**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS**

**DEPARTAMENTO ACADÊMICO DE ELETRÔNICA**

**CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA**

**NYCOLAS COELHO DE ABREU**

**TÍTULO DO TRABALHO: e subtítulo se houver**

**FLORIANÓPOLIS, 2022.**

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS**

**DEPARTAMENTO ACADÊMICO DE ELETRÔNICA**

**CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA**

**NYCOLAS COELHO DE ABREU**

**TÍTULO DO TRABALHO: e subtítulo se houver**

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro em Eletrônica.

Orientador:   
Prof. Renan Augusto Starke, Dr.

**FLORIANÓPOLIS, 2022.**

**PÁGINA PARA COLOCAÇÃO**

**DA FICHA DE IDENTIFICAÇÃO DA OBRA**

(A Ficha de identificação da obra deve ser elaborada de acordo com o padrão adotado pela biblioteca do IFSC a partir do formulário disponível em: <http://ficha.florianopolis.ifsc.edu.br/>

Observação: por questões de compatibilidade do *site*, recomenda-se que a Ficha de identificação seja gerada no navegador ***Mozilla Firefox***)

**TÍTULO DO TRABALHO**

**NYCOLAS COELHO DE ABREU**

Este trabalho foi julgado adequado para obtenção do título de Engenheiro em Eletrônica e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, \_\_\_ de \_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_.

Banca Examinadora:

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Nome do Orientador, Titulação

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Nome do Coorientador (se houver), Titulação

Instituição/Empresa

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Nome do Membro da Banca, Titulação

Instituição/Empresa

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Nome do Membro da Banca, Titulação

Instituição/Empresa

**RESUMO**

O resumo deve mostrar a natureza e o objetivo do trabalho, o método que foi empregado, os resultados e as conclusões. O resumo deve conter entre 150 e 500 palavras e constitui-se de um único parágrafo, sem recuo.

**Palavras-chave**: Primeira palavra-chave. Segunda palavra-chave. Terceira palavra-chave. Quarta palavra-chave (opcional). Quinta palavra-chave (opcional).

**ABSTRACT**

The abstract should show the nature and scope of work, the method that was used, the results and conclusions. The abstract may contain between 150 and 500 words, and it must be only one paragraph.

**Keywords**: First keyword. Second keyword. Third keyword. Fourth keyword (optional). Fifth keyword (optional).

**LISTA DE FIGURAS**

[Figura](#_heading=h.1ksv4uv) 1 – Sintaxe, semântica e pragmática 17

[Figura 2 – Código compilado e interpretado](#_heading=h.1ksv4uv) 17

Figura 3 – Exemplo de código escrito em C embarcado 17

Figura 4 – Caracteres de linguagens de programação 17

Figura 5 – Ambiente de programação em blocos Scratch 17

Figura 6 – Diferença de performance de alunos em linguagem  
 textual e em blocos 17

Figura 7 – Implementação da biblioteca Blockly 17

Figura 8 – Interface do MicroBlocks 17

Figura 9 – Tradução do código em BIPES para MicroPython 17

Figura 10 – Tipos de conexões do BIPES 17

Figura 11 – Buggy AWD 17

Figura 12 – Shield para MSP-EXP430G2 17

Figura 13 – Circuito ponte H utilizado 17

**LISTA DE QUADROS**

Quadro 1 – Exemplos de símbolos16

Quadro 2 – Placas suportadas pelo MicroBlocks16

**LISTA DE TABELAS**

[Tabela 1 – Produção de petróleo na Bahia 17](#_heading=h.44sinio)

**LISTA DE ABREVIATURAS E SIGLAS**

| ADC | Analog-to-Digital Converter |
| --- | --- |
| ANSI | American National Standards Institute |
| API | Application Programming Interface |
| AWD | CARRINHO |
| BIPES | Block based Integrated Platform for Embedded Systems |
| HTTP | Hypertext Transfer Protocol |
| I²C | Inter-Integrated Circuit |
| I/O | Input / Output |
| IFRN | Instituto Federal do Rio grande do Norte |
| IFSC | Instituto Federal de Santa Catarina |
| IoT | Internet of Things |
| LED | Light-Emitting Diode |
| LiPo | Lithium Polymer Battery |
| LPAE | Laboratório de Pesquisa Avançada em Eletrônica |
| MIT | Massachusetts Institute of Technology |
| MQTT | Message Queuing Telemetry Transport |
| PWM | Pulse Width Modulation |
| RAM | Random Access Memory |
| SPI | Serial Peripheral Interface |
| STL | Standard Template Library |
| TCP/IP | Transmission Control Protocol / Internet Protocol |
| UART | Universal Asynchronous Receiver-Transmitter |
| UFRN | Universidade Federal do Rio Grande do Norte |
| UFSCar | Universidade Federal de São Carlos |
| USB | Universal Serial Bus |
| USP | Universidade de São Paulo |
| WDT | Watchdog Timer |
| WiFi | Wireless Fidelity |

**LISTA DE SÍMBOLOS**

| Kb | Quilobyte |
| --- | --- |
| pA | Pico Ampere |
| V | Volts |

**SUMÁRIO**

[**1 INTRODUÇÃO**](#_heading=h.im2yqln38dw7) **13**

[**1.1 Justificativa**](#_heading=h.3znysh7) **14**

[**1.2 Definição do Problema**](#_heading=h.2et92p0) **15**

[**1.3 Objetivo Geral**](#_heading=h.tyjcwt) **15**

[**1.4 Objetivos Específicos**](#_heading=h.3dy6vkm) **15**

[**1.5 Estrutura do Trabalho**](#_heading=h.1t3h5sf) **16**

[**2 FUNDAMENTAÇÃO TEÓRICA**](#_heading=h.4d34og8) **16**

[**2.1 Programação textual**](#_heading=h.17dp8vu) **16**

[2.1.1 Linguagem C](#_heading=h.vkbiiogq3c9v) 18

[2.1.2 Linguagem C++](#_heading=h.opigied4p5mn) 19

[2.1.3 MicroPython](#_heading=h.psoyhpqfn1le) 20

[2.1.4 Sintaxe de linguagens textuais](#_heading=h.jehsj1buftmq) 21

[**2.2 Programação em blocos**](#_heading=h.nsjsemhy72de) **23**

[**2.3 Ensino de programação em blocos**](#_heading=h.ao15g8ekdkvu) **24**

[**2.4 Scratch**](#_heading=h.3qhm5qfooqa5) **26**

[**2.5 Blockly**](#_heading=h.7ldqb8nsiar2) **27**

[**2.6 MicroBlocks**](#_heading=h.bym36fs4sdpl) **28**

[**2.7 BIPES**](#_heading=h.fn5l21o432fb) **30**

[**2.8 Buggy AWD**](#_heading=h.3g0qhdqkm37c) **33**

[2.8.1 Cérebro](#_heading=h.frw3ia68x39x) 33

[2.8.2 Sensores](#_heading=h.vmjm7jaoxive) 34

[2.8.2.1 Encoder](#_heading=h.y0zi8cop6ppv) 34

[2.8.2.2 Monitoramento de baterias](#_heading=h.css9wpwyl1w2) 35

[2.8.2.3 Sensores seguidores de linha](#_heading=h.yoe05bgdjarn) 35

[2.8.2.4 Sensor de distância ultrassônico](#_heading=h.r117tbylnixn) 35

[2.8.3 Motores](#_heading=h.36u74ot2f897) 36

[2.9 Resumo](#_heading=h.7ctc7mxazpet) 37

[**3 METODOLOGIA**](#_heading=h.1y810tw) **37**

[**3.1 Métodos aplicados**](#_heading=h.4i7ojhp) **37**

[**4 APRESENTAÇÃO DOS RESULTADOS**](#_heading=h.2xcytpi) **38**

[**4.1 Análise e discussão dos resultados**](#_heading=h.1ci93xb) **38**

[**5 CONSIDERAÇÕES FINAIS**](#_heading=h.3whwml4) **39**

[**5.1 Sugestões para trabalhos futuros**](#_heading=h.2bn6wsx) **39**

# INTRODUÇÃO

O projeto de sistemas embarcados cresce de forma constante nos últimos anos. As aplicações são onipresentes na fabricação de produtos de consumo, veículos, telecomunicações, sistemas militares e de defesa do meio ambiente (MICHELI; SAMI, 1996).

Esses componentes estão se tornando mais complexos e possuem cada vez menos tempo para serem apresentados ao mercado. Portanto, surgiu a necessidade de criar um ambiente de projeto mais eficiente e de fácil utilização, onde os projetistas e jovens sem muita experiência possam utilizar da programação visual ou *no-code* para auxiliar em seus estudos e protótipos.

Atualmente, a base da programação de microcontroladores são linguagens textuais como C, C++, Assembly e MicroPython, sendo todas linguagens com sintaxes rigorosas. Como esclarece Caballar (2020), a programação *no-code* seria a progressão natural do desenvolvimento de software, envolvendo a adição de camadas de abstração e escondendo as complexidades por trás do código textual, objetivando uma programação mais fácil para os desenvolvedores, principalmente para iniciantes.

Ferramentas simples e intuitivas de programação em blocos já são amplamente utilizadas por desenvolvedores e estudantes, plataformas de programação em blocos como o Scratch, que foi projetado para ensinar programação para jovens, possui mais de 95 milhões de usuários e quase 115 milhões de projetos compartilhados (MIT; 2022) e também a Blockly,uma biblioteca criada pelo Google para facilitar a criação de linguagens visuais em bloco, utilizada em centenas de projetos, principalmente educacionais (GOOGLE; 2022).

Este trabalho tem como proposta o desenvolvimento de uma aplicação utilizando plataformas de programação em blocos projetadas para microcontroladores, utilizando como base o projeto de carro autônomo AWD disponibilizado pelo LPAE.

A elaboração da aplicação foi realizada pela plataforma *open* *source* BIPES, desenvolvida no Brasil e planejada para programação de sistemas embarcados e aplicações IoT (JUNIOR et al., 2020), e também na plataforma MicroBlocks um ambiente de programação projetado para kits de eletrônica, inspirado em Scratch, GP e Snap! (CABRERA; MALONEY; WEINTROP, 2019).

## Justificativa

A programação de microcontroladores é um processo que necessita de desenvolvedores e engenheiros qualificados com amplo conhecimento de uma determinada plataforma e linguagem de programação, além de requerer uma pesquisa minuciosa de diferentes documentos e *datasheets,* para que se consiga obter os conhecimentos necessários para a execução de um projeto. Como resultado, as bases de códigos geralmente carecem de modularidade e apresentam uma estrutura de difícil compreensão.

Para jovens e iniciantes, analisar e criar bases de código para microcontroladores torna-se uma tarefa difícil e demorada. Por esses motivos, novas ferramentas e metodologias de programação foram criadas para facilitar e acelerar o processo de aprendizado e prototipagem.

Um destes métodos é a programação em blocos, que introduz uma dimensão visual para apoiar o processo de aprendizagem. Segundo Weintrop (2019), este ambiente produz uma nova maneira de se escrever código, incluindo cores e formas, podendo reduzir a curva de aprendizado que os alunos possuem quando iniciam o aprendizado de programação em um ambiente baseado em texto.

A ferramenta de programação em blocos, além de facilitar a codificação para o usuário, prepara o ambiente de programação de forma automatizada, colocando em prática a ideia *plug-and-play*, o usuário não precisa se preocupar com diversas variáveis ​​de preparação, instalação e configuração da plataforma.

Diante disso, esse estudo propõe abordar a programação em blocos voltada para microcontroladores, implementando uma aplicação utilizando plataformas existentes e atuais de programação em blocos, a fim de evidenciar sua praticidade, viabilidade e funcionamento.

## Definição do Problema

Dependendo da plataforma e da linguagem escolhida, a programação de sistemas embarcados pode ser complexa e demorada, envolvendo a compreensão de um conjunto de teoria, prática, conhecimento sobre paradigmas, sintaxe, semântica da linguagem, lógica e abstração. Esses aspectos dificultam o aprendizado para iniciantes e são uma barreira para aqueles que desejam criar protótipos rapidamente.

Dessa forma, diante das antigas e atuais opções de linguagens e ferramentas de programação para sistemas embarcados, podemos utilizar alguma plataforma para criar protótipos simples e com um aprendizado facilitado?

## Objetivo Geral

O presente trabalho tem como objetivo o estudo de ferramentas de programação alternativas e acessíveis, visando a facilidade na prototipação e uma alternativa de aprendizado moderna para programação de microcontroladores. Aplicando as descobertas do estudo no desenvolvimento de uma aplicação utilizando programação em blocos para o projeto do veículo autônomo AWD (github.com/xtarke/automated\_buggy), presente no LPAE.

## Objetivos Específicos

Com o objetivo geral apresentado, destaca-se os seguintes objetivos específicos:

1. Estudar as ferramentas e tecnologias disponíveis para o desenvolvimento da aplicação;
2. Escolha dos componentes para construção do *hardware*;
3. Projetar a interface de comunicação com o microcontrolador;
4. Implementar a aplicação;
5. Testar o funcionamento e a viabilidade;
6. Analisar os resultados alcançados;

## Estrutura do Trabalho

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto.

# FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os principais temas relacionados a este trabalho. Primeiro, é apresentado um embasamento teórico sobre programação textual e em blocos, com ênfase em suas estruturas e características. Posteriormente, são abordadas as linguagens de programação em blocos utilizadas para este trabalho, apresentando suas funcionalidades e particularidades.

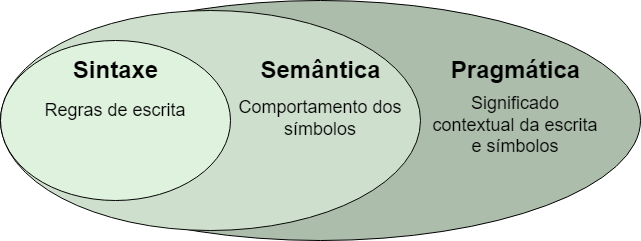
## Programação textual

De acordo com Sammet (1969), uma linguagem de programação é um conjunto de caracteres que possuem regras para combiná-los e uma especificação de seus efeitos quando executados em um computador. O nível de abstração deve ser tal que os usuários não precisem conhecer código de máquina e para que a linguagem seja amplamente independente, permitindo que o mesmo programa seja executado em computadores com arquiteturas diferentes.

Slonneger e Kurtz (1995) definem as linguagens de programação textual pelas suas três características em comum:

* Sintaxe: Refere-se às formas como os símbolos podem ser combinados para criar comandos na linguagem. É uma descrição estrutural das expressões que compõem a linguagem. A sintaxe lida apenas com a forma e a estrutura dos símbolos, sem qualquer ligação com seus significados.
* Semântica: Revela o significado dos comandos em uma linguagem. Correlacionando seus símbolos com os comportamentos que devem ser executados.
* Pragmática: Trata-se dos aspectos da linguagem que envolvem os usuários, ou seja, fenômenos psicológicos e sociológicos como utilidade, escopo de aplicação e seus efeitos, incluindo questões como facilidade de implementação, eficiência na aplicação e metodologia de programação.

**Figura 1 – Sintaxe, semântica e pragmática**

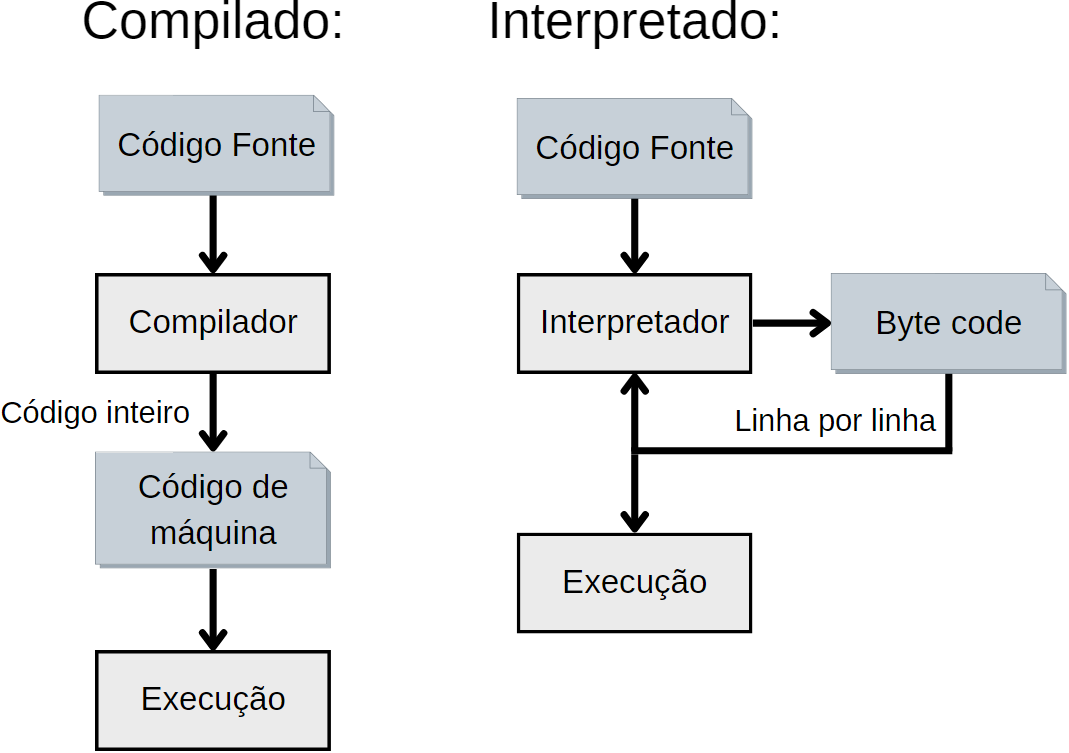


Fonte: Elaboração própria (2022).

As linguagens de programação podem ser divididas em duas categorias, sendo elas linguagens compiladas e linguagens interpretadas.

Em uma linguagem compilada, um compilador pré-traduz o programa em código de máquina para uma plataforma específica, podendo assim ser carregado na memória do computador e executado diretamente por ele (WATSON, 2017). Nas linguagens interpretadas não ocorre esta antecipação, o código é executado diretamente na plataforma pelo interpretador, traduzindo o programa em código de máquina linha por linha simultaneamente com a execução do programa (KWAME; MARTEY; CHRIS, 2017).

**Figura 2 – Código compilado e interpretado**



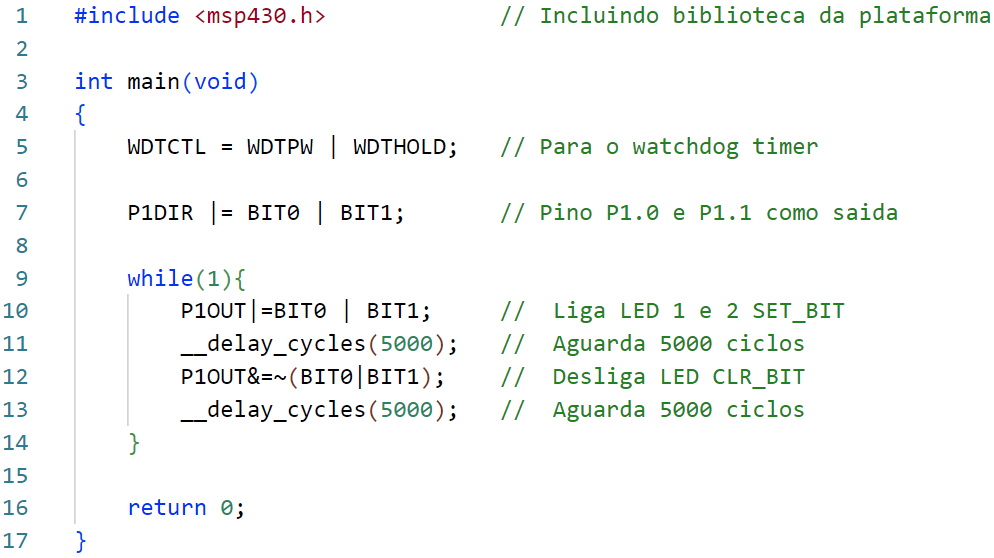
Fonte: Elaboração própria (2022).

Os microcontroladores são tipicamente programados em linguagens compiladas como C e C++, como mostrado atualmente por Mazzei et al. (2015), devido ao desenvolvimento da tecnologia e ao aumento da capacidade de memória dos microcontroladores, é possível implementar máquinas virtuais em ambientes embarcados assim podendo interpretar linguagens modernas como Java, Python e MicroPython.

### Linguagem C

Como o criador da linguagem Ritchie (1993) afirma, C surgiu entre os anos de 1969 e 1973 como uma linguagem para ser utilizada no sistema operacional Unix, mas apenas em meados de 1980 a linguagem foi padronizada pelo comitê ANSI, fazendo seu uso se espalhar e colocando-a entre as linguagens mais usadas em toda a indústria de computadores.

C é uma linguagem de programação compilada, de uso geral, com paradigmas estruturais, imperativos e procedurais, possui uma sintaxe reduzida, controle de fluxo moderno, estruturas de dados e um rico conjunto de operadores. Sua ausência de limitações e sua generalidade a tornam mais conveniente e eficiente para muitas tarefas do que linguagens supostamente mais poderosas (KERNIGHAN; RITCHIE, 1988).

**Figura 2 – Exemplo de código escrito em C embarcado**

Fonte: Elaboração própria (2022).

Os microcontroladores são tradicionalmente programados em linguagem assembly, como Pardue (2015) afirma, cada plataforma possui sua própria sintaxe, dificultando o aprendizado e tornando o código obsoleto no momento em que se muda de família. Alunos, professores e hobistas tendem a pular o estudo detalhado de assembler, seguindo direto para o C, que é mais simples e eficiente. Devido a quantidade crescente de compiladores e sua facilidade de implementação, C se tornou a linguagem padrão para programação em microcontroladores (BATES, 2008).

Todos os microcontroladores possuem periféricos e outros componentes que devem ser acessados e controlados ​​pelo programador. A linguagem C permite configurar e inicializar os bits sinalizadores nos registradores de controle para cada operação, além de facilitar o acesso aos endereços e a memória interna do dispositivo, dando ao programador total controle da plataforma que está sendo utilizada.

### Linguagem C++

Como Stroustrup (2013) o criador da linguagem explica, C++ foi desenvolvida em 1983, projetada para tornar a programação mais eficiente e moderna. C++ é uma extensão da linguagem de programação C, dessa forma ela possui retrocompatibilidade com os códigos já existentes escritos em C.

C++ é uma linguagem de programação compilada, de propósito geral, assim como C, mas com paradigmas modernos, incluindo abstração de dados, orientação a objetos e programação genérica. Ao longo do tempo, a linguagem passou por diversas atualizações, sendo a principal adição uma nova biblioteca padrão chamada STL em 1998, que forneceu ferramentas úteis como contêineres, algoritmos e iteradores de contêineres (STROUSTRUP, 2013).

Entre as maiores diferenças com C inclui-se a programação orientada a objetos, que reduz o tempo de desenvolvimento, devido a facilidade de reutilização de código, também inclui-se a alocação dinâmica de memória utilizando ponteiros inteligentes, que diminui drasticamente a chance de falhas de segurança, estes pontos fazem a linguagem C++ ser mais popular que C em projetos de grande porte (BARR, 1999).

C++ foi projetado para programação de sistemas e software embarcado, focado em grandes sistemas e com recursos limitados, com desempenho, eficiência e flexibilidade de uso como destaques do design (STROUSTRUP, 2014). Portanto, devido a todos os fatores, a utilização de C++ em microcontroladores se torna uma experiência mais moderna e completa quando se comparada com a linguagem C.

### MicroPython

“Python é uma linguagem de script de alto nível, interpretada e orientada a objetos. Sua principal característica é a sua sintaxe clara e simples, é próximo do inglês, sendo possível ler o código e entendê-lo sem ter aprendido Python” (BEEL, 2017, p.19). A linguagem é utilizada no desenvolvimento de sites, aplicativos gráficos, análise de dados, aprendizado de máquina e IoT. Tornou-se popular devido a sua simplicidade e pelo seu conjunto de bibliotecas e API’s desenvolvidas pela comunidade.

O MicroPython surgiu em 2013, como uma implementação do Python 3, desenvolvido por Damien George com foco em microcontroladores e sistemas embarcados, podendo executar códigos escritos em Python diretamente no hardware (TOLLERVEY, 2017). Ele apresenta um subconjunto da biblioteca padrão do Python, é otimizado para ambientes embarcados e compacto o suficiente para ser executado em apenas 256 kb de *flash* e 16 kb de RAM (GEORGE, 2014).

**Figura 3 – PyBoard, microcontrolador oficial do MicroPython**

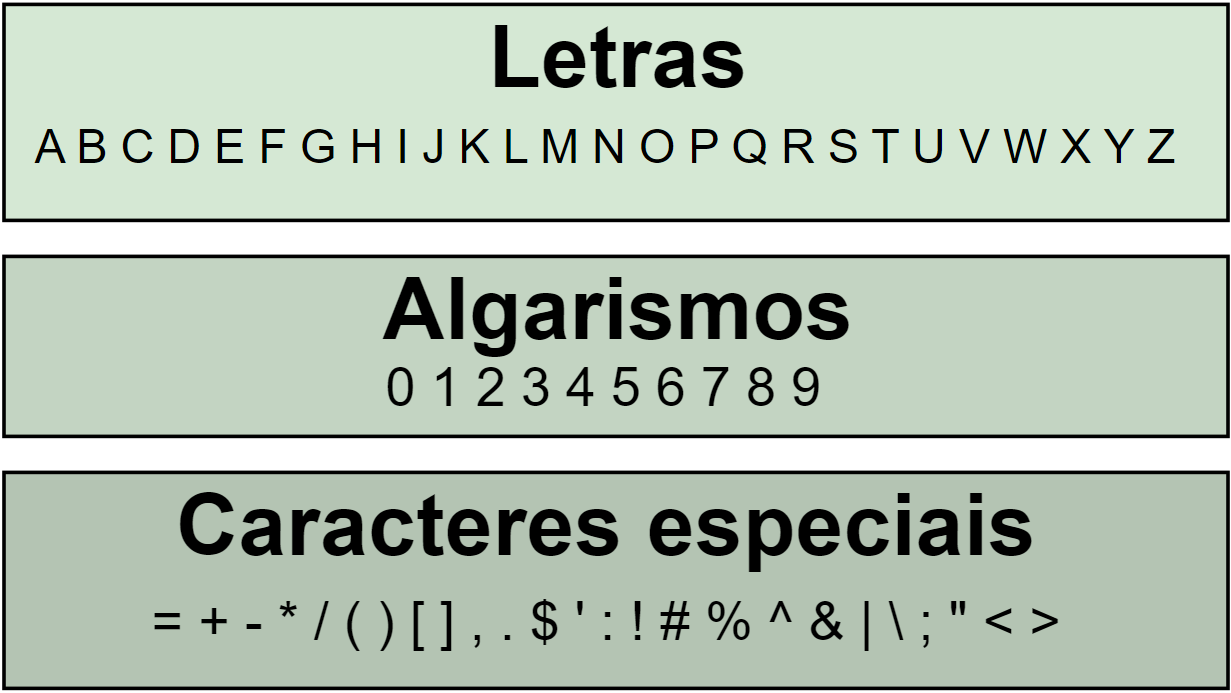
Fonte: George (2014).

A linguagem possui diversos recursos para facilitar o desenvolvimento em projetos embarcados, incluindo bibliotecas de I/O, ADC, PWM, UART, SPI, I²C, rede e Bluetooth. “Em geral, pode-se esperar que o MicroPython suporte cerca de 80% das bibliotecas mais utilizadas em Python” (BEEL, 2017, p.30).

### Sintaxe de linguagens textuais

Um dos problemas das linguagens de programação é a diversidade de pessoas que precisam entender o seu funcionamento. Os usuários devem ser capazes de determinar como escrever soluções de software consultando o manual de referência, livros e outros documentos da linguagem. A sintaxe de uma linguagem de programação define as regras de escrita, delimitando a utilização dos símbolos e facilitando a detecção de erros (SEBESTA, 2018).

Segundo Horowitz (1984), toda linguagem de programação é baseada em um alfabeto de caracteres. Em inglês e português, o alfabeto possui 26 letras e mais 10 algarismos. Os conjuntos de caracteres para linguagens de programação geralmente começam com estes 36 caracteres mais a adição de caracteres especiais.

**Figura 4 – Caracteres de linguagens de programação**

Fonte: Adaptado de Horowitz (1984).

Certos caracteres ou suas combinações que possuem significados bem definidos em um determinado contexto são chamados de símbolos, geralmente seus nomes ou lexemas, são reservados para que os desenvolvedores não os utilizem em outras situações. Símbolos reservados ajudam a tornar o código mais legível e padronizado, mas à medida que o número de símbolos aumenta, torna-se mais difícil memorizá-los (HOROWITZ, 1984).

**Quadro 1 – Exemplos de símbolos**

| **Símbolos** | **Lexemas** |
| --- | --- |
| Palavras-chave | if, while, for, int, float |
| Separadores | { } ( ) ; : [ ] |
| Operadores | / \* - + = < > |
| Identificadores | x, nota, pi, D2 |
| Literais | true, "musica", 100 |

Fonte: Adaptado de Aho et al. (2007).

Como explica Aho et al. (2007), espera-se que os compiladores e interpretadores ajudem o programador a localizar e rastrear estes erros de sintaxe presentes nos programas:

* Erros lexicais: Incluem-se erros ortográficos de identificadores, palavras-chave ou operadores.
* Erros sintáticos: Ponto-e-vírgula mal posicionado ou chaves extras ou ausentes.
* Erros semânticos: Incompatibilidades de tipo entre operadores e operandos.
* Erros lógicos: Utilização de operadores de maneira indevida.

Linguagens de programação como C e C++ possuem uma sintaxe mais rigorosa, com a utilização de separadores para finalizar linhas e delimitar escopos, além do uso de palavras-chave para declaração de variáveis (AHO et al., 2007). Python por ser uma linguagem de alto nível, delimita seus escopos e linhas apenas com identação, não necessitando de palavras-chave para declarar previamente uma variável.

## Programação em blocos

Como aponta Bau et al. (2017), as interfaces de programação baseadas em blocos utilizam metáforas de programação simples, como peças de um quebra-cabeça, para fornecer aos usuários dicas visuais sobre como e onde usar comandos para restringir a composição do programa. Permitindo que jovens e alunos concentrem-se na compreensão dos conceitos básicos de programação e na estrutura geral de um programa (MOORS; LUXTON-REILLY; DENNY, 2018).

A programação nesses ambientes assume a forma de arrastar blocos para a área de *script* e os agrupar para formar um comando. Se dois blocos não puderem ser unidos para formar uma instrução de sintaxe válida, o ambiente impede que eles sejam agrupados, evitando assim erros de sintaxe (MOORS; LUXTON-REILLY; DENNY, 2018).

A eliminação de erros de sintaxe torna-se uma das principais características dos ambientes de programação em blocos, pois iniciantes são capazes de se concentrar no fluxo de controle, condicionais e loops, sem a frustração de garantir que um programa seja sintaticamente correto (WEINTROP, 2019).

Ao contrário das linguagens textuais, as linguagens de bloco estão profundamente relacionadas ao seu ambiente de programação, ferramentas como *Scratch* e *Blockly* empregam algumas convenções de interface, como lidar com a organização, separando os blocos de comandos em abas relacionadas por categoria e adicionando cores diferentes para cada tipo de instrução (BAU et al., 2017), pode-se ver a combinação destes elementos na Figura 5.

**Figura 5 – Ambiente de programação em blocos Scratch**

Fonte: Weintrop (2019).

De acordo com Weintrop e Wilensky (2015), a programação baseada em blocos é conceituada por alguns como uma introdução à programação facilitando uma eventual transição para a programação baseada em texto, sendo uma questão empírica aberta, que está apenas começando a ser respondida.

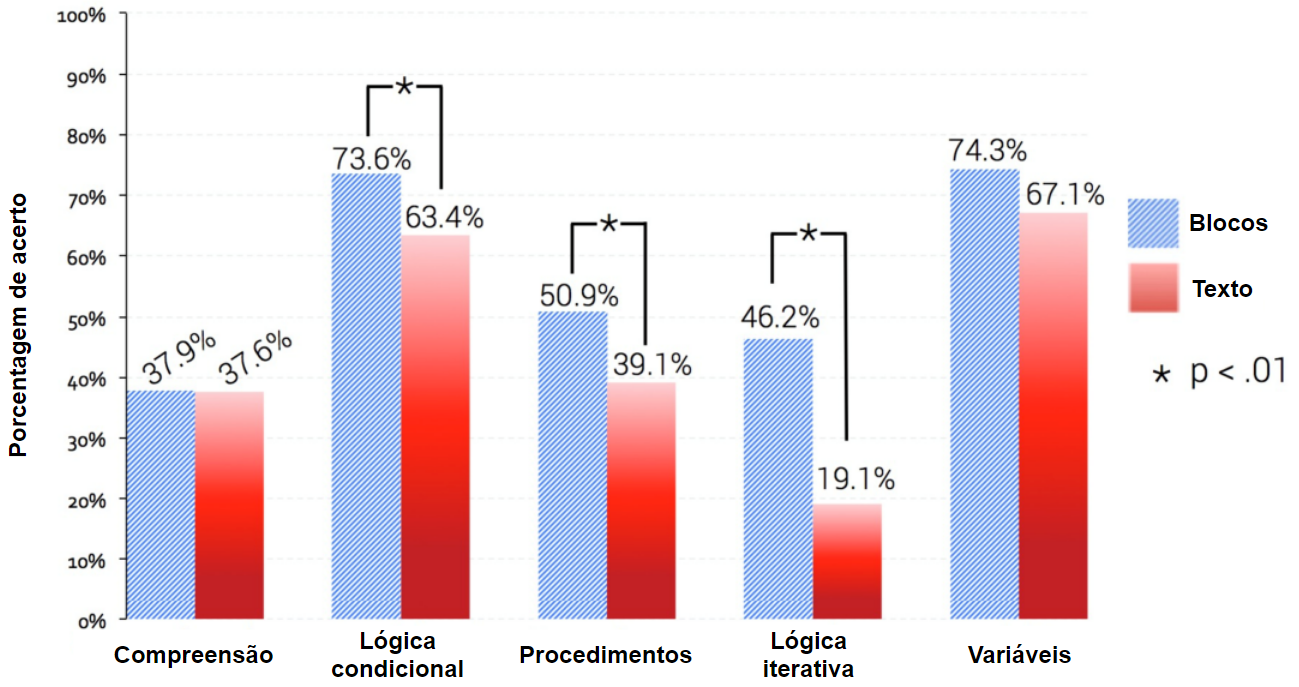
## Ensino de programação em blocos

Algumas das vantagens da programação em blocos podem ocasionar problemas ao escalar o desenvolvimento para grandes projetos e equipes. Segundo Bau et al. (2017) os blocos ocupam mais espaço na tela, dificultando a navegação e a edição de seus comandos, além disso, pequenas edições em comandos existentes são mais difíceis, pois seus blocos precisam ser selecionados e excluídos individualmente.

Apesar de seus problemas, existe uma questão maior sobre o papel das ferramentas baseadas em blocos no campo da ciência da computação. Estas ferramentas servem como uma introdução para o ambiente de programação, em suma, para que uma ferramenta de programação possa ser usada por programadores novos ou casuais, seu design deve focar na capacidade de aprendizado, com a expectativa de que os alunos migrem para linguagens de programação textuais no futuro (WEINTROP, 2019).

Estudos comprovam que as linguagens de programação em blocos facilitam o aprendizado em diversos elementos da área de programação. Em uma pesquisa comutativa feita por Weintrop e Wilensky (2015), é demonstrado um estudo de 10 semanas com 90 alunos, entre a oitava série e terceiro ano, em um curso introdutório de programação. Os alunos foram introduzidos durante 5 semanas a uma linguagem de programação semelhante a *Scratch*, com atividades introdutórias e intermediárias. Após o término das 5 semanas foi aplicada uma prova de 28 questões, dividida entre os temas que foram apresentados, seguido pela introdução da linguagem textual *Java*, por meio de atividades similares às do período anterior e seguido por uma prova de 28 questões, finalizando as 10 semanas.

A Figura 6 apresenta a porcentagem de acerto em média para cada categoria avaliada e sua respectiva linguagem utilizada.

**Figura 6 – Diferença de performance de alunos em linguagem textual e em blocos**

Fonte: Adaptado de Weintrop e Wilensky (2015).

Weintrop e Wilensky (2015) concluem que para todas as categorias apresentadas, houve uma melhoria no desempenho dos alunos quando foi utilizada a linguagem baseada em blocos comparando com a linguagem textual, como pode ser visto pelas barras de porcentagem na Figura 6, comprovando a ideia de que a linguagem de programação em blocos influencia positivamente no aprendizado de conceitos de programação para jovens.

## Scratch

Scratch é uma linguagem de programação visual desenvolvida desde 2003 pelo MIT, financiado pela *National Science Foundation*, Intel *Foundation*, Nokia e pelo MIT *Media Lab Research Consortium*, coordenado por Mitchel Resnick e inspirado nas linguagens Logo e Squeak. “É a maior comunidade de programação do mundo para crianças e uma linguagem de programação com uma interface visual simples que permite que os jovens criem histórias, jogos e animações digitais” (MIT, 2022).

Foi apresentado ao mundo pela primeira vez em 15 de maio de 2007, atualmente é usado em mais de 150 países e disponível em mais de 40 idiomas. Foi concebido para ajudar jovens a partir dos 8 anos. A ferramenta promove o pensamento computacional e desenvolve habilidades de resolução de problemas, auto-expressão, colaboração, lógica e programação (MIT, 2022).

O ambiente de desenvolvimento é dividido em três áreas, conforme mostrado na Figura 5. A primeira área contém os blocos de comando que serão adicionados ao programa, existem comandos relacionados ao movimento, fala, eventos e controle da área de animação. A segunda área é o desenvolvimento do programa, onde é possível adicionar, editar e remover os blocos de comando. A terceira área é a tela de animação, chamada de palco, onde o programa é executado e animado.

Um dos principais recursos do Scratch é seu feedback instantâneo, sem etapas de compilação ou distinções de modo de edição/execução. Os usuários podem clicar em um comando a qualquer momento para testar seu funcionamento. É possível alterar parâmetros ou adicionar blocos ao código enquanto o programa está em execução. Ao eliminar opções de modo potencialmente irritantes e pausas de construção, o Scratch ajuda os usuários a permanecerem envolvidos em testes, depuração e aprimoramento de seus projetos (MALONEY et al., 2010).

## Blockly

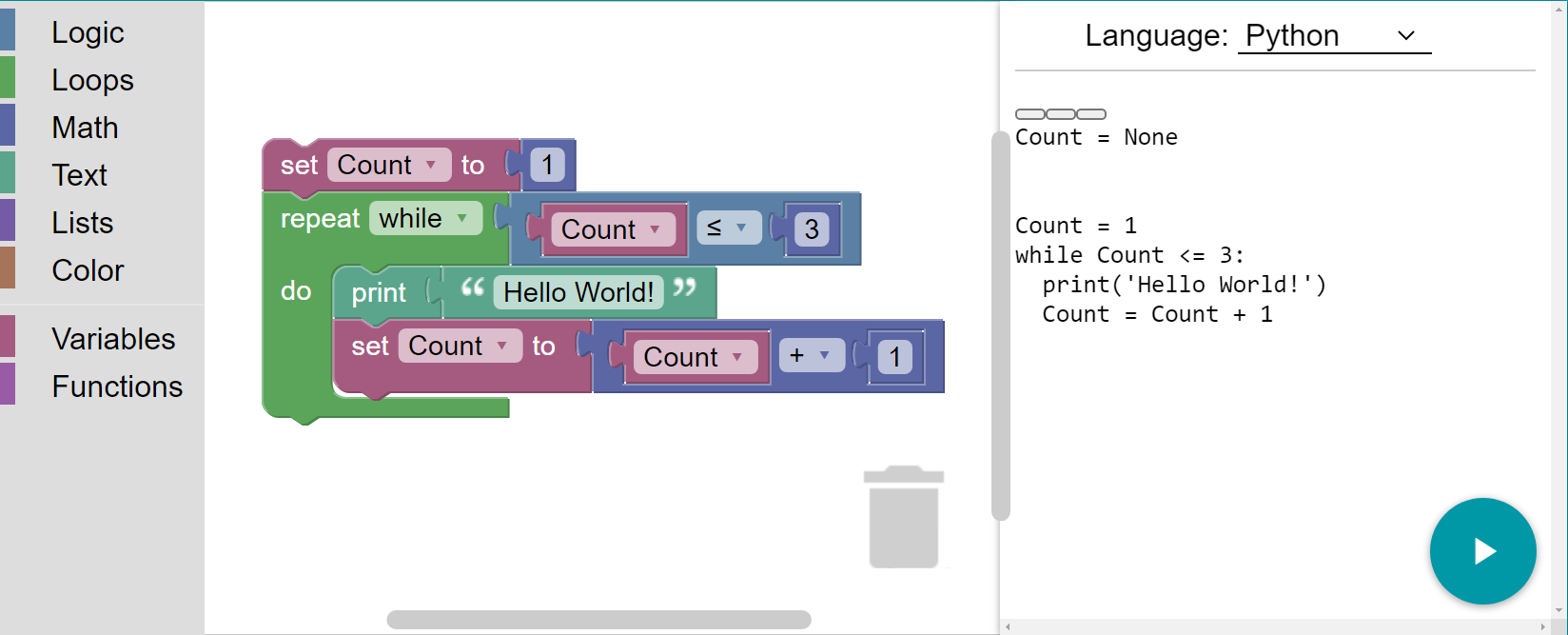
Blockly é uma biblioteca desenvolvida pelo Google em 2012 em colaboração com uma equipe de desenvolvimento do MIT. É utilizada para construir editores de programação em blocos para os mais diversos projetos. A biblioteca é gratuita e de código aberto, desenvolvida em Javascript, por ser uma ferramenta *web*, não requer instalação de nenhum recurso além de um navegador para funcionar corretamente (GOOGLE, 2022).

Blockly não é uma linguagem de programação nem uma plataforma para usuários finais, constitui-se de uma biblioteca que deve ser adicionada ao desenvolvimento de um projeto para facilitar a implementação de linguagens de programação em blocos. Fornece os recursos necessários para estruturar uma linguagem de programação em blocos, como uma interface gráfica para o editor de blocos, lógicas de conexões entre os blocos e um gerador de código para a linguagem textual.

Sua principal característica é a capacidade de exportar nativamente suas linguagens de bloco para linguagens de programação, incluindo JavaScript, Python, PHP, Lua e Dart. Caso seja necessária a tradução para outras linguagens, a biblioteca Blockly fornece todas as ferramentas para realizar esta tarefa. O processo de tradução da linguagem em blocos facilita o aprendizado do usuário e a adaptação às linguagens de programação textual (GOOGLE, 2022).

A Figura 7 mostra um exemplo da implementação do Google da biblioteca Blockly. À esquerda, pode-se observar os blocos separados por categorias, blocos de lógica, loop, operações matemáticas, texto, listas, cores, variáveis ​​e funções. No centro da imagem está a área de edição dos blocos. À direita, o código em blocos é transformado em linguagem de programação textual e pode ser executado apertando o botão de *play*.

**Figura 7 – Implementação da biblioteca Blockly**



Fonte: Google (2022).

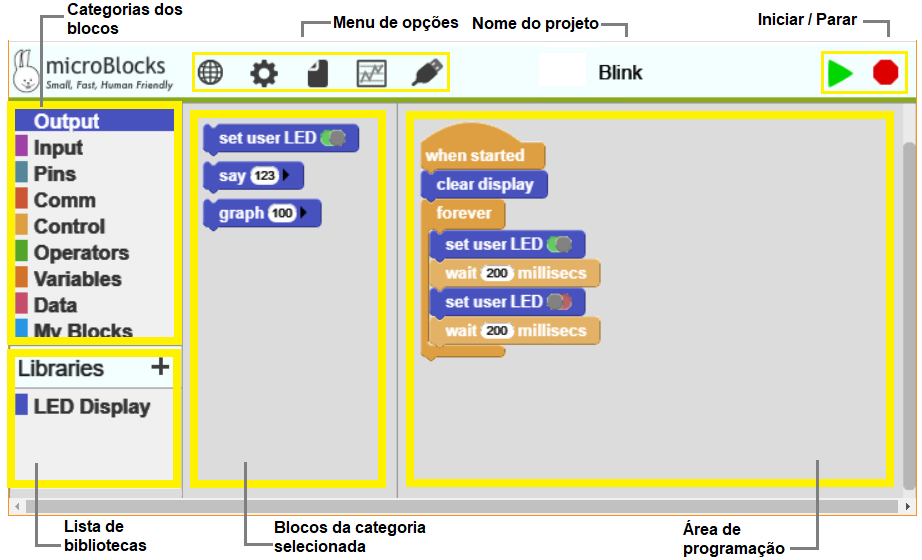
## MicroBlocks

O MicroBlocks é um ambiente de linguagem de programação visual semelhante ao Scratch, focado no ensino de programação para microcontroladores educacionais, como o micro:bit, Adafruit Circuit Playground Express e muitos outros. Fundado por John Maloney e lançado em 2020, sendo uma extensão da linguagem GP também desenvolvida por Maloney (MALONEY; MÖNIG ;ROMAGOSA, 2021).

O que diferencia o MicroBlocks de outras ferramentas de programação de blocos é a combinação de programação com *feedback* imediato e uma execução de forma autônoma. Outras linguagens de bloco suportam um desses recursos, mas não ambos (MALONEY; MÖNIG ;ROMAGOSA, 2021).

Esta linguagem pode ser separada entre seu editor de blocos e sua máquina virtual. O editor permite ao usuário criar, editar e gerenciar códigos baseado em blocos e suas bibliotecas, é organizado de forma simples, separando os blocos em categorias e cores, assim como o Scratch. Inclui bibliotecas de sensores, redes, displays, servomotores, áudio e de seus kits educacionais suportados.

**Figura 8 – Interface do MicroBlocks**



Fonte: Adaptado de Maloney, Mönig e Romagosa (2021).

O código em MicroBlocks é executado por um interpretador próprio. Sua máquina virtual foi projetada para microcontroladores de 32 bits com pelo menos 16 Kb de RAM e 64 Kb de memória *Flash*, é suportada pelas seguintes placas:

**Quadro 2 – Placas suportadas pelo MicroBlocks**

| **Placas suportadas** | |
| --- | --- |
| BBC micro:bit | Citilab's ED1 (ESP32) |
| Calliope mini | Arduino Due (Parcialmente) |
| AdaFruit Express | Arduino Zero (Parcialmente) |
| AdaFruit Bluefruit | M5Atom Matrix |
| AdaFruit Clue | NodeMCU |
| ESP32 | Raspberry Pi Pico (RP2040) |

Fonte: Adaptado de Maloney, Mönig e Romagosa (2021).

O código em MicroBlocks é executado diretamente no microcontrolador, devido ao seu interpretador. À medida que se programa, o código é interpretado, baixado e armazenado de forma incremental na memória flash, caso o microcontrolador seja desconectado do ambiente de programação, o código estará salvo e será possível executar o programa diretamente na placa, possibilitando que os projetos sejam utilizados em baterias e ao ar livre, garantindo portabilidade e autonomia.

Pode-se destacar estas características importantes presentes na plataforma:

* *Feedback* imediato: É possível executar um bloco individualmente com apenas um clique, receber valores de sensores em tempo real e plotar na ferramenta gráfica.
* Leitura do microcontrolador: Ao conectar novamente uma placa com scripts de MicroBlocks armazenados, fará com que os blocos utilizados reapareçam da mesma maneira que foram programados.
* Paralelismo: MicroBlocks permite escrever scripts separados para cada tarefa e executá-los simultaneamente.

O paralelismo acontece devido ao agendamento de tarefas, a máquina virtual executa algumas instruções do código e alterna em pontos pré-definidos para outro bloco de instruções, causando a impressão de estarem sendo executados paralelamente. Estes pontos pré-definidos encontram-se geralmente no final de um loop ou em uma instrução de espera (MALONEY; MÖNIG ;ROMAGOSA, 2021).

No entanto, a linguagem MicroBlocks também apresenta limitações, como a impossibilidade de converter seus blocos em uma linguagem de programação textual, dificultando a adaptação e aprendizado de novas linguagens. A linguagem suporta apenas números inteiros positivos e negativos, números decimais não podem ser usados, além disto, interrupções e *timers* disponíveis no microcontrolador não possuem blocos para acioná-los.

## BIPES

BIPES é um acrônimo de plataforma integrada baseada em blocos para sistemas embarcados, consiste em um software de código aberto baseado em Blockly, desenvolvido no Brasil em cooperação com as universidades UFSCar, USP, IFRN e UFRN. Seu desenvolvimento iniciou em 2020 e atualmente está na fase beta (JUNIOR et al., 2020).

O BIPES permite que qualquer pessoa projete, programe, construa, implante e teste de forma rápida e confiável sistemas embarcados e dispositivos IoT (JUNIOR et al., 2020). Faz uso das vantagens da programação em blocos aplicado na programação de sistemas embarcados, com o objetivo de facilitar o desenvolvimento de protótipos e auxiliar no aprendizado de novos desenvolvedores.

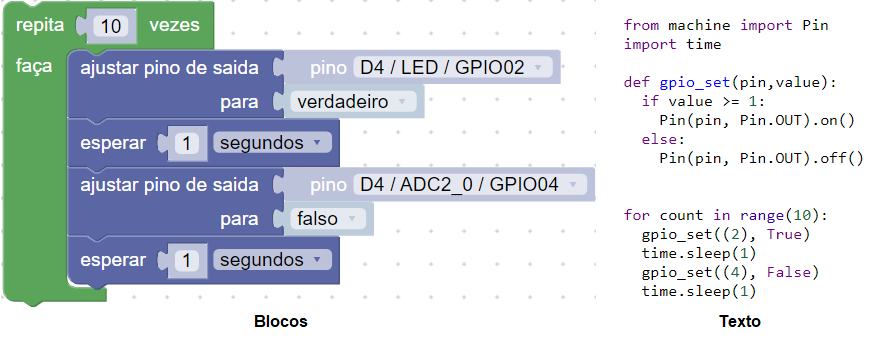
A plataforma roda diretamente no navegador em ambiente *web* sem *downloads* adicionais. É capaz de trabalhar com várias placas e microcontroladores usando linguagens de programação como Python, CircuitPython e MicroPython. O único requisito para o BIPES funcionar é o *firmware* do MicroPython ou CircuitPython estar previamente carregado no dispositivo que irá ser utilizado (JUNIOR et al., 2020).

**Quadro 3 – Placas suportadas pelo BIPES**

| **Placas suportadas** | |
| --- | --- |
| WEMOS D1 Mini | stm32 / mbed |
| NodeMCU | Arduino (UNO, Nano, Pro mini) |
| ESP32 | MicroPython pyboard |
| Raspberry Pi Pico | BeagleBone |
| bbc:microbit | Raspberry Pi (1, 2, 3, 4, Zero) |

Fonte: Adaptado de BIPES (2022).

O código escrito em BIPES é traduzido automaticamente para MicroPython ou CircuitPython, assim que os blocos são adicionados na área de programação sua tradução em formato de linguagem textual fica disponível na aba de arquivos. Deste modo a linguagem se torna um híbrido de programação em blocos e textual, facilitando a adaptação e o aprendizado (BIPES, 2022).

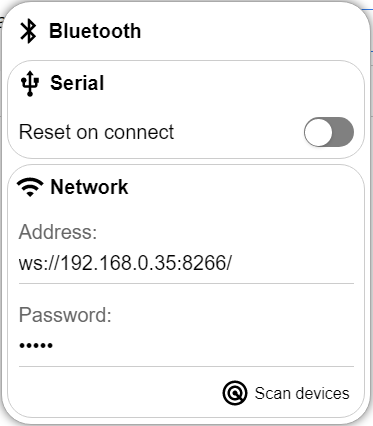
**Figura 9 – Tradução do código em BIPES para MicroPython**

Fonte: Elaboração própria (2022).

A plataforma possui um editor de blocos, semelhante ao Scratch, permite ao usuário criar funções, variáveis, editar e gerenciar códigos existentes e navegar pelas bibliotecas já implementadas, é organizado de forma simples, separando os blocos em categorias e cores. Conta com uma aba para utilização do console, possibilitando a entrada e saída de dados, assim como uma linguagem textual padrão.

A plataforma dispõe uma aba de gerenciamento dos arquivos, permitindo aos usuários enviar arquivos locais diretamente para o dispositivo, que serão armazenados permanentemente no sistema do dispositivo. Esses arquivos também podem ser baixados da placa ou abertos no editor de texto nesta mesma guia (JUNIOR et al., 2020).

É possível fazer a conexão das placas de três formas distintas, como mostrado na Figura 10, está disponível a conexão serial através de USB, conexão WiFi e Bluetooth.

**Figura 10 – Tipos de conexões do BIPES**

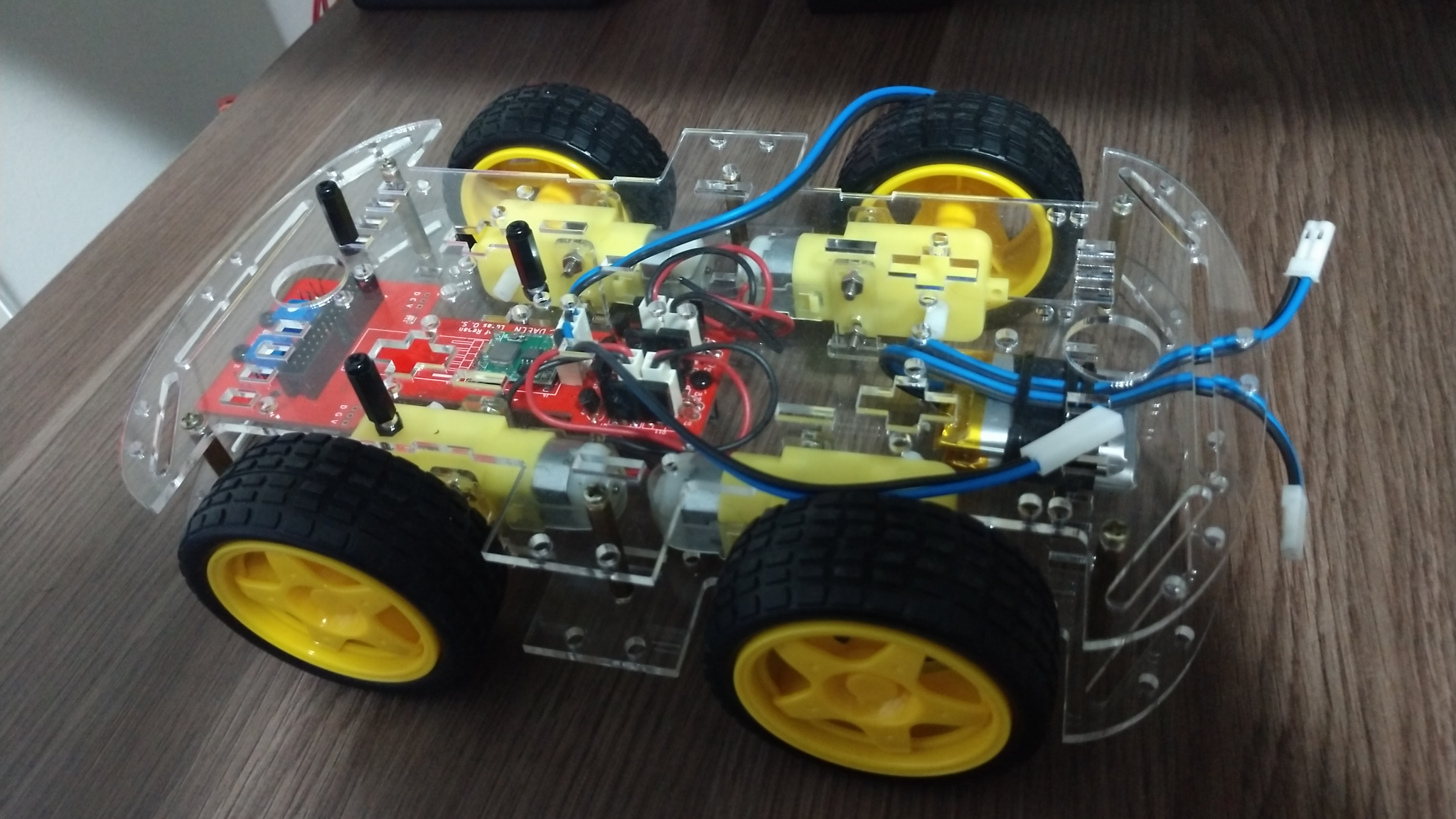
Fonte: BIPES (2022).

Para utilizar uma conexão sem fio, o dispositivo precisa ser configurado usando as ferramentas fornecidas pela plataforma, assim é possível programar em qualquer dispositivo móvel, como celular, tablet ou notebook e enviar o código programado nestes dispositivos diretamente para a placa, sem a necessidade de conectar um cabo (BIPES, 2022).

Como o MicroPython e o CircuitPython são usados ​​para projetos de IoT o BIPES aproveita seus recursos fornecendo módulos para I²C, SPI, *OneWire*, UART, arquivos, sensores, atuadores, displays e redes como HTTP, MQTT e TCP/IP. Além disso, existem blocos para operações com recursos internos do microcontrolador como interrupções, WDT, *deep-sleep* e memória (JUNIOR et al., 2020).

## Buggy AWD

O Buggy AWD apresentado na Figura 11, é um protótipo de carrinho modular autônomo, originalmente construído por Lucas Odair Silva em 2019 como tema do Projeto Integrador 3, atualmente foi modificado pelo Prof. Renan Starke para ser usado como kit de desenvolvimento e ensino, está disponível no Laboratório de Pesquisa Avançada em Eletrônica do IFSC-Florianópolis.

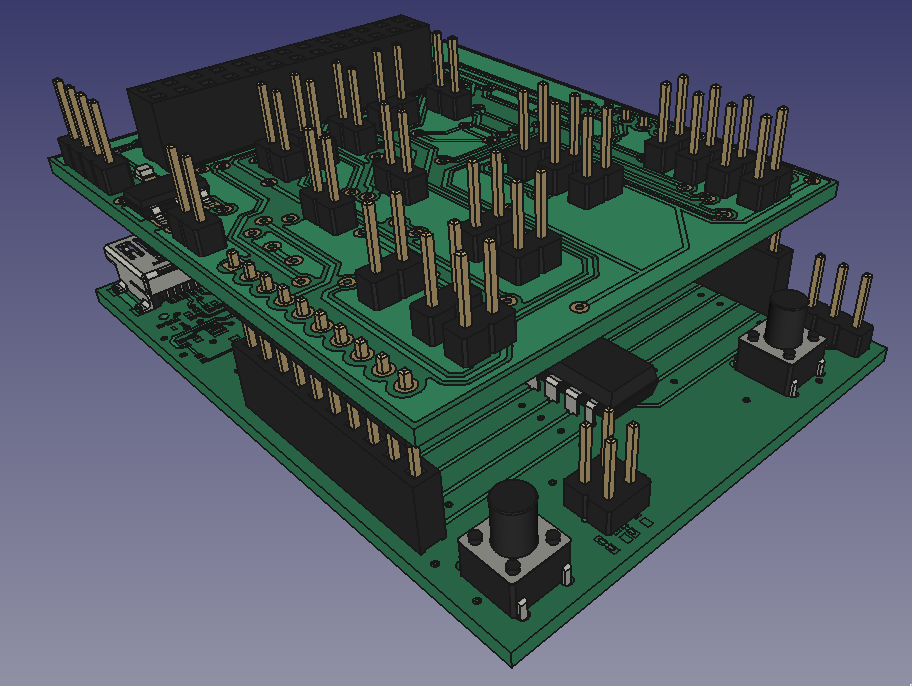
**Figura 11 – Buggy AWD**

Fonte: como posso fazer a fonte? [xtarke/automated\_buggy (github.com)](https://github.com/xtarke/automated_buggy) (2022).

O hardware deste projeto pode ser dividido em três partes, o cérebro como núcleo do sistema, os módulos de sensores e o controle dos motores.

### 2.8.1 Cérebro

O cérebro consiste em um microcontrolador e seu conector de 26 vias que conecta os módulos e motores do carrinho. Por se tratar de um projeto de ensino, foram desenvolvidas placas para fazer a comunicação dos microcontroladores com o Buggy, como mostra a Figura 12, assim como a programação para fazer o controle. Até o momento foram elaboradas placas para o Arduino Uno, MSP-EXP430G2, MSP-EXP430FR2355 e Raspberry Pi. Este presente trabalho irá adicionar uma placa para o kit ESP-WROOM-32, assim como sua programação em MicroPython.

**Figura 12 – *Shield* para MSP-EXP430G2**

Fonte: como posso fazer a fonte? [xtarke/automated\_buggy (github.com)](https://github.com/xtarke/automated_buggy) (2022).

### 2.8.2 Sensores

O Buggy possui sensores de encoder para aferição de velocidade, monitoramento para as baterias de LiPo, sensores infravermelho seguidores de linha, sensor de distância ultrassônico e módulo Bluetooth para comunicação com celulares e aplicativos de programação.

#### 2.8.2.1 Encoder

Para medir a velocidade é utilizado um encoder com sensor óptico, composto por um diodo emissor de luz e um receptor, separados por um disco com perfurações. A velocidade é calculada com base na variação do sensor óptico e devido aos 20 furos presentes no disco, como mostra a Equação 2.1.

(2.1)

Onde:

: velocidade em metros por segundo

: número de pulsos por período

: diâmetro do disco em metros

: tempo entre medições em segundos

#### 2.8.2.2 Monitoramento de baterias

Em geral, uma bateria não pode ser descarregada abaixo de uma determinada tensão, duas baterias LiPo de 3,7 V são utilizadas em série para gerar 7,4 V. Para este projeto foi definido uma tensão segura de 3 V em ambas as baterias, sendo necessário o monitoramento. É utilizado um circuito diferencial com ampop com ganho de 1/3 para a tensão da bateria 1 e de 2/3 para a tensão total em série, tornando-se possível a aferição da tensão com microcontroladores de baixa tensão.

#### 2.8.2.3 Sensores seguidores de linha

O sensor seguidor de linha funciona com sensores de infravermelho, onde um LED emissor e um fototransistor de infravermelho são colocados lado a lado, definindo a saída baixa quando um obstáculo ou objeto passa dentro do alcance de ambos os sensores.

#### 2.8.2.4 Sensor de distância ultrassônico

Para calcular a distância entre o carrinho e algum obstáculo é utilizado um sensor ultrassônico, que utiliza um emissor e um receptor de ultrassom, de modo que o tempo entre os sinais transmitidos e retornados pode ser usado para calcular a distância, como mostrada na Equação 2.2.

(2.2)

Onde:

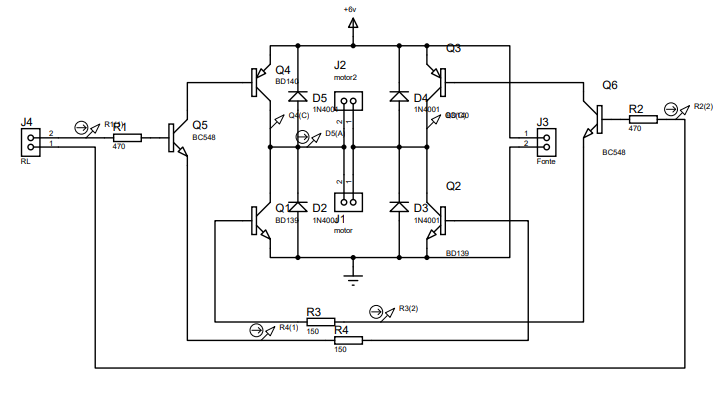
: distância

: tempo do pulso ultrassônico

: velocidade do som

### 2.8.3 Motores

Para controlar os motores, um circuito de ponte H é usado porque a direção do fluxo de corrente pode ser alterada invertendo a tensão nos terminais dos mosfets, permitindo que o carrinho se mova em todas as direções possíveis. Os motores são ligados em paralelo, sendo necessária a utilização de duas pontes para controlar quatro motores.

**Figura 13 – Circuito ponte H utilizado**

Fonte: como posso fazer a fonte? [xtarke/automated\_buggy (github.com)](https://github.com/xtarke/automated_buggy) (2022).

A Figura 13 demonstra o circuito utilizado para fazer o controle de dois motores de uma única vez, com queda de tensão de carga de 0,4 V em relação a tensão de entrada e corrente de polarização externa de 640 pA, não exigindo uma corrente excessiva do microcontrolador.

## Resumo

O que foi usado no carrinho

# METODOLOGIA

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto.

## Métodos aplicados

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto.

# APRESENTAÇÃO DOS RESULTADOS

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto.

## Análise e discussão dos resultados

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto.

# CONSIDERAÇÕES FINAIS

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto.

## Sugestões para trabalhos futuros

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto.

**REFERÊNCIAS**

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. **Compilers: principles, techniques, & tools**. Pearson Education India, 2007.

BARR, Michael. **Programming embedded systems in C and C++**. O'Reilly Media, Inc., 1999.

BATES, Martin P. **Programming 8-bit PIC microcontrollers in C: with interactive hardware simulation**. Newnes, 2008.

BAU, D.; GRAY, J.; KELLEHER, C.; SHELDON, J.; TURBAK, F. **Learnable programming: blocks and beyond**. Communications of the ACM, v. 60, n. 6, p. 72-80, 2017.

BELL, Charles. **MicroPython for the Internet of Things**. Springer, 2017.

BIPES. **BIPES**. Disponível em: https://bipes.net.br/. Acesso em: 26 nov. 2022.

CABALLAR, Rina Diane. **Programming Without Code: the rise of no-code software development**. 2020. Disponível em: https://spectrum.ieee.org/programming-without-code-no-code-software-development. Acesso em: 14 out. 2022.

CABRERA, Lautaro; MALONEY, John H.; WEINTROP, David. **Programs in the Palm of your Hand: how live programming shapes children's interactions with physical computing devices**. 2019. Association For Computing Machinery, New York. Disponível em: https://dl.acm.org/doi/10.1145/3311927.3323138. Acesso em: 16 out. 2022.

GEORGE, Damien. **MicroPython-Python for microcontrollers**. 2014. Disponível em: https://micropython.org/. Acesso em: 14 out. 2022.

GOOGLE. **Blockly**. Disponível em: https://developers.google.com/blockly. Acesso em: 14 out. 2022.

HOROWITZ, Ellis. **Fundamentals of programming languages**. Springer Science & Business Media, 1984.

JUNIOR, A. G. D. S.; GONÇALVES, L. M. G.; CAURIN, G. A. D. P.; TAMANAKA, G. T. B.; HERNANDES, A. C.; & AROCA, R. V. **BIPES: Block Based Integrated Platform for Embedded Systems**. IEEE Access, 2020. Disponível em: https://ieeexplore.ieee.org/document/9246562

KERNIGHAN, Brian W.; RITCHIE, Dennis M. **The C programming language**. Pearson Education, 1988.

KWAME, Ampomah Ernest; MARTEY, Ezekiel Mensah; CHRIS, Abilimi Gilbert. **Qualitative assessment of compiled, interpreted and hybrid programming languages**. Communications on Applied Electronics (CAE), v. 7, n. 7, p. 8-13, 2017.

MALONEY, J. RESNICK, M. RUSK, N. SILVERMAN, B. EASTMOND, E. **The scratch programming language and environment**. ACM Transactions on Computing Education, v. 10, n. 4, p. 1-15, 2010.

MALONEY, John; MÖNIG, Jens; ROMAGOSA, Bernat. **MicroBlocks**. 2021. Disponível em: https://microblocks.fun/. Acesso em: 26 nov. 2022.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY(MIT). **Scratch**. Disponível em: https://scratch.mit.edu/. Acesso em: 14 out. 2022.

MAZZEI, D.; MONTELISCIANI, G.; BALDI, G.; FANTONI, G. **Changing the programming paradigm for the embedded in the IoT domain**. IEEE 2nd World Forum on Internet of Things (WF-IoT). IEEE, 2015. p. 239-244.

MICHELI, Giovanni de; SAMI, Mariagiovanna. **Hardware/Software Co-Design**. Londres: Kluwer Academic Publishers, 1996.

MOORS, Luke; LUXTON-REILLY, Andrew; DENNY, Paul. **Transitioning from block-based to text-based programming languages**. 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE). IEEE, 2018. p. 57-64.

PARDUE, Joe. **C programming for microcontrollers**. Smiley Micros. com, 2005.

RITCHIE, Dennis M. **The development of the C language**. ACM Sigplan Notices, v. 28, n. 3, 1993.

SAMMET, Jean E. **Programming languages: History and fundamentals**. Prentice-Hall, Inc., 1969.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. Bookman Editora, 2018.

SLONNEGER, Kenneth; KURTZ, Barry L. **Formal syntax and semantics of programming languages**. Addison-Wesley, 1995.

STROUSTRUP, Bjarne. **The C++ programming language**. Pearson Education, 2013.

STROUSTRUP, Bjarne. **The essence of C++**. Video. Edinburgh: The University of Edinburgh, 2014.

TOLLERVEY, Nicholas H. **Programming with MicroPython: embedded programming with microcontrollers and Python**. O'Reilly Media, Inc., 2017.

WATSON, Des. **Compilers and Interpreters**. Springer, 2017.

WEINTROP, David. **Block-based programming in computer science education**. Communications of the ACM, v. 62, n. 8, p. 22-25, 2019.

WEINTROP, David; WILENSKY, Uri. **To block or not to block, that is the question: students' perceptions of blocks-based programming**. Proceedings of the 14th international conference on interaction design and children. 2015. p. 199-208.

WEINTROP, David; WILENSKY, Uri. **Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs**. Proceedings of the eleventh annual international conference on international computing education research. 2015. p. 101-110.

**APÊNDICES**

APÊNDICE A – Título

APÊNDICE B – Título

**ANEXOS**

ANEXO A – Título

ANEXO B – Título