreensão, por parte das crianças, da sintaxe das linguagens de programação existentes na época, bem como a falta de conexão dos programas desenvolvidos com os interesses delas acabou contribuindo para o insucesso de muitas dessas iniciativas.

Hoje a maioria das pessoas veem a programação como uma atividade técnica e restrita, apropriada apenas para uma pequena parcela da população (RESNICK et al., 2009, p. 62, tradução nossa). Resnick et al. (2009) entende que existem vários fatores para a perda do entusiasmo do uso dos computadores nas escolas e principalmente pela perda de interesse em programar: (i) as linguagens de programação iniciais são muito difíceis de usar e muitas crianças não conseguem dominar a sintaxe; (ii) programação é normalmente introduzida com atividades que não são conectadas aos interesses ou experiências de crianças e adolescentes, como gerar uma lista de números; (iii) a programação é normalmente introduzida em contextos onde não há ninguém que possa ajudar quando as coisas não ocorrem conforme o planejado.

> Desde então, novas iniciativas surgiram no sentido de se tentar vencer estas dificuldades. No que diz respeito ao entendimento da sintaxe das linguagens de programação textuais, uma alternativa encontrada foi o uso de Linguagens de Programação Visuais (Visual Programming Language - VLP), ou seja, linguagens nas quais “a sintaxe (semanticamente significativa) inclui expressões visuais”. (QUEIROZ; SAMPAIO; SANTOS, 2016, p. 1171).

Com base no paradigma de programação visual o MIT Media Lab iniciou em 2003 o desenvolvimento de uma ferramenta de programação visual baseada em blocos, que tinha por objetivo permitir que qualquer pessoa, de qualquer idade, pudesse programar através de blocos de encaixar, apresentam Queiroz, Sampaio e Santos (2016). Essa ferramenta, batizada como Scratch, pode ser visualizada na Figura 1, onde é exibida a tela principal do programa, destacando os principais elementos: (i) a biblioteca de blocos, na esquerda; o programa desenvolvido, ao centro; (iii) a saída do programa, a direita. O sucesso do projeto foi tal que, em 2009, apenas dois anos depois do lançamento do site, usuários de todas as partes do mundo, na sua maioria crianças e jovens de 8 a 16 anos, já faziam upload de mais de 1500 projetos por dia (QUEIROZ; SAMPAIO; SANTOS, 2016, p. 1171).

Figura 1 – Interface principal do programa Scratch



Fonte: Digitalizado pelo autor.

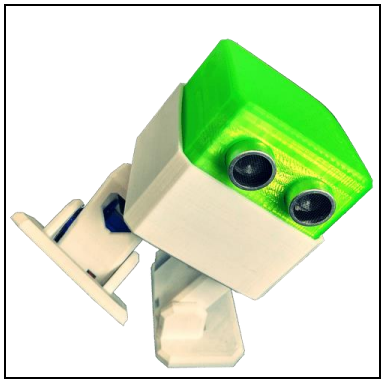
> Além do Scratch, outros projetos de grande vulto, voltados ao ensino de programação para crianças, também adotam o conceito da Programação Visual por meio de blocos de encaixar. Dentre eles podemos citar o Code.org, que possui suas próprias ferramentas de ensino de programação por blocos e o Programaê e Code Club Brasil, que utilizam o Scratch em seus programas de aprendizado de programação. (QUEIROZ; SAMPAIO; SANTOS, 2016, p. 1171).

Gomes Júnior et al. (2016) também cita o MIT App Inventor e novamente o Scratch. Essas linguagens são uma alternativa à forma de programação tradicional. Usar uma linguagem baseada na conexão de blocos de comandos facilita o aprendizado pois garante que os alunos se concentrem muito mais na lógica de programação do que na sintaxe da linguagem.

2.3 OTTO DIY

Segundo Otto Diy (2019), Otto DIY é um robô interativo que qualquer pessoa pode construir. O projeto Otto DIY possui código aberto, tanto do seu software quanto do seu hardware, permitindo que qualquer interessado possa modificar e construir sua própria versão. Compatível com Arduino, o objetivo do projeto é entregar uma oportunidade para as pessoas iniciarem no mundo da robótica, se divertindo enquanto aprendem. Na Figura 2 é mostrado o Otto DIY executando um movimento.

Figura 2 – Otto DIY em movimento



Fonte: Otto Diy (2019).

> (Otto DIY) é mais que apenas um robô; o ato de construir e codificar seu próprio Otto irá criar uma conexão emocional entre você e ele. Otto é um robô que aproxima as pessoas da tecnologia. Aprenda a conexão lógica entre código, ações, a construção de seus componentes e como a eletrônica funciona. (OTTO DIY, 2019, tradução nossa).

Otto Diy (2019) apresenta duas formas de programar o robô: (i) usando uma linguagem de programação visual baseada em blocos. A ferramenta para usar a linguagem é baseada na biblioteca Google Blockly e necessita conexão serial com o robô para gravar o programa criado; (ii) usando a linguagem C++ com o *framework* Arduino através da IDE Arduino, também exigindo uma conexão serial com o robô para gravar o programa criado.

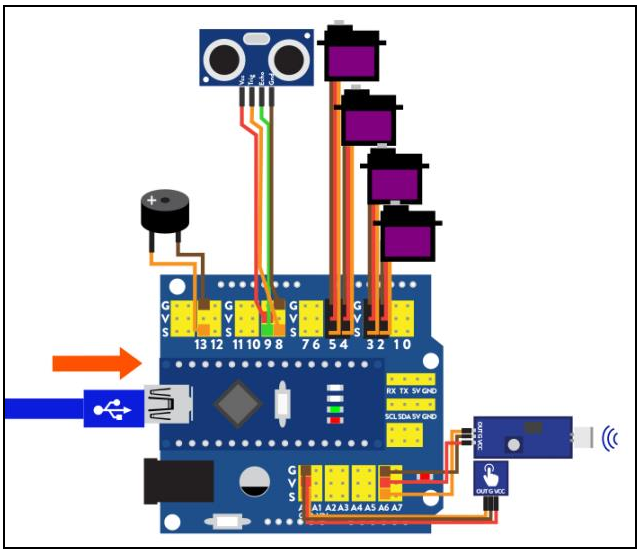
Existem quatro modelos principais do robô no web site oficial do projeto, conforme descrito por Otto Diy (2019): (i) Otto DIY, com a menor quantidade de componentes, impressão 3D, montagem mais simples e preço mais baixo; (ii) Otto DIY+, com mais componentes que a modelo anterior, impressão 3D e preço mais alto; (iii) Otto DIY Humanoid, que também possui braços e maior quantidade de componentes que o modelo previamente descrito, também impresso em 3D; (iv) Otto Papercraftcs, com estrutura de papel criado para dispensar o uso de uma impressora 3D.

No Quadro 10 são listadas as partes que compõem o projeto na versão 9 do modelo Otto DIY+, segundo Otto Diy (2019), junto à quantidade e preço unitário médio para compra no Brasil através da internet, sem considerar valor de frete.

O custo total do projeto usando a bateria LiPo é de R$311,50, ou R$244,30 usando as baterias AA. No site oficial do projeto é vendido um kit de montagem com as mesmas peças (versão com baterias AA) mais a impressão 3D por €29.99, ou R$139.98 considerando a cotação do Euro igual a R$4,67.

Na Figura 3 é apresentado o esquema eletrônico do Otto DIY+, contendo o Arduino Nano anexo ao Arduino Nano Shield I/O como centro do circuito. Também estão representados quatro motores servo, um sensor de distância ultrassônico, um sensor de contato e um *buzzer* ativo. Por último, é exibida uma conexão com um módulo *bluetooth*, com o objetivo de controlar o Otto DIY via aplicativo móvel.

Figura 3 – Esquema eletrônico do Otto DIY+



Fonte: Otto Diy (2019).

2.4 TRABALHOS CORRELATOS

A seguir estão descritos trabalhos considerados similares ao projeto proposto. O Quadro 1 apresenta o projeto Scratch, mantido pela Scratch Foundation (2019), que hoje é referência em programação visual baseada em blocos. No Quadro 2 Von Wangenheim et al. (2017) descreve o trabalho Scratchboard, que estende o projeto Scratch adicionando interação com hardware através da plataforma Arduino. Por fim, o Quadro 3 discorre a respeito do trabalho Micro:bit, realizado por Protzenko (2015), quem tem como objetivo desenvolver uma plataforma de ensino de programação para crianças do Reino Unido através de uma linguagem de programação visual baseada em blocos e plataforma de hardware próprio.

Quadro 1 – Scratch

Referência	Scratch Foundation (2019)
Objetivos	Segundo o autor, o Scratch tem por objetivo ajudar os jovens a pensar de forma criativa, a raciocinar sistematicamente e trabalhar colaborativamente, todas as competências essenciais à vida humana no século XXI.
Principais funcionalidades	A ferramenta permite a programação visual, por blocos, de estruturas básicas, variáveis, condições, repetições, entradas e saídas de dados e abstrações pelo uso de funções. O resultado dos programadas criados é visualizado em tela através de uma animação baseada em atores. É possível criar e programar múltiplos cenários e atores (personagens), que podem ser complementados com a importação e de sons e imagens. O Scratch pode ser executado tanto offline no computador, com Windows, Linux e Mac através de instalação, quanto no navegador via internet.
Ferramentas de desenvolvimento	Não foi possível observar as ferramentas de desenvolvimento utilizadas na construção do Scrtach.
Resultados e conclusões	Não foi possível observar nenhum resultado ou conclusão sobre o trabalho, pois a fonte utilizada é o site do projeto, que apenas apresenta o projeto sem tomar conclusões.

Fonte: elaborado pelo autor.

Quadro 2 – Scratchboard

Referência	Von Wangenheim et al. (2017)
Objetivos	Os autores tinham como objetivo desenvolver uma plataforma de baixo custo para ensino de programação composta pelo microcontrolador Arduino Nano, componentes de hardware, uma extensão para Scratch, plano de ensino e material de apoio.
Principais funcionalidades	A ferramenta disponibiliza o ambiente de programação através do software Scratch, que tem como principal característica o formato de programação visual baseada em blocos. A plataforma Arduino Nano permite a inclusão de sensores e atuadores compatíveis com a mesma, que podem ser adquiridos através de qualquer fornecedor.
Ferramentas de desenvolvimento	Para o protótipo os autores utilizaram como base do hardware a plataforma Arduino Nano anexada à uma placa de circuito impresso com conectores RJ11, utilizados para conectar sensores e atuadores. Para o desenvolvimento da plataforma de programação foi utilizado o Scratch, focando a maior parte do trabalho no compilador responsável por transformar o

	programa em blocos em linguagem Arduino. Os autores não descrevem maiores detalhes sobre a implementação.
Resultados e conclusões	Von Wangenheim et al. (2017) apresenta a aplicação da plataforma através de um workshop para professores do ensino médio que tem como objetivo ensinar a programar um robô super-herói. Os assuntos abordados são comandos e eventos simples, condicionais, repetições, operadores, interfaces de leitura e escrita para componentes externos e como criar e usar funções. Como conclusão, os autores afirmam que que os resultados preliminares indicam que o workshop e a plataforma desenvolvidos são efetivos e motivaram a maioria dos participantes a introduzir computação em suas aulas. O workshop também resultou em professores com entendimento básico de computação e programação, possibilitando a integração da plataforma de forma multidisciplinar em suas aulas. Von Wangenheim et al. (2017) também destaca que a aplicação da tecnologia pelos professores em sala de aula requer treinamentos mais longos para que possam prover o suporte necessário.

Fonte: elaborado pelo autor.

Quadro 3 – Micro:bit

Referência	Protzenko (2015)
Objetivos	O autor tem como objetivo desenvolver uma plataforma de ensino de programação para crianças de 11 a 12 anos de idade no Reino Unido. Com sua utilização é esperado que os estudantes compreendam estruturas de código simples, condicionais, repetições e abstrações via funções.
Principais funcionalidades	A ferramenta é composta por duas partes: hardware e aplicação de programação. O hardware conta com um microcontrolador e uma série de sensores e atuadores embutidos. São eles: um compasso, um acelerômetro, bluetooth de baixa energia (BLE), uma série de pinos de entrada e saída, um sistema de LEDs 5x5 e dois botões. Com os pinos de entrada e saída é possível adicionar sensores e atuadores de terceiros. O ambiente de programação permite o desenvolvimento de programas para o hardware através de uma linguagem de programação visual baseada em blocos. O acesso à ferramenta é online e não necessita nenhuma instalação. O sistema está disponível em português, porém sua documentação é em inglês.
Ferramentas de desenvolvimento	No desenvolvimento do ambiente de programação foi utilizada a biblioteca Blockly. O autor não especifica mais detalhes do desenvolvimento, como esquema eletrônico e linguagens de programação utilizadas.
Resultados e conclusões	O autor declara que apesar da idade e região especificadas originalmente, hoje o projeto abrange diversos países, incluindo o Brasil, e aplica-se a estudantes das mais diversas idades. Não são apresentados outros dados ou resultados obtidos sobre o projeto.

Fonte: elaborado pelo autor.

3 DESCRIÇÃO DA PLATAFORMA

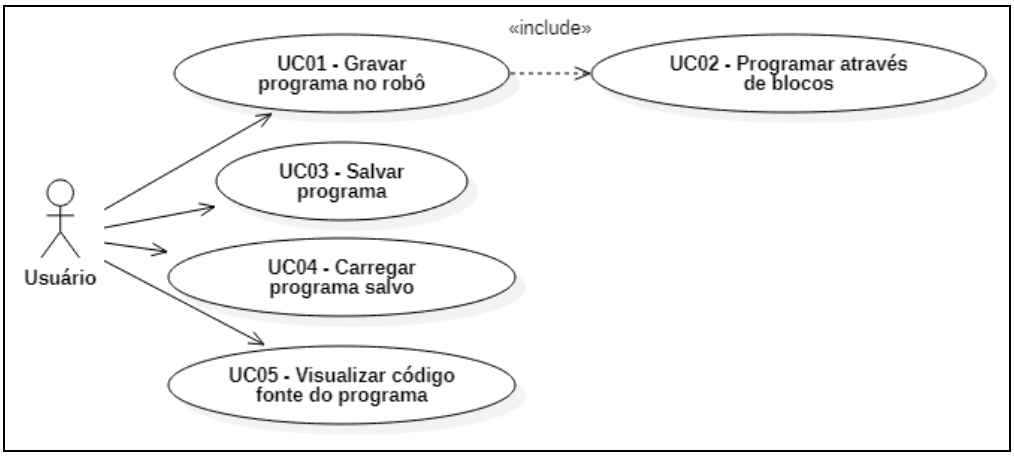
Buscando atender ao objetivo de possibilitar o acesso à robótica em sala de aula para o ensino de lógica de programação e aumentar o interesse dos alunos pelo tema, foi especificada uma plataforma de programação visual baseada em blocos onde o programa criado pelo usuário é executado em um robô. O robô possui uma lista de comandos baseada em movimentos básicos, como andar, virar, dançar e cantar, removendo a complexidade dos conceitos relacionados à robótica focando a atenção do usuário na lógica de programação.

Na plataforma é abordada apenas a segunda categoria que define a robótica educacional, segundo Campos (2011), com foco no uso da robótica como recurso tecnológico para o aprendizado de conceitos interdisciplinares, sendo este trabalho focado nos conceitos de lógica. Sendo assim, este capítulo descreve a especificação da plataforma, apresentando os casos de uso, requisitos funcionais relacionados, visão arquitetural e partes constituintes. Também é detalhado o funcionamento de cada componente, sendo eles *frontend*, *backend* e robô.

3.1 ESPECIFICAÇÃO

Como parte da especificação da plataforma, são apresentados, na Figura 4, os casos de uso. O único ator presente, `Usuário`, pode salvar um programa criado (`UC04`) e carregá-lo posteriormente para continuar a edição ou executar (`UC03`). Nota-se que para gravar um programa no robô (`UC01`) é necessário primeiro realizar sua programação (`UC02`), através da linguagem de programação visual baseada em blocos disponibilizada. É possível visualizar o código fonte, na linguagem C++, do programa criado (`UC05`) mesmo sem nenhum bloco adicionado, tendo acesso, nesse caso, à estrutura inicial.

Figura 4 – Diagrama de casos de uso



Fonte: elaborado pelo autor.

A partir dos casos de uso apresentados na Figura 4, foram definidos os requisitos funcionais (RF) apresentados no Quadro 4, que também relaciona o caso de uso de origem. Em complemento, o Quadro 5 apresenta as definições de disponibilidade, compatibilidade, forma de acesso e tecnologias envolvidas, através dos requisitos não funcionais (RNF).

Quadro 4 – Requisitos funcionais (RF) e relação aos casos de uso

Requisito funcional	**Caso de uso**
RF01 – A plataforma deve permitir a criação de programas através de uma linguagem de programação visual baseada em blocos	UC02
RF02 – A plataforma deve permitir a gravação no robô, através da internet, do programa criado	UC01
RF03 – A plataforma deve permitir salvar o programa criado no disco através de um arquivo	UC03
RF04 – A plataforma deve permitir o carregamento, a partir do sistema de arquivos, de um programa previamente salvo	UC04
RF05 – A plataforma deve executar, através do robô, o programa mais recente gravado	UC01
RF06 – A plataforma deve permitir a visualização do código fonte do programa criado pelo usuário	UC05

Fonte: elaborado pelo autor.

Quadro 5 – Requisitos não funcionais (RNF)

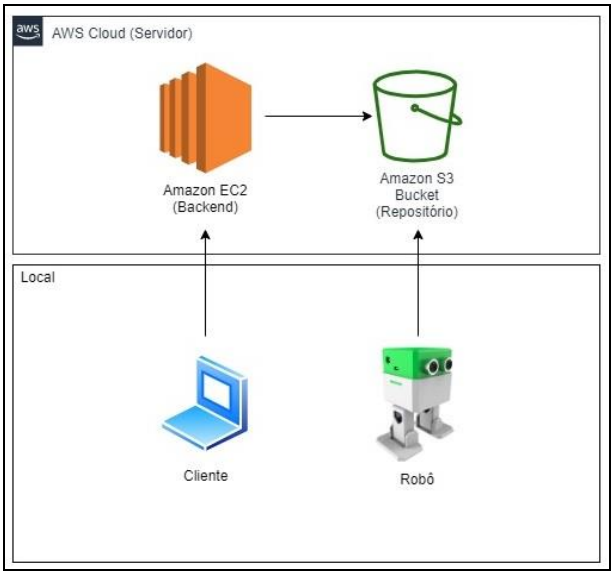
Requisito não funcional
RNF01 – A plataforma deve utilizar a biblioteca Google Blockly para implementar a linguagem de programação visual baseada em blocos
RNF02 – A plataforma deve ser compatível com o navegador Google Chrome versão 78 ou superior
RNF03 – A plataforma não deve exigir nenhuma instalação no computador do usuário além do navegador compatível
RNF04 – A plataforma necessita de acesso à internet
RNF05 – A plataforma deve utilizar a biblioteca PlatformIO versão 4.0.3 para compilar o programa criado pelo usuário
RNF06 – A plataforma deve utilizar o repositório de arquivos AWS S3 para armazenar os programas gerados
RNF07 – A plataforma deve utilizar o microcontrolador Espressif ESP8266 na construção do robô
RNF08 – A plataforma deve utilizar a pilha tecnológica Quarkus para a construção do *backend*

Fonte: elaborado pelo autor.

A Figura 5 ilustra a arquitetura da plataforma. Nela é possível observar dois ambientes e quatro partes constituintes. O primeiro ambiente, `Local`, representa a estrutura que contém o `Cliente` e o `Robô`. Em relação ao mundo real, esse ambiente pode representar o ambiente de ensino. O ambiente `AWS Cloud (Servidor)` suporta a execução do serviço da plataforma e repositório de arquivos. As quatro partes caracterizam: (i) `Cliente`, o computador que executará o *frontend*; (ii) `Robô`, o hardware que executa os programas criados na plataforma através da linguagem de programação visual; (iii) `Amazon EC2 (Backend)`, servidor contendo o *backend* da plataforma, responsável por

processar o programa enviado pelo `Cliente`; (iv) `Amazon S3 Bucket (Repositório)`, repositório de arquivos que mantém a última versão do programa criado pelo usuário, já compilado e neste trabalho denominado *firmware*,.

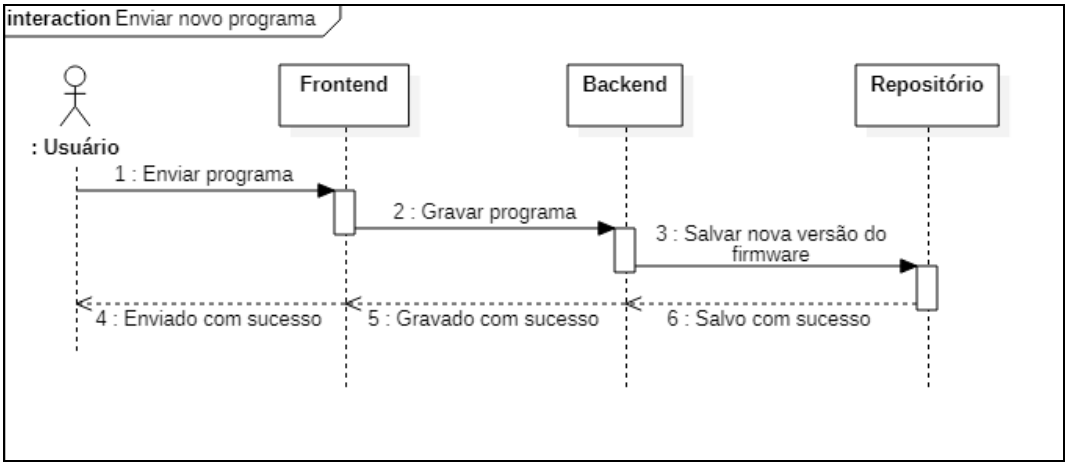
Figura 5 – Diagrama arquitetural



Fonte: elaborador pelo autor.

A Figura 5 faz referência à serviços da plataforma de computação na nuvem Amazon Web Services (AWS), porém o provedor poderia ser substituído por outra empresa, exceto no serviço de repositório de arquivos, cuja troca necessitaria de alterações no código fonte da plataforma. Já na Figura 6 é apresentado o diagrama de sequência de um processo de gravação de um programa elaborado pelo usuário no *frontend*, usando a linguagem de programação visual baseada em blocos. O diagrama considera as etapas de interação entre os componentes da plataforma e tem como objetivo clarificar a fluxo realizado por um programa desde o desenvolvimento até a execução pelo robô. O ator, `Usuário`, inicia o fluxo enviando o programa através de uma funcionalidade do *frontend*. O objeto `Frontend`, realiza uma requisição Hypertext Transfer Protocol (HTTP) para o objeto `Backend` que, após processar o programa, salva a nova versão no repositório de arquivos, representado pelo objeto `Repositório`. Ao final do fluxo uma mensagem é retornada através de todos os componentes, com o valor de sucesso ou falha.

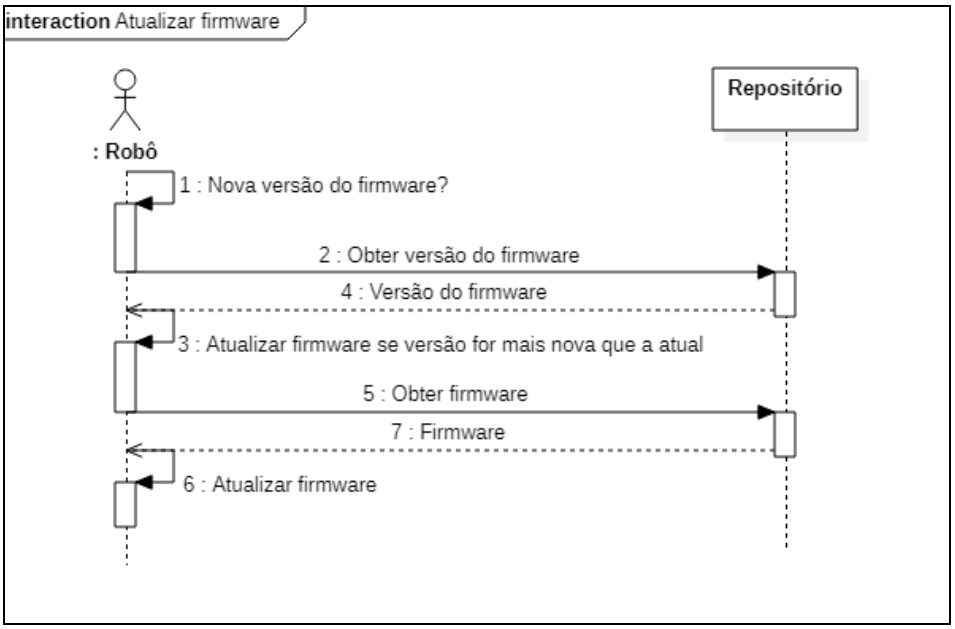
Figura 6 – Diagrama de sequência da gravação deum novo programa



Fonte: elaborado pelo autor.

A Figura 7 apresenta o diagrama de sequência de uma atualização de *firmware*, arquivo binário representado um programa já processado, realizada pelo robô. O processo exposto inicia automaticamente a cada 5 minutos. No diagrama é possível identificar o ator `Robô` que inicia o fluxo enviando uma mensagem para si próprio perguntando se existe uma nova versão. Em seguida esse ator solicita a versão atual do *firmware* ao objeto `Repositório`, que retorna a informação. Caso a versão seja mais nova que a atual, o `Robô` solicita o novo *firmware* e, após obter a resposta, realiza a atualização.

Figura 7 – Diagrama de sequência da atualização do firmware



Fonte: elaborado pelo autor.

Definidos os componentes da plataforma e seus principais fluxos, as demais seções do capítulo detalham o funcionamento independente de cada parte, abrangendo tecnologias envolvidas, responsabilidades e algoritmos.

3.2 PLATAFORMA

A plataforma pode ser definida, com referência em uma observação de alto nível, em três partes, ou componentes. Essa seção discorre sobre cada parte, descrevendo as tecnologias envolvidas, algoritmos utilizados e suas respectivas responsabilidades. O primeiro componente abordado é o *frontend*, focando no uso da biblioteca Google Blockly e na execução da linguagem de programação visual baseada em blocos. Também será abordada a comunicação do *frontend* com o segundo componente, *backend*. Este, por sua vez, tem sua descrição desenvolvida com foco nos passos de processamento do programa criado pelo usuário e sua compilação, junto do processo de envio do *firmware* resultante para o repositório de arquivos. A terceira parte descreve sobre o robô, detalhando sobre os componentes eletrônicos, montagem do hardware, rotina de atualização do *firmware* e papel na plataforma.

3.2.1 Frontend

O *frontend* possui a responsabilidade de realizar a interação com o usuário, iniciando o fluxo de processamento e interação com a plataforma. São definidas como funcionalidades do *frontend*: (i) possibilitar que o usuário crie um programa usando uma linguagem de programação visual baseada em blocos; (ii) possibilitar salvar e carregar posteriormente um programa; (iii) enviar um programa para execução no robô.

A linguagem de programação visual baseada em blocos que possibilita a criação de programas foi desenvolvida usando a biblioteca Google Blockly, que permite a definição dos blocos, criando a interação entre eles e geração do código na linguagem C++. Em conjunto está a linguagem de programação ECMAScript 2019, junto às linguagens HTML 5 e CSS. Bibliotecas adicionais como jQuery ou Angular não foram utilizadas. Para criar uma biblioteca no Google Blockly é necessário definir seus blocos, adicioná-los à um *workspace* e definir o código que será gerado, considerando as entradas e saídas do bloco.

O Quadro 6 apresenta a programação da geração de código de um bloco cuja função é adicionar uma espera de `x` segundos no programa, onde `x` é definido pela entrada do bloco. Nele atribuímos uma função à propriedade `JavaScript` da instância global `Blockly`, que recebe os parâmetros de entrada informados pelo usuário e retorna o código gerado. Nesse caso o código gerado é a função *delay* do da linguagem C++;

Quadro 6 – Programação da geração de código no Google Blockly

```
Blockly.JavaScript['control_waitseconds'] = function (block) {
    let number_seconds = block.getFieldValue('seconds');

    return `delay(${number_seconds * 1000});\n`;
};
```

Fonte: elaborador pelo autor.

A linguagem de programação visual definida tem como objetivo facilitar a aprendizagem de lógica de programação através de blocos para as rotinas *while*, *while(true)*, *if* e *if else*. Em complemento, foram definidos comandos que possibilitam interações básicas com o robô, gerando mais possibilidades na criação dos programas, integrando lógica de programação e ações físicas no mundo real. Divididos em três categorias, os comandos criados são listados a seguir, com o detalhamento de cada:

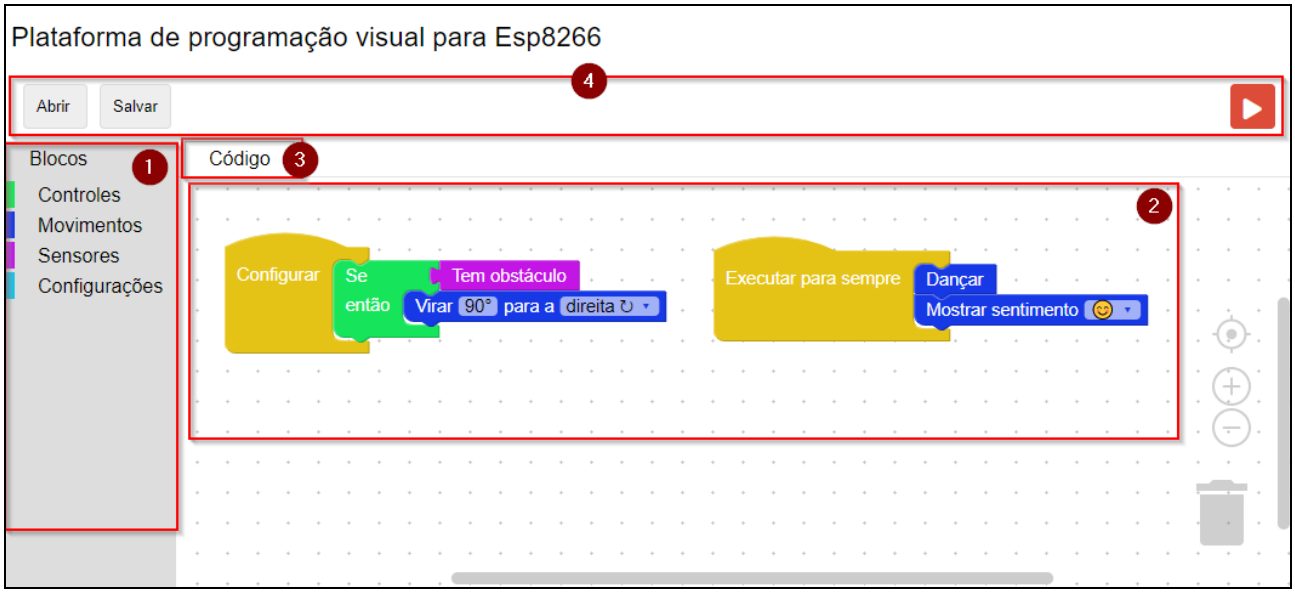
a) controles:
   - esperar x segundos,
   - repita para sempre,
   - repita até (condição),
   - repita x vezes,
   - se (condição) então,
   - se (condição) então senão;

b) movimentos:
   - andar x passos para frente,
   - andar x passos para trás,
   - virar x para a (direita/esquerda),
   - dançar,
   - mostrar sentimento (feliz/triste/confuso/apaixonado/neutro),
   - cantar (felicidade/tristeza/surpresa/feliz aniversário);

c) sensores:
   - tem obstáculo.

A Figura 8 mostra a interface do *frontend*. É exibido, no destaque `1`, a lista de blocos disponíveis para uso. O destaque `2` mostra o *workspace* com o programa criado pelo usuário. É possível visualizar o código fonte gerado a partir dos blocos na aba do destaque `3`. Por fim, o destaque `4` mostra ações adicionais que o usuário pode executar, como salvar o programa, carregar e enviar para o robô.

Figura 8 – Interface gráfica completa do *frontend*



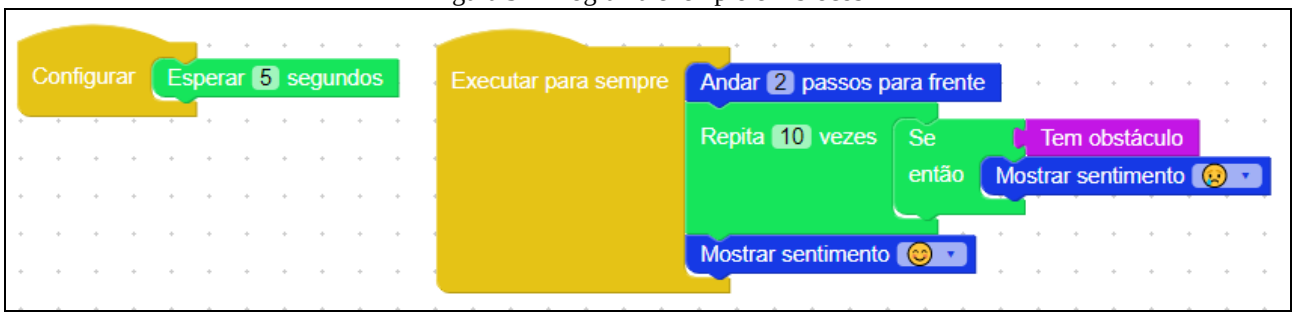
Fonte: elaborador pelo autor.

Ao executar a ação de enviar o programa para o robô, iniciada pelo usuário, o *frontend* realiza a geração do código na linguagem C++ a partir dos blocos selecionados. O código, após gerado com sucesso, é enviado ao *backend* através de uma requisição HTTP POST. Enquanto a requisição não for concluída, o botão de envio permanece desabilitado com um ícone giratório representando o carregamento. Após a conclusão da requisição o botão é novamente habilitado e é exibida uma mensagem com o resultado, sendo esse de sucesso ou falha.

O Quadro 8, no Anexo A, apresenta o código C++ gerado a partir do programa construído para exemplo utilizando a linguagem de programação visual baseada em blocos, apresentado na Figura 9. É possível identificar uma relação direta entre o bloco `Configurar` e a função `setup()`, da mesma forma entre o bloco `Executar` e a função `loop()`. No início do código é importada a biblioteca Otto9, original do projeto Otto DIY, que é responsável pela maioria das ações, realiza-se a definição dos pinos de entrada e saída em constantes. Outro exemplo de conversão simples é o bloco `Andar 2 passos para frente`, que gerou a linha `Otto.walk(2,1000,1);`. No código foram

geradas linhas com os comentários `//DO NOT REMOVE!!!`, que servem como marcações para o *backend* injetar, durante o processamento do programa, códigos necessários para a execução do robô. O processo detalhado é apresentado na seção dedicada ao *backend*.

Figura 9 – Programa exemplo em blocos



Fonte: elaborado pelo autor.

3.2.2 Backend

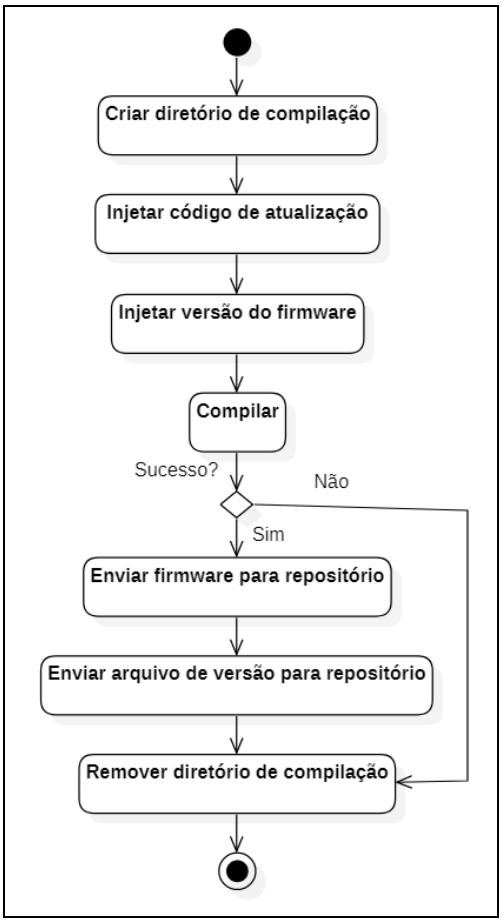
Na plataforma, o *backend* possui a responsabilidade de processar e compilar o código gerado pelo *frontend* na linguagem C++, recebido através de uma requisição HTTP. O código compilado, quando correto, gera um arquivo binário, denominado *firmware*, que deve ser instalado no robô para executar o programa. A instalação ocorre via transferência para o repositório de arquivos. Para possibilitar a compilação, o *backend* necessita realizar diversas operações de processamento, transformação e validação do código fonte, garantindo a integridade do fluxo e a continuidade do processo de atualização. Nesta seção são tratados os detalhes sobre a implementação dos comportamentos descritos, junto à descrição das tecnologias e algoritmos utilizados.

Um possível local de execução do *backend* é a nuvem. Com essa opção, optou-se por tecnologias que permitissem sua construção de forma adequada para o ambiente, considerando velocidade de inicialização, uso de memória, quantidade de bibliotecas disponíveis e maturidade do eco sistema. Como linguagem de programação é usado o Java 11, aderente à pilha tecnológica Quarkus, que substitui os pesados servidores de aplicação, sem abrir mão da especificação Java Enterprise Edition 8 (JEE) e suas funcionalidades. Junto ao módulo *core* do Quarkus, é adicionado a extensão RestEasy, que implementa a especificação contida no JEE Java API for RESTful Web Services (JAX-RS) e entrega a possibilidade de desenvolvimento de APIs HTTP Representational State Transfer (REST).

As maiores responsabilidades do *backend* dentro da plataforma são o processamento e compilação do programa criado pelo usuário. O processo de compilação do código utiliza o ecossistema de desenvolvimento embarcado PlatformIO, que possui uma interface de comunicação via linha de comando chamada PlatformIO Core (CLI), pela qual é possível compilar um código C++ na plataforma Arduino (ou outras), obtendo o resultado de forma automatizada. Essa forma de interação permite que o *backend* delegue a tarefa e responsabilidade de compilação do programa para a ferramenta especializada, diminuindo a complexidade da plataforma. Como o PlatformIO Core (CLI) não é uma biblioteca, mas uma aplicação, seu uso de forma automatizada possui diversas complexidades, tanto na manutenção do ambiente de execução, que deve conter a ferramenta instalada corretamente, quando na comunicação, que deve ser realizada através de parâmetros por linha de comando e da interpretação das saídas em forma de texto. Uma dependência forte combinada com uma comunicação sem contrato definido entre ambas as partes gera uma rotina frágil. Atualizações na versão da aplicação PlatformIO possuem grande probabilidade de exigir modificações no código do *backend*.

A Figura 10 demonstra o fluxo de compilação, iniciada com uma chamada da API REST, que recebe o código fonte C++ gerado no *frontend*. O passo `Criar diretório de compilação` prepara o ambiente onde o novo *firmware* será compilado. É criado um novo diretório com nome gerado pelo algoritmo UUID, garantindo a unicidade, no sistema de arquivos do servidor. Esse diretório é cópia de uma pasta modelo que contém o projeto do *firmware* no modelo do PlatformIO, com suas bibliotecas e dependências. Finalizada a etapa, o ambiente está pronto para receber o código do programa criado e compilar.

Figura 10 – Diagrama de fluxo do processo de compilação



Fonte: elaborado pelo autor.

O passo `Injetar código de atualização`, exibido na Figura 10, modifica o código de entrada do programa adicionando novas funcionalidades responsáveis pela atualização do *firmware* no robô, através de injeção de código fonte. O programa sendo processado até essa etapa possui apenas o código fonte gerado pelos blocos usados pelo usuário, porém durante sua execução uma rotina de atualização precisa ser executada. Sem ela, depois da primeira atualização de firmware novas versões não seriam mais verificadas, pois esse código de verificação não existiria mais no executor. A injeção da rotina de atualização deve ser executada pelo *backend* pois na geração de código no *frontend* o usuário não deve ter acesso a ela, o que dificultaria o entendimento da relação entre código e blocos. A injeção é realizada substituindo três *tokens* em formato de comentários, distribuídos nas linhas necessárias, pelo código da rotina. Este procedimento é detalhado na seção sobre o robô.

O passo `Injetar versão do firmware`, exposta na Figura 10, insere no código do programa uma nova versão única para o *firmware* gerado. Essa versão é usada durante a execução para identificar se a versão disponível no repositório é mais nova que a versão em execução. Para inseri-la no código, é substituído um *token* em formato de comentário pela versão gerada. A geração da versão usa a Era UNIX, ou Epoch *time*, atual. O número 1574721427 é um exemplo de versão válida.

Na Figura 10, o passo `Compilar` salva o código processado no ambiente de compilação criado anteriormente e executa a aplicação PlatformIO Core (CLI) por linha de comando informando os parâmetros para realizar a compilação do código. O resultado da compilação é obtido processando a saída da aplicação. Caso encontre a frase “Compiling program -> SUCCESS” o programa foi compilado com sucesso. Caso contrário, houve um erro na compilação e o fluxo é finalizado retornando uma mensagem de erro para o cliente, no caso o *frontend*.

Ao compilar um programa com sucesso através da integração com o PlatformIO, é necessário enviar o *firmware* gerado para o repositório, atualizar o arquivo de versão e limpar o diretório de execução. Esses passos são representados na Figura 10 por `Enviar firmware para o repositório`, `Enviar arquivo de versão para o repositório` e `Remover diretório de compilação`, respectivamente. O resultado da compilação é salvo em um diretório de caminho fixo no sistema de arquivos. O *backend* carrega esse arquivo e envia para o repositório de arquivos AWS S3, dentro de um *bucket* já criado, cujo nome é configurável. Caso o *bucket* já possua uma versão do

arquivo, ele é sobrescrito. Esse envio é feito através do SDK oficial para Java fornecido pela AWS. Também é enviado um arquivo de texto chamado “version.txt” com apenas a versão do novo *firmware* em seu conteúdo.

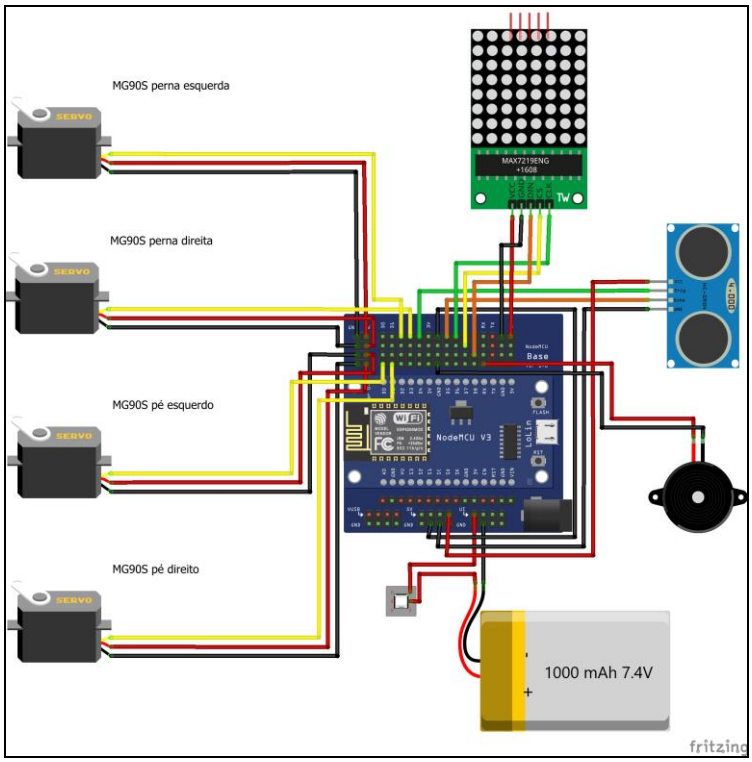
O último passo do fluxo, ilustrado por `Remover diretório de compilação` na Figura 10, é a limpeza do diretório de compilação. A etapa garante que não sejam mantidos arquivos desnecessários no sistema de arquivos do servidor, que causaria um crescimento no espaço em uso comprometendo a capacidade de armazenamento. Esse diretório não é reutilizado em outro fluxo para evitar problemas com arquivos remanescentes que possam modificar o resultado da compilação e problemas de concorrência de acesso.

3.2.3 Robô

O robô é a parte responsável, na plataforma, pela execução do programa criado através da linguagem de programação visual baseada em blocos. Ele é construído com base no projeto de código aberto Otto DIY, com o modelo Otto DIY+. No Quadro 11 são apresentados os componentes eletrônicos necessários para a montagem do robô, junto à quantidade necessário e valor unitário médio no Brasil, comprando pela internet sem considerar frete.

O preço total do projeto é de R$276,50, sem considerar o custo da impressão 3D. O custo dos componentes do projeto Otto DIY, modelo Otto DIY+, é de R$311,50, também não considerando a impressão 3D. A Figura 11 ilustra o esquema eletrônico. Ao centro, é apresentado o microcontrolador Espressif Esp8266, no NodeMCU, anexo à placa base, que tem o objetivo de alimentar o circuito com tensão de 3,3 V e 5 V, sem cortar a alimentação de nenhum componente quando outro exigir maior corrente. A fonte de energia é uma bateria de LiPo recarregável de 1000 mAh com 7,4 V, passando por um *switch* com trava, permitindo desligar o circuito. Também são exibidos um *buzzer* ativo, para reproduzir os sons de sentimento e músicas, e uma grid de *leds*, para exibir sentimentos através de símbolos. Um sensor de distância ultrassônico é usado para identificar obstáculos. A alimentação desse sensor é 5 V, ao invés de 3,3 V como o resto dos componentes, e por isso é necessário adicionar uma ligação entre o terra do circuito 5 V com o terra do circuito 3,3 V. Por último, são apresentados os quatro motores servo, sendo dois usados na articulação do pé do robô e dois usados na articulação das pernas.

Figura 11 – Esquema eletrônico do robô



Fonte: elaborado pelo autor.

A Figura 12 apresenta o robô montado com os componentes descritos. A estrutura é resultado de impressão 3D com material ácido polilático (PLA), resolução de 0.20 mm e preenchimento de 25%. É possível observar alto nível de fidelidade com o projeto original.

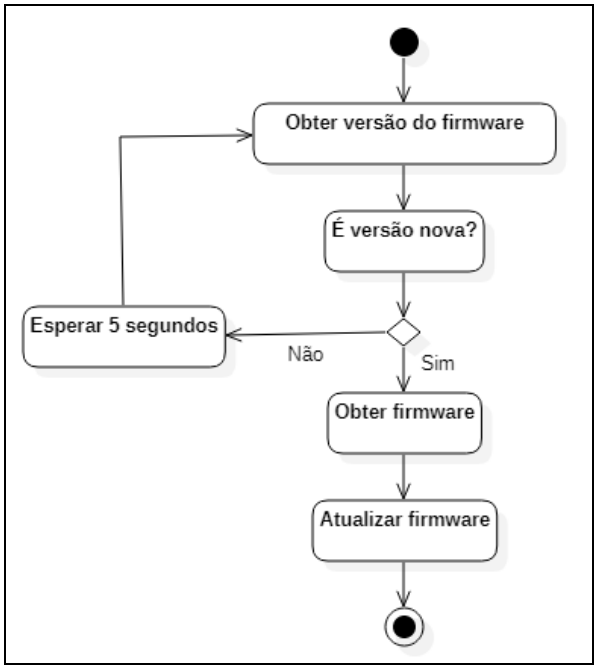
Figura 12 – Robô concluído

Fonte: fotografado pelo autor.

Para a programação do microcontrolador Espressif Esp8266 é utilizada a linguagem de programação C++, com o framework Arduino. Com o objetivo de abstrair a programação dos comportamentos iguais ao projeto original Otto DIY+, como andar, verificar obstáculos e mostrar formas através da grade de LEDs, utiliza-se a biblioteca Otto9, importada do projeto original que possui seu desenvolvimento focado no microcontrolador Arduino Nano ATmega328, resultando em incompatibilidades com o microcontrolador da plataforma. Modificou-se a biblioteca original corrigindo os problemas, principalmente relacionados à erros de compilação. O Anexo B mostra as alterações.

Além de realizar a execução do programa criado corretamente, o robô deve manter-se atualizado em relação ao último *firmware* publicado no repositório. Caso ocorra alguma publicação sem atualização ou uma atualização tardia, acima do tempo de espera aceitável definido pelos requisitos não funcionais, o usuário interpretará seu programa como incorreto, pois a execução não atende a última solução enviada. Com o objetivo de garantir que o problema descrito não ocorra, é executado um processo de atualização do firmware do robô, cujo diagrama de fluxo é apresentado na Figura 13. No primeiro passo, `Obter versão do firmware`, é buscado no repositório a versão do último *firmware* disponível, através de uma requisição HTTP. A versão está no arquivo "version.txt", porém o repositório processa o arquivo e entrega seu conteúdo diretamente em formato texto. Caso o *firmware* instalado seja o mais recente, o processo aguarda por 5 segundos e solicita novamente, como representado pelos passos `É versão nova?` e `Esperar 5 segundos`, repetindo até encontrar um nova versão.

Figura 13 – Diagrama de fluxo do processo de atualização



Fonte: elaborado pelo autor.

Caso o robô encontre uma nova versão do *firmware* executando o fluxo de atualização, ilustrado na Figura 13, é iniciado o processo de obtenção e aplicação. O primeiro passo, `Obter firmware`, obtém o arquivo binário do último *firmware* publicado no repositório de arquivos através de uma requisição HTTP. Se o arquivo for obtido com sucesso, a

atualização é aplicada no dispositivo, representado por `Atualizar firmware`. Esse processo é definido por atualização remota, Over-The-Air (OTA) ou Firmware Over-The-Air (FOTA). Ao final do fluxo o robô é reiniciado e todos seus processos iniciam novamente, buscando novas versões do programa.

4 RESULTADOS

Neste capítulo são apresentados os experimentos realizados, seus objetivos e resultados coletados. Foram aplicados, na maioria dos testes, cenários semelhantes sobre a plataforma apresentada nesse trabalho e o programa Scratch, com o objetivo de obter critérios de comparação, considerando a maturidade e representatividade do programa no mercado. A análise e interpretação dos resultados permitiu obter informações como falhas na plataforma até sugestões de melhorias e pontos de destaque em relação à comparação. Na primeira seção são apresentadas as informações relacionadas ao perfil dos usuários que participaram dos testes. Na segunda seção é apresentada a metodologia, testes aplicados e objetivos. A terceira seção demonstra de forma gráfica os resultados e descreve a análise das informações. A última seção apresenta os resultados dos testes de avaliação, considerado principalmente informações sobre a efetividade da facilitação do ensino de lógica de programação, usabilidade e estabilidade.

4.1 PERFIL DE USUÁRIO

Os experimentos que buscam verificar o atingimento dos objetivos deste trabalho com o uso da plataforma foram realizados com 8 usuários, alunos do primeiro semestre de um curso técnico de informática com habilitação em desenvolvimento de software. A análise do perfil dos usuários foi realizada aplicando um questionário. Os resultados são exibidos no Quadro 7, onde é possível identificar que a idade possui pouca variação, permanecendo entre 17 e 23 anos, com distribuição quase homogênea. Entre os entrevistados, apenas 25% declararam não saber programar, sendo que o tempo máximo de contato com desenvolvimento de software é de 6 meses, onde nenhum deles programa profissionalmente. Quando questionados sobre o contato com a ferramenta Scratch, apenas um usuário declarou conhecimento.

Quadro 7 – Resultado do questionário de perfil dos usuários

Idade	12,5% com 17 anos 37,5% com 18 anos 12.5% com 19 anos 25% com 22 anos 12,5% com 23 anos
Sabe programar	75% declarou que sabe programar 25% declarou que não sabe programar
Programa profissionalmente	100% não tem a programação como profissão
Conhece a ferramenta Scratch	12,5% conhece 87,5% não conhece

Fonte: elaborado pelo autor.

A faixa etária próxima, baixo conhecimento em programação e pouco contato com a ferramenta Scratch se mostrou bastante benéfico para a aplicação do teste, pois a amostra de usuários utilizada representa de forma próxima o público alvo da plataforma. Com as informações de perfil de usuário coletados, prossegue-se para a definição da metodologia e analise de resultado da lista de tarefas.

4.2 METODOLOGIA

Após a coleta das informações de perfil dos usuários, aplicaram-se dois exercícios, sendo um simples e um complexo, tanto na plataforma quanto na ferramenta Scratch, com o objetivo de verificar o funcionamento da criação de programas, identificar de erros, analisar a expectativa dos usuários e fornecer uma visão comparativa aos entrevistados, possibilitando a execução do restante da avaliação. Os exercícios foram apresentados através de listas de atividades. Os usuários devem seguir todos os passos em ordem, sem pular nenhuma questão, e podem tirar dúvidas durante a execução. Deve ser concedido um tempo de 3 minutos no início dos testes em cada plataforma para ambientação com as ferramentas.

Com as listas de atividades concluídas, os usuários são instruídos a preencher um questionário de avaliação com o objetivo de coletar dados comparativos sobre usabilidade, eficiência no ensino de lógica de programação e usabilidade. São disponibilizadas duas tabelas com afirmações sobre a plataforma e o Scratch, onde deve-se assinalar, em um grupo de cinco opções, se discorda totalmente ou concorda totalmente, sendo não concorda e nem discorda a resposta neutra. Também foram solicitados, através de perguntas abertas, os pontos positivos e negativos de cada ferramenta. Por fim, o usuário deve informar qual das ferramentas identifica como a mais intuitiva e qual a mais motivadora para programar.

No final da avaliação, através de texto livre, os entrevistados podem adicionar sugestões para a plataforma e o Scratch, gerando dados para extensões e permitindo a identificação de recursos úteis não presentes.

4.3 LISTA DE TAREFAS

Para executar a lista de tarefas, foi disponibilizado um computador contendo a plataforma e o Scratch com ambiente controlado, onde a execução de ambas as ferramentas foi previamente testada. Também estava disponível o uso de mouse, dispensando o *touchpad*, que poderia dificultar o uso da interface para alguns usuários. Após a execução da lista, as questões apresentadas deveriam ser preenchidas, onde a observação era opcional.

O exercício 1 na plataforma foi concluído por todos os usuários com tempo médio de 2,6 minutos e máximo de 4,83 minutos, mostrando que não houve nenhuma dificuldade que impactou consideravelmente no tempo. Destaca-se que todas as execuções do último passo receberam uma observação informando que o programa foi gravado no robô via cabo, pois a gravação via internet não funcionou, devido a uma instabilidade no sistema. Um entrevistado adicionou uma observação apontando que as opções do bloco `Mostrar sentimento` deveriam ser textuais, significando, possivelmente, dificuldade de interpretação ou visualização dos ícones utilizados. Outra observação apontada foi a dificuldade de identificar a função do bloco `Configurar`, sugerindo mudar o nome para algo com mais significado.

O exercício 1 no programa Scratch foi realizado no tempo médio de 1,95 minutos e máximo 3,67 minutos, mostrando novamente que nenhum entrevistado necessitou uma quantidade de tempo a mais considerável. Todas as respostas indicam que os passos foram executados com sucesso, exceto pelo último onde 1 usuário não conseguiu completar, porém não forneceu nenhuma observação. Três entrevistados adicionaram observações onde apontam que as animações resultantes dos programas são pouco chamativas e não perceberam que estavam acontecendo nas primeiras execuções.

No exercício 2 com a plataforma todos os passos, exceto o último, foram executados com sucesso, com baixa variação de tempo, sendo a média 4,85 minutos. O último passo desse exercício apresentou resultado ruim, onde nenhum entrevisto conseguiu executá-lo. Através das observações identificou-se que mesmo com a tentativa de gravação do programa no robô via cabo, não foi obtido sucesso. Esse comportamento pode significar, além da falha na atualização via internet que foi percebida no primeiro exercício, um problema na geração do código com a ordem específica de blocos usada.

O último exercício, executado no Scratch, obteve resultado igual ao exercício 1 na mesma ferramenta. Com tempo médio de 7,4 minutos, sem grandes variações, o único passo indicado como não executado com sucesso foi o último, porém o avaliado não informou nenhuma observação sobre o motivo. Foi adicionada uma observação sobre o bloco `Mudar fantasia`, onde o usuário relatou que não conseguiu identificar qual o significado do termo “fantasia”, remetendo inicialmente à ideia o que o personagem iria mudar de roupa.

4.4 AVALIAÇÃO

Após executar as listas de tarefas e estabelecer critérios comparativos e maior conhecimento das ferramentas, os usuários preencheram duas tabelas avaliativas, tanto sobre a plataforma quanto sobre o Scratch.

O resultado da afirmativa `A` indica que é fácil localizar os elementos na tela da plataforma e identificar seu significado, onde todos os usuários concordaram, na maioria totalmente, com a afirmação. Com resultado parecido, pode-se afirmar o mesmo para os blocos e sua variedade, segundo respostas das afirmativas `A` e `C`.

As afirmativas `C` e `D` buscam informações sobre a qualidade da execução do programa criado. Metade dos entrevistados discordou ou foi neutro sobre a execução não ter apresentado erros e mais da metade discordou ou foi neutro sobre a execução ser condizente com a interpretação da descrição dos blocos. Esse resultado mostra que a plataforma ainda apresenta instabilidade na conversão da linguagem de programação visual para a linguagem executada pelo robô e, principalmente, na atualização do *firmware* via internet. Em complemento, o tempo de espera entre o comando de execução e o início efetivo também insatisfez 12,5% dos usuários, onde outros 12,5% permaneceram neutros. O resultado dessa afirmativa foi agravado pela execução insatisfatória conforme identificado pelas afirmativas anteriores.

Ter o interesse aumentado pelo uso da robótica, interpretar o programa apenas pela leitura dos blocos e entender o programa através dos blocos e execução do robô são afirmações que 87,5% dos usuários concordam parcialmente ou totalmente, onde apenas 12,5% permanecem neutros, conforme resultado das afirmações `F`, `H` e `I`. Isso mostra que o objetivo educacional de ensinar lógica de programação está sendo satisfatoriamente atingido.

Os pontos positivos da plataforma, segundo questões abertas aos usuários, são a interface intuitiva, simplicidade, uso da robótica e o auxílio no ensino de lógica de programação. Como pontos negativos foram elencados a baixa quantidade de blocos, falha no envio do programa pela internet, quantidade de erros e dificuldade para ler as opções de alguns blocos.

O resultado da afirmativa `A` mostra que 12,5% dos usuários tiveram dificuldades ao buscar os elementos na tela e entender quais seus significados, mesmo resultado da afirmativa `B`, sobre a descrição dos blocos e sua variedade.

12,5% se mantiveram neutros. Esse número mostra que a maior quantidade de funcionalidades no Scratch pode causar confusão em usuários iniciantes.

Em relação à execução dos programas criados, considerando erros, travamentos e animações condizentes, segundo as afirmações `D` e `E`, 25% dos usuários discordam parcialmente sobre sua eficácia. Relacionando esses dados com as demais respostas dos mesmos entrevistados, é possível identificar que o motivo da baixa avaliação é a animação de execução pouco chamativa e muito simples, com movimentos pouco fluídos. O tempo de espera entre acionar a execução e o início efetivo agradou a maioria dos usuários, restando apenas 12,5% com respostas neutras.

O entendimento do programa através da leitura dos blocos e visualização das animações durante a execução, segundo as afirmações `H` e `I`, é clara para a maioria dos usuários. Nenhum entrevistado discordou das afirmações.

Com maior discordância sobre resultados obtidos na plataforma, na afirmativa `F`, que descreve que a animação de execução aumenta o interesse em criar novos programas, metade dos usuários se mostraram neutros e a outra metade afirmou concordar parcialmente ou totalmente. Em comparação à execução no robô, todos os usuários concordaram, onde 75% concordaram totalmente. Esse número suporta o uso da robótica educacional como forma de alcançar os objetivos deste trabalho.

Como pontos positivos, através de questões abertas, os entrevistados apontaram as mesmas características citadas na plataforma e a grande quantidade de blocos como item adicional. Nos pontos negativos foram citadas as dificuldades em encontrar blocos e identificar suas funções devido à grande quantidade disponível. Também foram descritas as dificuldades em iniciar sem explicações e o uso de animações de execução muito simples.

Para o Scratch, foi descrito um modo de execução com animações mais robustas como sugestão de melhoria, junto à um modo iniciante com menos opções e blocos mais explicativos. Para a plataforma, as sugestões de melhorias foram adicionar mais blocos e tornar os blocos fixos iniciais mais explicativos.

5 CONCLUSÕES

Através do desenvolvimento deste trabalho, permitiu-se verificar a eficiência da plataforma criada para facilitar o ensino de lógica de programação em sala de aula, aumentando também, através da robótica educacional, o interesse dos alunos pelo tema. Os resultados obtidos através da metodologia avaliativa aplicada mostram que os entrevistados apontaram a plataforma como uma boa ferramenta para aprender lógica, na maior parte concordando totalmente que o uso do robô aumenta o interesse pela aprendizagem. O percentual de respostas muito positivas sobre a visualização do resultado através de um robô foi ligeiramente maior que a obtido com a fermenta Scratch, solução referência em ensino de programação através de uma linguagem visual baseada em blocos no mercado, que usa animações para apresentar o resultado do programa. Desta maneira, o objetivo foi atendido.

O uso de uma linguagem de programação visual baseada em blocos é outro objetivo atendido. Os resultados comprovam que na amostra de usuários entrevistados a maioria concorda sobre o uso da linguagem criada como ferramenta de programação da plataforma. Em comparação ao Scratch, a plataforma apresentou uma linguagem mais fácil de usar devido, principalmente, à menor quantidade de blocos disponíveis.

O objetivo de realizar a execução dos programas criados na plataforma através do hardware Espressif Esp8266 foi atendido. Como o uso do microcontrolador é parte central da construção do robô, os resultados que mostram sucesso na execução dos programas também indicam o sucesso no uso do componente eletrônico. Apesar da falha ao enviar os programas via internet em alguns cenários durante a aplicação dos testes, quando gravados via cabo serial a execução ocorreu conforme esperado em metade dos testes. O atingimento de um resultado abaixo de 100% mostra que a rotina de geração do código ou a construção do robô ainda possui erros. Um problema identificado durante o desenvolvimento do hardware foi a falta de entradas/saídas de uso genérico no NodeMCU, módulo de desenvolvimento usado contendo o microcontrolador. O projeto inicial possuía também um sensor de toque, que serviria como mais um bloco condicional permitindo, por exemplo, criar programas onde são feitas ações ao tocar na cabeça do robô. Mesmo sem o uso do sensor, uma entrada/saída genérica continuou faltando. A solução foi usar a saída RX, porém quando um programa é gravado via serial, é necessário desconectar esse pino, caso contrário a conexão não é estabelecida.

O último objetivo apresentado, referente à garantia de execução dos programas de forma direta, sem a necessidade de o usuário interagir com programas terceiros, não foi atendido totalmente. Durante a aplicação dos testes, não foi possível atualizar o robô via internet, solução escolhida para tratar o objetivo, sendo necessário copiar o código disponibilizado pelo *frontend* e realizar a compilação e gravação do *firmware* manualmente. A falha na atualização é resultado da presença de erros na rotina presente do robô, que muitas vezes não consegue realizar uma conexão válida com o servidor, tanto para obter a versão quando para obter o *firmware*, ou na execução da biblioteca OTA. Como sugestão de melhoria, buscando maior estabilidade no processo, poderia ser realizada a substituição da verificação de versão via requisição HTTP pela inscrição em um tópico através do protocolo Message Queue Telemetry Transport (MQTT), reduzindo o tempo gasto pela rotina, possibilitando executá-la com mais frequência, sendo hoje invocada apenas uma vez a cada iteração do programa.

A linguagem de programação visual baseada em blocos definida neste trabalho, apesar de ter obtido bons resultados durante a sua validação, ainda pode ser melhorada. A principal necessidade desta parte da plataforma é o aumento na quantidade de blocos disponíveis, adicionando novas ações ao robô, como mais movimentos e outros usos para os sensores existentes, além da adição de rotinas de programação, como o uso de variáveis e procedimentos.

A adição da possibilidade de gravação dos programas no robô via cabo USB, sem a necessidade de interação do usuário com ferramentas de terceiros, apenas com o acionamento de um botão, também se mostra útil. Além da estabilidade que a conexão com a internet e suas rotinas dependentes pode apresentar, conforme visualizado nos resultados, alguns locais de interesse para a aplicação da plataforma podem não conter acesso à internet, impedindo seu uso. A adição de cabos seria uma solução de complexidade menor do que a apresentada neste trabalho e atenderia a necessidade dos locais com problemas de conexão.

Outra possibilidade de extensão é o desenvolvimento do *frontend*, com a linguagem de programação visual baseada em blocos, como aplicativo para dispositivos móveis. Com a popularidade dos *smartphones* essa nova interface expandiria o número de usuários com recursos deficientes para usar a plataforma. A conexão para gravação do programa por internet também pode ser substituída por uma conexão *bluetooth*.

Além da plataforma elaborada, o presente trabalho mostrou-se relevante por apresentar o projeto Otto DIY, pouco tratado por outros autores, com seu funcionamento e aplicações, junto à substituição do seu microcontrolador pelo Esp8266, entregando uma alternativa de custo e funcionalidades ao projeto.

REFERÊNCIAS

BENITTI, Fabiane Barreto Vavassori et al. Experimentação com Robótica Educativa no Ensino Médio: Ambiente, atividades e resultados. In: WORKSHOP DE INFORMÁTICA NA ESCOLA, 15., 2019, Porto Alegre. **Anais...** Porto Alegre: Wie, 2009. p. 1811 - 1820.

CAMPOS, Flavio R. **Currículo, tecnologias e robótica na educação básica**. 2011. 243 f. Tese (Doutorado em Educação) – Curso de Pós-Graduados em Educação: Currículo, Pontifícia Universidade Católica de São Paulo, São Paulo.

CHELLA, Marco Túlio. **Ambiente de Robótica para Aplicações Educacionais com SuperLogo**. 2002. 186 f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Estadual de Campinas.

GOMES JÚNIOR, Josué da Silva et al. Um ambiente baseado em blocos para programação paralela com OpenCL. **International Journal Of Computer Architecture Education**. Paraíba, v. 5, n. 1, p. 38-43, dez. 2016.

OTTO DIY (Czech Republic). **Otto DIY**. Staré Brno, [2019?]. Disponível em: <https://www.ottodiy.com/>. Acesso em: 20 nov. 2019.

PIAGET, Jean. **To understand is to invent**: the future of education; right to education in the modern world. New York: The Viking Press, Inc., 1974. 148 p.

PINTO, Marcos C. **Aplicação de Arquitetura Pedagógica em Curso de Robótica Educacional com Hardware Livre**. 2011. 158 f. Dissertação (Mestrado em Informática), Instituto Federal do Rio de Janeiro, Rio de Janeiro.

PROTZENKO, Jonathan. Pushing blocks all the way to C++. In: IEEE BLOCKS AND BEYOND WORKSHOP (BLOCKS AND BEYOND), 1., 2015, Atlanta. **Proceedings...** Atlanta: Ieee, 2015. p. 91 - 95.

QUEIROZ, Rubens Lacerda; SAMPAIO, Fábio Ferrentini; SANTOS, Mônica Pereira dos. DuinoBlocks4Kids: Ensinando conceitos básicos de programação a crianças do Ensino Fundamental I por meio da Robótica Educacional. In: CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 5., 2016, Uberlândia. **Anais dos Workshops**. Uberlândia: CBIE, 2016. p. 1169 - 1178.

RESNICK, Mitchel. Behavior Construction Kits. **Communications Of The Acm**, Massachusetts, v. 36, n. 7, p.78-95, jul. 1993.

RESNICK, Mitchel et al. Scratch: Programming for all. **Communications Of The Acm**, New York, v. 52, n. 11, p.60-67, 1 nov. 2009.

SCRATCH FOUNDATION. **Scratch**. [S.l.], [2019?]. Disponível em: <https://scratch.mit.edu/>. Acesso em: 18 nov. 2019.

SENTANCE, Sue et al. Creating Cool Stuff: Pupils' Experience of the BBC micro:bit. In: ACM SIGCSE TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 48., 2017, Seattle. **Proceedings of the 2017**. Seattle: Acm, 2017. p. 531 - 536.

VON WANGENHEIM, Aldo et al. Motivating Teachers to Teach Computing in Middle School – A Case Study of a Physical Computing Taster Workshop for K-12 Teachers. **International Journal Of Computer Science Education In Schools,** [s.l.], v. 1, n. 4, p.35-50, 31 out. 2017.

VON WANGENHEIM, Christiane Gresse; NUNES, Vinícius Rodrigues; SANTOS, Giovane Daniel dos. Ensino de Computação com SCRATCH no Ensino Fundamental – Um Estudo de Caso. **Revista Brasileira de Informática na Educação**, [s.l.], v. 22, n. 03, p.115-125, 23 nov. 2014.

ANEXO A – CÓDIGO FONTE COMPLETO GERADO A PARTIR DO PROGRAMA EM BLOCOS

Neste anexo é apresentado, no Quadro 8, o código fonte em C++ gerado a partir do programa exemplo construído na linguagem de programação baseada em blocos da plataforma. A geração foi realizada pelo *frontend* e nessa apresentação estão mantidos os comentários e linhas em branco.

Quadro 8 – Código fonte gerado completo

```
#include <Otto9.h>
Otto9 Otto;  //This is Otto!

//---------------------------------------------------------------------
//-- Make sure the servos are in the right pin
/*               --------
*               |  O  O  |
*               |--------|
*  RIGHT LEG D3 |        | LEFT LEG D2
*                --------
*                ||     ||
*RIGHT FOOT D1 |---     ---| LEFT FOOT D0
*/

#define PIN_LEFTLEG D2
#define PIN_RIGHTLEG D3
#define PIN_LEFTFOOT D0
#define PIN_RIGHTFOOT D1
#define PIN_NOISE_SENSOR 1 //TX - NOT USED
#define PIN_BUZZER 13
#define PIN_USTRIGGER D4
#define PIN_USECHO D5

/*SOUNDS******************
* S_connection  S_disconnection  S_buttonPushed S_mode1 S_mode2 S_mode3 S_surprise
S_OhOoh  S_OhOoh2  S_cuddly
* S_sleeping  S_happy S_superHappy S_happy_short S_sad S_confused S_fart1 S_fart2
S_fart3
*/

/*MOVEMENTS LIST**************
* dir=1---> FORWARD/LEFT
* dir=-1---> BACKWARD/RIGTH
* T : amount of movement. HIGHER VALUE SLOWER MOVEMENT usually 1000 (from 600 to 1400)
* h: height of mov. around 20
*    jump(steps=1, int T = 2000);
*    walk(steps, T, dir);
*    turn(steps, T, dir);
*    bend (steps, T, dir); //usually steps =1, T=2000
*    shakeLeg (steps, T, dir);
*    updown(steps, T, HEIGHT);
*    swing(steps, T, HEIGHT);
*    tiptoeSwing(steps, T, HEIGHT);
*    jitter(steps, T, HEIGHT); (small T)
*    ascendingTurn(steps, T, HEIGHT);
*    moonwalker(steps, T, HEIGHT,dir);
*    crusaito(steps, T, HEIGHT,dir);
*    flapping(steps, T, HEIGHT,dir);
*/

/*GESTURES LIST***************
* OttoHappy OttoSuperHappy  OttoSad   OttoSleeping  OttoFart  OttoConfused OttoLove
OttoAngry
* OttoFretful OttoMagic  OttoWave  OttoVictory  OttoFail
*/

//DO NOT REMOVE!!!
//@@REPLACE_GLOBAL_INIT@@

int MATRX_DIN = D8;
```

```
int MATRX_CS =  D7;
int MATRX_CLK = D6;
int MATRIX_DIRECTION = 2;

//////////////////////////////////////////////////////////////////
//-- Setup ------------------------------------------------------//
//////////////////////////////////////////////////////////////////
void setup() {
      Otto.init(PIN_LEFTLEG,PIN_RIGHTLEG,PIN_LEFTFOOT,PIN_RIGHTFOOT,true,PIN_NOISE_SEN
SOR,PIN_BUZZER,PIN_USTRIGGER,PIN_USECHO); //Set the servo pins
      Otto.home(); //Otto at rest position
      Otto.putMouth(thunder, true);
      Otto.sing(S_connection);
      delay(50);

      //DO NOT REMOVE!!!
      //@@REPLACE_SETUP@@

      Otto.initMATRIX(MATRX_DIN, MATRX_CS, MATRX_CLK, MATRIX_DIRECTION);
      Otto.matrixIntensity(1);

      delay(5000);

}

//////////////////////////////////////////////////////////////////
//-- Principal Loop ---------------------------------------------//
//////////////////////////////////////////////////////////////////
void loop() {
      //DO NOT REMOVE!!!
      //@@REPLACE_LOOP@@

      Otto.walk(2,1000,1); //2 steps, "TIME". IF HIGHER THE VALUE THEN SLOWER (from
600 to 1400), 1 FORWARD

      for (int i = 0; i < 10; i++) {

            if ((Otto.getDistance() < 10)) {
                  Otto.putMouth(sadOpen, true);
                  Otto.sing(S_sad);

            };

      };

      Otto.putMouth(happyOpen, true);
      Otto.sing(S_superHappy);

}

//DO NOT REMOVE!!!
//@@REPLACE_GLOBAL_FUNCTIONS@@
```

Fonte: elaborado pelo autor.

ANEXO B – Alterações realizadas na biblioteca Otto9

Neste anexo é apresentado, no Quadro 9, as alterações realizadas na biblioteca Otto9 para compatibilizar com o microcontrolador Espressif Esp8266, além do já suportado Arduino Nano ATmega329. O código apresentado está no formato *diff*.

Quadro 9 – Alterações realizadas na biblioteca Otto9 no formato *diff*

```
diff --git a/firmware/lib/Oscillator/Oscillator.cpp
b/firmware/lib/Oscillator/Oscillator.cpp
index ca06847..f01c1d0 100644
--- a/firmware/lib/Oscillator/Oscillator.cpp
+++ b/firmware/lib/Oscillator/Oscillator.cpp
@@ -48,8 +48,8 @@ void Oscillator::attach(int pin, bool rev)
       //-- Initialization of oscilaltor parameters
       _TS=30;
       _T=2000;
-      _n = _T/_TS;
-      _inc = 2*M_PI/_n;
+      _N = _T/_TS;
+      _inc = 2*M_PI/_N;

       _previousMillis=0;

@@ -84,8 +84,8 @@ void Oscillator::SetT(unsigned int T)
   _T=T;

   //-- Recalculate the parameters
-  _n = _T/_TS;
-  _inc = 2*M_PI/_n;
+  _N = _T/_TS;
+  _inc = 2*M_PI/_N;
 };

 /*******************************/
diff --git a/firmware/lib/Oscillator/Oscillator.h
b/firmware/lib/Oscillator/Oscillator.h
index 7359f16..8448426 100644
--- a/firmware/lib/Oscillator/Oscillator.h
+++ b/firmware/lib/Oscillator/Oscillator.h
@@ -52,7 +52,7 @@ class Oscillator
     int _trim;        //-- Calibration offset
     double _phase;    //-- Current phase
     double _inc;      //-- Increment of phase
-    double _n;        //-- Number of samples
+    double _N;        //-- Number of samples
     unsigned int _TS; //-- sampling period (ms)

     long _previousMillis;
diff --git a/firmware/lib/Otto9/Otto9.cpp b/firmware/lib/Otto9/Otto9.cpp
index bb8dbbc..99ba6ec 100644
--- a/firmware/lib/Otto9/Otto9.cpp
+++ b/firmware/lib/Otto9/Otto9.cpp
@@ -180,14 +180,11 @@ void Otto9::_execute(int A[4], int O[4], int T, double
phase_diff[4], float step

   //-- Execute complete cycles
   if (cycles >= 1)
-    for(int i = 0; i < cycles; i++) {
+    for(int i = 0; i < cycles; i++)
       oscillateServos(A,O, T, phase_diff);
-      yield();
-    }

   //-- Execute the final not complete cycle
   oscillateServos(A,O, T, phase_diff,(float)steps-cycles);
-  yield();
 }
```

```
@@ -254,7 +251,7 @@ void Otto9::walk(float steps, int T, int dir){
   //--       90 : Walk backward
   //-- Feet servos also have the same offset (for tiptoe a little bit)
   int A[4]= {30, 30, 20, 20};
-  int O[4] = {0, 0, 4, 4};
+  int O[4] = {0, 0, 4, -4};
   double phase_diff[4] = {0, 0, DEG2RAD(dir * -90), DEG2RAD(dir * -90)};

   //-- Let's oscillate the servos!
@@ -277,7 +274,7 @@ void Otto9::turn(float steps, int T, int dir){
   //--   the right leg are bigger than the left. So, the robot describes an
   //--   left arc
   int A[4]= {30, 30, 20, 20};
-  int O[4] = {0, 0, 4, 4};
+  int O[4] = {0, 0, 4, -4};
   double phase_diff[4] = {0, 0, DEG2RAD(-90), DEG2RAD(-90)};

   if (dir == LEFT) {
```

Fonte: elaborado pelo autor.

ANEXO C – LISTA DE COMPONENTES ELETRÔNICOS

Neste anexo são apresentadas as listas de componentes eletrônicos do Otto DIY, no Quadro 10, e do robô desenvolvido na plataforma, no Quadro 11.

Quadro 10 – Lista de componentes eletrônicos do Otto DIY

Quantidade	**Componente eletrônico**	**Preço unitário (R$)**
1	Arduino Nano	25,00
1	Arduno Nano Shield I/O	18,00
1	Cabo USB-A para Mini-USB	10,00
1	Babateria LiPo de 7.4 V 1000 mAh (opcional);	80,00
1	Módulo Bluetooth HC-60 ou similar	25,00
1	Sensor Touch	8,00
4	Motores micro servo MG90	25,00
1	Buzzer	12,00
10	Cabos *jumper* fêmea/fêmea	1,00
1	Micro *switch self lock on/off* 8x8 mm	1,50
1	Sensor de som digital ou analógico	12,00
1	Sensor HC-SR04	10,00
4	Baterias AA (substitui a bateria LiPo recarregável)	3,20

Fonte: elaborado pelo autor.

Quadro 11 – Lista de componentes eletrônicos do robô

Quantidade	**Componente eletrônico**	**Preço unitário (R$)**
1	NodeMCU Esp8266	25,00
1	Base NodeMCU V3 Esp8266 Lolin	20,00
1	Bateria LiPo de 7.4 V 1000 mAh (opcional);	80,00
4	Motores micro servo MG90S	25,00
1	Buzzer	12,00
10	Cabos *jumper* fêmea/fêmea	1,00
1	Micro *stwitch self lock on/off* 8x8 mm	1,50
1	Módulo matriz de LED 8x8 com MAX77219	18,00
1	Sensor HC-SR04	10,00

Fonte: elaborado pelo autor.