

---

**Interface de Programação em Blocos  
para Robôs**

***Francisco de Freitas Pedrosa***

---

# Interface de Programação em Blocos para Robôs

*Francisco de Freitas Pedrosa*

Orientador: Eduardo do Valle Simões

Monografia referente ao projeto de conclusão de curso dentro do escopo da disciplina SCC0293 - Projeto de Graduação I do Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP para obtenção do título de Bacharel em Ciências de Computação.

Área de Concentração: Controle de Robôs Móveis

**USP – São Carlos  
12 de Junho 2023**

# Resumo

No ensino de robótica e de computação, robôs educacionais que se movimentam no chão são uma forma consolidada de começar o ensino de programação. Muitos robôs comerciais foram desenvolvidos com este propósito, sendo que a maioria deles usa linguagens de programação baseada em blocos. Com a popularização de microcontroladores nos últimos anos, foram introduzidas algumas alternativas de código aberto, que visam reduzir o custo e aumentar o acesso a esses robôs. Neste trabalho, foi desenvolvido o projeto de um robô educacional que hospeda uma interface de programação em blocos, que permite programar o robô a partir de um navegador Web, sem a necessidade de softwares adicionais. A interface permite programar o robô usando instruções como 'vá para frente', 'vire para esquerda' e 'vire para direita', além de estruturas de controles como 'repita tantas vezes', 'se então' e 'enquanto'. O robô foi programado com um interpretador para estes comandos, permitindo que a interface envie apenas eles, sem precisar compilar para código de máquina. Também foi adicionado um giroscópio para garantir que o robô consiga girar 90 graus para a esquerda e para a direita. Os resultados experimentais mostram que enviando apenas as instruções praticamente se zera o tempo de envio do código e com o giroscópio o robô consegue realizar as curvas em ângulos retos.

# **Abstract**

In the teaching of robotics and computer science, educational robots that move on the floor are a well-established way to start teaching programming. Many commercial robots have been developed for this purpose, with the majority of them using block-based programming languages. With the popularization of microcontrollers in recent years, some open-source alternatives have been introduced, aiming to reduce costs and increase access to these robots.

In this work, the project of an educational robot was developed, which hosts a block-based programming interface, allowing the robot to be programmed from a web browser without the need for additional software. The interface allows you to program the robot using instructions such as 'go forward', 'turn left', and 'turn right', as well as control structures such as 'repeat as many times', 'if then', and 'while'. The robot was programmed with an interpreter for these commands, allowing the interface to send just them, without having to compile to machine code. A gyroscope has also been added to ensure the robot can rotate 90 degrees left and right. The experimental results show that by sending only the instructions, the sending time is practically zero and with the gyroscope the robot is able to perform the curves at right angles.

# Índice

|   |           |
|---|-----------|
| <b>Lista de Gráficos</b>                      | <b>6</b>  |
| <b>Lista de Tabelas</b>                       | <b>7</b>  |
| <b>Lista de Figuras</b>                       | <b>8</b>  |
| <b>CAPÍTULO 1: INTRODUÇÃO</b>                 | <b>9</b>  |
| 1.1. Contextualização e Motivação             | 9         |
| 1.2. Objetivos                                | 11        |
| 1.3. Organização do Trabalho                  | 12        |
| <b>CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA</b>      | <b>13</b> |
| 2.1. Considerações Iniciais                   | 13        |
| 2.2. Conceitos e Técnicas Relevantes          | 13        |
| 2.2.1. Robótica Educacional                   | 13        |
| 2.2.2. Linguagens de Programação Virtuais     | 14        |
| 2.2.3. Microcontroladores                     | 16        |
| 2.2.4. Interpretadores                        | 18        |
| 2.2.5 Controlador PID                         | 19        |
| 2.3. Trabalhos Relacionados                   | 20        |
| <b>CAPÍTULO 3: DESENVOLVIMENTO DO SISTEMA</b> | <b>22</b> |
| 3.1 Considerações Iniciais                    | 22        |
| 3.2. Estrutura                                | 22        |
| 3.3. Circuito do Robô                         | 24        |
| 3.4. Software                                 | 26        |
| 3.4.1. Inicialização                          | 27        |
| 3.4.2. Servidor Web                           | 27        |
| 3.4.3. Interpretador                          | 27        |
| 3.4.4. Controle de Direção                    | 30        |
| 3.4.4. Controle de Movimento                  | 31        |
| 3.5. Interface de Programação                 | 32        |
| 3.5.1. Padrões Visuais                        | 33        |
| 3.5.2. Gramática de Linguagem                 | 35        |
| <b>CAPÍTULO 4: RESULTADOS EXPERIMENTAIS</b>   | <b>37</b> |
| 4.1. Tempo de Envio do Código                 | 37        |
| 4.2. Trajetória do Robô                       | 38        |
| 4.3. Considerações Finais                     | 41        |
| <b>CAPÍTULO 5: CONCLUSÃO</b>                  | <b>42</b> |
| 5.1. Contribuições                            | 42        |
| 5.2. Considerações Sobre o Curso de Graduação | 42        |
| 5.3. Trabalhos Futuros                        | 43        |
| <b>Referências</b>                            | <b>44</b> |

# **Lista de Gráficos**

|   |    |
|---|----|
| Gráfico 1 - Direção do Robô de Obstáculo com Sonar em Graus                     | 40 |
| Gráfico 2 - Direção do Robô no Trajeto Quadrado Ida e Volta                     | 40 |
| Gráfico 3 - Direção do Robô no Trajeto Quadrado Ida e Volta com Duas Repetições | 41 |

# **Lista de Tabelas**

|   |    |
|---|----|
| Tabela 1 - Comparaçāo Arduino Uno e ESP32                 | 17 |
| Tabela 2 - Instruções e seus Parāmetros                   | 28 |
| Tabela 3 - Ativações dos Motores Pela Ponte-H             | 31 |
| Tabela 4 - Tempo de Envio nos Experimentos                | 37 |
| Tabela 5 - Tamanho dos Arquivos Enviados nos Experimentos | 38 |

# **Lista de Figuras**

|   |    |
|---|----|
| Figura 1 - Exemplos de Blockly e Scratch  | 10 |
| Figura 2 - Exemplo Bloco de Comando do Scratch  | 15 |
| Figura 3 -Exemplo Bloco de Controle do Scratch  | 15 |
| Figura 4 - Exemplo de Bloco de Expressão  | 16 |
| Figura 5 - Exemplo de Bloco de Gatilho  | 16 |
| Figura 6 - Fotografia Frontal e Superior do Robô  | 22 |
| Figura 7 - Dimensões Estrutura do Robô  | 23 |
| Figura 8 - Projeto Estrutura Roda e Motor do Robô   | 23 |
| Figura 9 - Conexões Elétricas e Componentes do Robô   | 25 |
| Figura 10 - Organização dos Módulos de Software do Robô                                     | 26 |
| Figura 11 - A Interface de Programação Vazia  | 33 |
| Figura 12 - A Interface de Programação com um Programa                                      | 33 |
| Figura 13 - Blocos de Comando e Blocos de Controle  | 34 |
| Figura 14 - Blocos de Expressão e Blocos de Gatilho   | 34 |
| Figura 15 - Trajeto Desviando de Obstáculo com Sonar  | 39 |
| Figura 16 - Trajeto Quadrado Ide a Volta e Trajeto Quadrado Ida e Volta com Duas Repetições | 39 |

# CAPÍTULO 1: INTRODUÇÃO

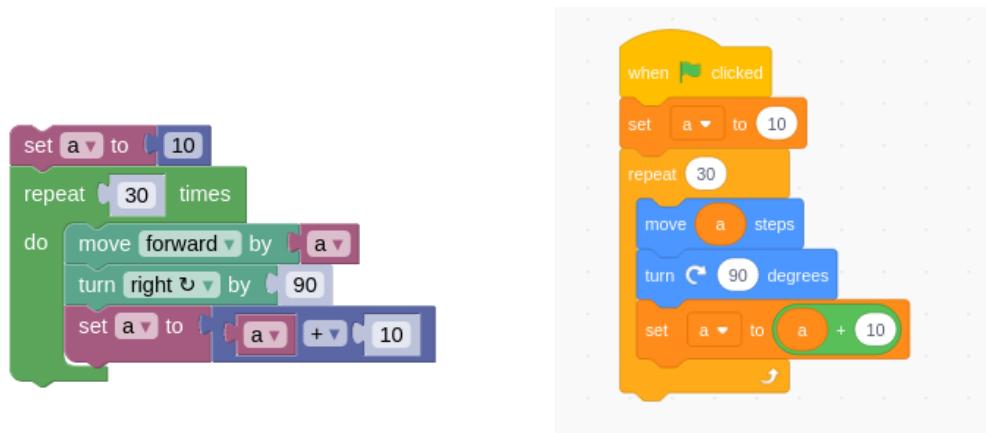
## 1.1. Contextualização e Motivação

Em 1980 Seymour Papert em seu livro *Mindstorm* descreveu o ambiente Logo, um programa no qual o usuário poderia usar uma linguagem de programação para controlar a movimentação de uma tartaruga, que poderia ser tanto um desenho na tela quanto um brinquedo mecânico que realmente se movia (PAPERT, 1993). Papert defendia que era um bom jeito de introduzir crianças a programação, ao programar e acompanhar a tartaruga executando a programação elas desenvolvem uma intuição, que se desenvolveria para uma raciocínio matemático. Desde então, a evolução tecnológica permitiu que computadores e a robótica ficassem muito mais acessíveis (PAPADAKIS, 2020). A robótica na educação tomou grandes proporções, várias escolas incluíram em seus currículos e ela até pode ser considerada uma tendência nos debates de modernizar a educação. Com isso, uma série de Kits Educacionais de Robóticas foram desenvolvidos, como por exemplo o Edison Robot, Coji Robot, Fynch, Thymio e Ozobot. Muitos desses kits são similares ou até diretamente inspirados pelo ambiente Logo e sua tartaruga.

Além dos Kits comerciais também surgiram, dentro e fora da academia, projetos similares de código-aberto que visam tornar os robôs ainda mais acessíveis, reduzindo custos ou aumentando a adaptabilidade dos robôs, para que eles possam ser modificados e realizem ainda mais funções. Exemplos são um grupo de pesquisadores da Unicamp que projetou um desses kits usando o microcontrolador Arduino Uno em 2013 (JUNIOR *et al*, 2013) e Ílias Kampouridis que fez um projeto educacional inspirado no BeeBot em 2023 usando o microcontrolador ESP32 (KAMPOURIDIS, 2023).

Outro avanço importante no ensino da computação para crianças, popularizado no século XXI, foi o desenvolvimento das linguagens de programação baseadas em blocos, por exemplo Scratch ou Blockly. As linguagens de programação tradicionais dependem de sintaxes muito específicas, que podem ser frustrantes para iniciantes (HORN; BERS, 2019). Linguagens de programação baseadas em blocos visam resolver esse problema, substituindo instruções escritas por recursos visuais chamados de blocos que podem ser movidos na tela e se encaixam entre si. As regras sintáticas de como esses blocos devem se encaixar são sugeridas por recursos visuais intuitivos como cores e formas, como pode ser visto na Figura 1.

**Figura 1 - Exemplos de Blockly e Scratch**



O uso de programação em blocos na robótica educacional também é bastante consolidado. O kit desenvolvido na Unicamp (JUNIOR *et al*, 2013) usa o ambiente de desenvolvimento integrado (IDE na sigla em inglês) Minibloq para programar o Arduino. Patrick Lamprecht, Simon Haller-Sieber, e Justus Piater (2021) apresentam uma extensão para a ferramenta Ardublockly para uma ESP32 controlar até 2 motores, o mesmo procedimento que todos os robôs citados acima usam para realizar os comandos ‘frente’, ‘esquerda’ e ‘direita’. Programação em blocos também é uma tendência entre os robôs educacionais comerciais, como é o caso do Edison Robot, Coji Robot, Fynch, Thymio e Ozobot (PAPADAKIS, 2020)

Outro aspecto importante desses robôs é como o código é enviado para o robô. No Arduino UNO isso se dá por um cabo de USB (Porta Serial Universal na sigla em inglês), que conecta o robô ao computador. Dessa forma, a cada vez que o robô é reprogramado é necessário que ele seja trazido para perto do computador e depois seja levado de volta para o ambiente onde se moverá. Isso é especialmente demorado em um contexto educacional no qual os alunos precisam reprogramar o robô diversas vezes para testar seus códigos. Outra opção para programar o Arduino é usar a biblioteca ArduinoOTA, (acrônimo para Over The Air, ou “pelo ar” em português) (CURVELLO, 2017), que se incluída no código permite que o microcontrolador seja reprogramado por Wi-Fi, sem a necessidade de se conectar fisicamente ao computador. Isso é especialmente interessante quando usamos a ESP32, pois ela, ao contrário do Arduino Uno, já vem com suporte Wi-Fi e Bluetooth no hardware, portanto não necessita de componentes adicionais para se conectar (ESPRESSIF SYSTEM, 2016a). Entre os robôs comerciais não foi encontrado exemplos que usam Wi-Fi, mas o Coji, o BlueBot e o Ozobot Evo também não dependem de cabos, pois podem ser programados por

Bluetooth. Mesmo com ArduinoOTA o tempo de programar o Arduino ou a ESP32 ainda é grande, visto que é necessário compilar e enviar todo o software do microcontrolador, o que no caso da ESP32 também inclui o sistema operacional FreeRTOS (Free Real Time Operating System).

Além da demora na programação, a baixa fidelidade entre o percurso realizado pelo robô e os comandos propostos também é um problema encontrado nos robôs de código aberto e alguns robôs comerciais (DAVCEV; KOCESKA; KOCESKI, 2019). Existem diversos fatores que vão interferir na movimentação, como irregularidades no solo e diferenças entre as rodas. Dessa forma, é provável que o robô penda para um lado, quando deveria andar para frente e não realize curvas com exatidão. Diversos sensores podem ser usados para aumentar a precisão do robô, e várias soluções podem ser encontradas em contextos que exigem precisão muito maior que a robótica educacional. Por exemplo, Patil e Agiwal (2015) traçam o percurso de robô usando uma bússola eletrônica e um sonar, e Paralija (2022) publicou em um blog de robótica, o projeto de um robô que usa um giroscópio para balancear dois motores para andar em linha reta.

## 1.2. Objetivos

Considerado a capacidade do hardware do ESP32 de se conectar com redes de Wi-Fi e até mesmo servir como ponto de acesso, é possível o próprio robô gerar uma rede Wi-Fi e hospedar uma página Web com uma interface gráfica para programá-lo em uma linguagem baseada em blocos. Dessa forma qualquer dispositivo com Wi-Fi e um navegador web será capaz de programar o robô. Se o conjunto de instruções for suficientemente simples, seria possível fazer um robô que interprete diretamente esses comandos, sem que haja a necessidade de compilá-los para código de máquina. Seria apenas necessário passar um código representando os blocos, reduzindo muito o tempo de programação do robô. Com o uso de giroscópio o robô deve ser capaz de realizar movimentos em linha reta e curvas mais precisas para que o usuário possa reconhecer sua trajetória como a programada nos blocos.

Sendo assim, este trabalho tem o objetivo de desenvolver um robô simples, controlado por um ESP32, que gere uma rede Wi-Fi e hospede uma interface de programação em blocos, que permita que leigos programem o robô em um navegador web sem a necessidade de instalar nenhum software adicional. O robô deverá ter capacidade de interpretar e executar esses blocos sem que haja necessidade de reprogramar todo o sistema operacional do robô.

Esses blocos devem incluir comandos de movimentação como ‘Andar para frente’, ‘Virar para esquerda’, ‘Virar para direita’, estruturas de controle como ‘Repetir N vezes’, ‘Repetir enquanto’, ‘Se então senão’ e a leitura de um sensor ultrassônico de distância. O robô usará um giroscópio para controlar sua trajetória. Este trabalho também deve realizar experimentos para medir o tempo de programação, comparar esses resultados com o robô que tem que receber todo o software compilado toda vez que é reprogramado e medir a precisão das trajetórias desse robô.

### **1.3. Organização do Trabalho**

No capítulo 2 são apresentados outros trabalhos que inspiraram este, e quais técnicas eles usam para enfrentar os problemas encontrados. No capítulo 3 é descrito como foi a implementação deste trabalho separando-a em módulos. No capítulo 4 são descritos os experimentos realizados e os resultados obtidos. Finalmente no capítulo 5 encontram-se as conclusões e trabalhos futuros.

# CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA

## 2.1. Considerações Iniciais

Nesta seção será realizada uma revisão bibliográfica dos principais tópicos relacionados a este projeto. Primeiramente, será discutido o uso de robótica na educação, seu embasamento teórico, sua evolução e suas tendências atuais. Em seguida, será apresentado o estado da arte das Linguagens de Programação Visual, principalmente, focando no Scratch como estudo de caso. Então será discutido o uso dos microcontroladores, como o Arduino e o ESP32, e será realizada uma comparação técnica entre os dois. Em seguida será apresentada a teoria dos interpretadores e como ela se relaciona com gramáticas formais. Por fim, serão apresentados os controladores PID e como eles são uma das principais técnicas de controle usadas na atualidade.

## 2.2. Conceitos e Técnicas Relevantes

### 2.2.1. Robótica Educacional

A robótica educacional é fundamentada pela teoria educacional do Construcionismo, criada por Seymour Papert adaptada do Construtivismo, que defende que o aprendizado deveria ser marcado pelo protagonismo dos alunos, que deveriam aprender conceitos novos explorando objetos tangíveis, de forma que facilite a compreensão dos conceitos e motive o aprendizado (ANWAR *et al*, 2019). Papert relata que durante sua infância ele tinha um brinquedo de construção com um kit de engrenagens, do qual gostava bastante. Ao aprender na escola sobre tabuadas e equações com duas variáveis, Papert relacionou esses conceitos com engrenagens, o que segundo ele não só o ajudou a entender esses conceitos, mas lhe fez gostar mais de matemática. Quando adulto, Papert defendeu que computadores tinham grande potencial de serem para outras crianças o que as engrenagens foram para ele, como eles eram extremamente flexíveis poderiam atrair diversos gostos (PAPERT, 1993). Com esse propósito ele co-desenvolveu em 1966 a linguagem de programação LOGO, que ele usava para ensinar crianças. Inicialmente, as atividades se limitavam a manipular textos e números, mas entre 1970 e 1971 ele criou a Tartaruga Logo, um robô que se movimentava seguindo comandos da linguagem, como “FRENTE 100” e “ESQUERDA 90”. Também foi criada uma versão virtual da Tartaruga, que seguia os comandos em uma interface gráfica na tela do computador (SOLOMON *et al*, 2020).

Na década de 1980 o laboratório em que Papert trabalhava no Massachusetts Institute of Technology (MIT), o Media Lab, fez uma parceria com a empresa LEGO, para produzir novas ferramentas que funcionassem com LOGO. Em 1989 foi feito o *6250 Programmable Brick*, um peça compatível com o Lego, que tinha um processador rodando um interpretador de LOGO e podia ser conectado com várias outras peças como motores e sensores. Outras versões desse bloco foram feitas como o *Pocket Programmable Brick* em 1995 e o *Model 120 Programmable Brick* de 1996, para essa versão Andrew Begel desenvolveu uma interface de programação em blocos chamada *LogoBlocks*. Em 1998 a Lego lançou um kit com seu próprio bloco programável independente do Media Lab, chamado *LEGO Mindstorms*, nome que homenageia um livro de Papert. O kit da LEGO também tinha uma interface de programação em blocos parecida com *LogoBlocks* (BELAND *et al* , 2000).

Inicialmente Papert e seu colega Brian Silverman duvidaram da utilidade das linguagens de programação em blocos, mas mudaram de opinião nos primeiros testes do LogoBlocks. Silverman Chegou a dizer:

*“Nós testamos com crianças e professores. Para minha surpresa, mas não para o Mitchel [Resnick], funcionou muito, muito bem. O que Seymour [Papert] e eu não tínhamos antecipado é que por parecer mais simples deixava as pessoas mais dispostas a engajar nos estágios iniciais. De alguma forma não era mais simples, só parecia e isso que importava.”*

(SOLOMON *et al*, 2020, tradução nossa)

Um dos problemas que eles tinham com esse tipo de linguagem é que elas eram muito limitadas, e não podiam realizar tudo que o LOGO em texto podia, isso se resolveu quando criaram interfaces que traduzem o código em blocos para código em texto (BELAND *et al*, 2000). Hoje linguagens baseadas em blocos são o padrão quando se pensa em linguagens voltadas para crianças, principalmente porque elas não exigem que se saiba digitar (SOLOMON *et al*, 2020).

### 2.2.2. Linguagens de Programação Virtuais

Linguagens de programação visuais são linguagens que usam recursos gráficos para definir um programa, elas estão cada vez mais populares, geralmente em contextos em que a programação deve ser realizada por pessoas que não são especialistas, como automação residencial ou no ensino de programação para crianças. Existem dois estilos principais de programação visual: fluxogramas e em blocos. Fluxogramas são a abordagem mais

tradicional, que consiste em ter diversas formas que têm suas relações indicadas por setas apontadas para outras formas. Já os blocos são uma abordagem mais moderna, muito consolidada na programação com crianças. Nessa abordagem as formas são encaixadas umas nas outras, dessa maneira elas podem ter formas com dicas visuais que mostram como elas devem se encaixar, e podem eliminar completamente erros de sintaxe, porque blocos que não podem ser usados em sequência não se encaixam (NOONE; MOONEY, 2018).

Um artigo escrito pelos desenvolvedores do Scratch (MALONEY *et al*, 2010) define que Scratch tem 4 tipos de blocos, cada um com dicas visuais de como devem ser usados:

**Blocos de Comandos** são os blocos que contém ordens simples a serem executadas, depois deles sempre pode vir outro comando, e eles podem vir de depois de qualquer comando, por isso são visualmente representados com uma saliência na parte superior e uma cavidade na parte inferior, como pode ser visto na figura 2. Vários comandos em sequência são chamados de cadeia.

**Figura 2 - Exemplo Bloco de Comando do Scratch**



**Blocos de Controle** são um tipo de comando especial que tem uma ou mais cadeias de comandos dentro dele, isso é representado por uma abertura na direita com formato de um comando, como pode ser visto na figura 3. Essa abertura cresce se forem adicionados mais blocos na cadeia. Recebem esse nome porque eles controlam o fluxo do programa. Na maioria das linguagens de programação em texto esses blocos seriam representados por um símbolo no começo e outro no final, como chaves em C. Por isso, não escrever o símbolo de fechamento gera um erro. Em Scratch e outras linguagens visuais, isso é impossível porque o bloco é um símbolo único inseparável.

**Figura 3 -Exemplo Bloco de Controle do Scratch**



**Blocos de Expressão** são os blocos que retornam algum valor, eles não encaixam sequencialmente nas cadeias, apenas podem ser encaixados dentro de outros blocos. O formato dos blocos depende do tipo que eles retornam. Se retornam valores booleanos eles são hexagonais, se retornam números ou strings eles são arredondados, como pode ser visto na figura 4. Os blocos em que podem ser encaixados exibem entalhes nos mesmos formatos.

**Figura 4 - Exemplo de Bloco de Expressão**



**Blocos de Gatilho** são os blocos que dão início às cadeias, por isso não podem seguir outros comandos, só podem ser seguidos por outros comandos. Visualmente isso é representado com uma curva na parte superior, que não se encaixa em nenhum outro bloco, como pode ser visto na figura 5.

**Figura 5 - Exemplo de Bloco de Gatilho**



### 2.2.3. Microcontroladores

Arduino é uma placa microcontroladora desenvolvida na Itália em 2005 (ARDUINO, 2022). Existe mais de um modelo da placa, mas sua versão mais tradicional, o Arduino Uno, usa chip ATmega328 e também inclui 20 pinos de entradas ou saídas com conectores fêmeas, que podem ser facilmente conectados sem necessidade de solda. Também possuem um módulo de comunicação USB que permite mudar a programação do chip conectando-o a um computador. Além disso, também foi feito um programa de computador, o Arduino IDE, que pode ser usado para desenvolver o código, gerenciar as bibliotecas utilizadas e enviar o código para a placa. Devido a sua simplicidade, o Arduino alcançou grande popularidade, sendo muito utilizado para prototipação e por amadores. Hoje o Arduino é muito associado à cultura Maker e ao movimento Faça Você Mesmo (DIY na sigla em inglês). Existe uma forte comunidade que compartilha projetos, tutoriais e bibliotecas para o Arduino. Os Arduinos

também ganharam espaço nas salas de aulas, sendo muito comuns em aulas de robótica (KOPECKA-PIECH, 2018).

O Arduino tem Hardware e Software livres, que podem ser reproduzidos livremente, isso fez com que surgissem outras placas similares, entre elas destaca-se a linha ESP criada pela empresa chinesa Espressif Systems, que lançou em 2014 a placa ESP8266 (BENCHOFF, 2014). Inicialmente, a placa tinha pouca documentação em inglês, mas devido ao baixo custo atraiu o interesse da comunidade que veio a traduzir a documentação, hoje as placas ESP são compatíveis com as bibliotecas criadas para Arduino e podem ser programadas usando a Arduino IDE. Além do baixo preço, as ESP também se destacam por ter antenas WiFi e Bluetooth embutidas e mais capacidade de processamento. Na Tabela 1 faz-se um comparativo entre um arduino Uno e uma ESP32, modelo sucessor da ESP8266 lançada em 2016 (ESPRESSIF SYSTEMS, 2016c) (ARDUINO COMPANY, 2021), produtos que têm custos similares.

**Tabela 1 - Comparaçāo Arduino Uno e ESP32**

|                                      | <b>Arduino UNO</b> | <b>ESP32</b>        |
|--------------------------------------|--------------------|---------------------|
| RAM                                  | 2 KB               | 520 KB              |
| Memória Flash                        | 32 KB              | 4 MB                |
| ROM                                  | 1 KB               | 128 KB              |
| Pinos Analógicos                     | 6                  | 18                  |
| Pinos com PWM                        | 6                  | 16                  |
| Pinos Totais                         | 20                 | 25                  |
| Clock                                | 16 MHz             | entre 160 e 240 MHz |
| Cores                                | 1                  | 2                   |
| Preço (CurtoCircuito.com.br - 06/23) | R\$ 63             | R\$ 58              |

## 2.2.4. Interpretadores

Interpretadores são programas que recebem como entrada um código fonte e o executam. Para isso, é preciso que o código fonte esteja escrito seguindo um padrão pré definido, ao que se dá o nome de linguagem de programação. Essas linguagens geralmente recebem o nome de linguagens interpretadas, em oposição às linguagens compiladas, que primeiro são traduzidas para código de máquina, para depois serem executadas. Esse processo de traduzir uma linguagem para outra é chamado de compilação, daí vem o nome linguagem compilada, mas não necessariamente a compilação precisa produzir código de máquina. A vantagem do código de máquina é que ele não necessita de um programa interpretador para ser executado, pois ele é interpretado diretamente pela CPU. Vale notar que a diferença entre linguagens compiladas e interpretadas não é uma característica das linguagens em si, é simplesmente a forma que são executadas, porque todas as linguagens podem ser compiladas para código de máquina, ou interpretadas diretamente.

Para se definir formalmente uma linguagem, tem-se que definir uma gramática e uma semântica. A gramática é o conjunto de regras de como o programa deve ser escrito, enquanto a semântica é o significado do programa, como ele executa. É bastante difícil definir a semântica de uma linguagem, mas pode-se defini-la informalmente ou descrevê-la baseando-se em outra linguagem. Já a gramática pode ser formalmente definida por:

**Símbolos Terminais:** Símbolos terminais são os símbolos que compõem a gramática, geralmente são caracteres ou palavras dependendo do contexto.

**Símbolos Não-Terminais:** São símbolos que não fazem parte da linguagem, mas são equivalentes a outras sequências de símbolos terminais ou não terminais, de acordo com as regras de produção. Neste trabalho símbolos não-terminais serão escritos em negrito.

**Símbolo inicial:** Um dos símbolos não-terminais que produz todas as sequências válidas.

**Regras de Produção:** São regras que definem quais sequências de símbolos são equivalentes a cada símbolo não terminal. Regras são escritas com o símbolo não terminal a esquerda, seguido por uma seta ( $\rightarrow$ ), e depois vêm as opções de sequência equivalentes ao símbolo separadas por barras verticais ( $|$ )

Um exemplo é a seguinte gramática. Os símbolos terminais são os [0,1,2,3,4,5,6,7,8,9,+,\*,(,)]. Os símbolos não-terminais são (**A**, **B**). O símbolo inicial é **A** e as regras de produção são:

$$\begin{aligned}\mathbf{A} &\rightarrow (\mathbf{A}) \mid \mathbf{A} + \mathbf{A} \mid \mathbf{A} * \mathbf{A} \mid \mathbf{B} \\ \mathbf{B} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

Partindo do símbolo inicial pode-se seguir as regras de produção para produzir a sequência  $3 * ( 5 + 4 )$  da seguinte forma:  $\mathbf{A} \rightarrow \mathbf{A} * \mathbf{A} \rightarrow \mathbf{B} * \mathbf{A} \rightarrow 3 * \mathbf{A} \rightarrow 3 * (\mathbf{A} + \mathbf{A}) \rightarrow 3 * (\mathbf{B} + \mathbf{A}) \rightarrow 3 * (5 + \mathbf{B}) \rightarrow 3 * (5 + 4)$ .

Ao ler uma expressão como essa, um interpretador precisa calcular a ordem das operações a serem realizadas, que é diferente da ordem das operações escritas, seria mais simples para o interpretador se elas já estivessem ordenadas, para isso pode-se realizar um passo antes da interpretação de traduzir (compilar) essa linguagem para uma representação intermediária na qual as operações estejam ordenadas. Um exemplo de uma representação assim é a notação posfixa, na qual a expressão ficaria  $3 \ 5 \ 4 \ + \ *$ . Para um interpretador interpretar a notação pós-fixa basta ler a sequência da esquerda para direita, se encontrar um valor numérico insira-o em uma pilha, se encontrar uma operação faça ela entre os valores de cima da pilha e insira o resultado da operação na pilha. Isso só funciona porque se sabe o número de valores utilizados em cada operação, e consequentemente quantos valores devem ser retirados da pilha ao encontrar um operador. Se a gramática tiver operações como MMC e MDC que podem ser realizadas entre dois ou mais números seria impossível distinguir se a sequência  $2 \ 3 \ 4 \ 5 \ \text{MMC} \ \text{MDC}$  deve ser interpretada como:  $\text{MDC}(\text{MMC}(5, 4, 3), 2)$  ou  $\text{MDC}(\text{MMC}(5, 4) 3, 2)$ . Nesse caso é necessário ter um símbolo para denotar o fim da lista de argumentos (AHO *et al.*, 2007).

## 2.2.5 Controlador PID

O controlador proporcional integral derivativo (PID) é um algoritmo de controle, amplamente usado. Ele pode ser usado em processos em que se controla uma variável de entrada que influencia uma variável de saída. Baseia-se no princípio do feedback, isso é o erro da variável de saída vai ser constantemente medido e usado para determinar o valor da variável de entrada. Esse erro é definido pela diferença entre o valor real e o valor desejado. Porém nos controladores PID não só o erro é considerado, como também sua integral e sua

derivada, cada um desses elementos influencia o processo de uma forma (ÅSTRÖM; HÄGGLUND, 1995).

A ação proporcional é calculada a partir do erro em si, ele é o elemento mais simples se o erro está para um sentido, o controle é acionado no outro.

A ação derivativa serve para contornar a demora entre uma mudança no controle e a resposta na saída. A derivada indica a velocidade que o erro está mudando. Se ele estiver diminuindo muito rápido, o controle está exagerando na resposta, e vai ultrapassar o estado desejado, por isso a ação derivativa vai diminuir a resposta. Se o erro estiver aumentando muito rápido a ação derivativa vai se juntar a ação proporcional para aumentar o controle no sentido oposto.

A ação integrativa serve para anular vieses do processo. Se usarmos só as ações proporcional e derivativa, sempre que o erro se estabilizar em zero, o controle também vai zerar, mas se o processo tiver um viés é possível que seja necessário manter o controle em determinado valor para manter o erro nulo. A ação integrativa vai usar o histórico todo do erro para aprender esse viés.

Matematicamente, tem-se a fórmula:

$$u(t) = K_p \left( e(t) + \frac{1}{T_1} \int_0^t e(\sigma) d\sigma + T_d \frac{de(t)}{dt} \right)$$

Onde  $u(t)$  é o valor do controle no tempo  $t$ ,  $e(t)$  é o valor do erro no tempo  $t$ .  $K_p$ ,  $T_i$  e  $T_d$  são constantes. Porém, pode-se aproximar na forma discreta (ATMEL CORPORATION, 2006):

$$u(t) = K_p e(t) + K_i \sum_{k=0}^t e(k) + K_d (e(n) - e(n-1))$$

Onde  $K_p$ ,  $K_i$  e  $K_d$  são outras constantes.

## 2.3. Trabalhos Relacionados

Alguns trabalhos que serviram de inspiração para esse projeto são:

***LogoBlocks: a graphical programming language for interacting with the world*** (BEGEL, 1996), onde Andrew Begel apresenta a implementação de *LogoBlocks*, uma linguagem visual para programar o *Programable Brick*. Neste trabalho vê-se a criação de uma gramática para blocos e compilação dela para uma notação pós-fixa que o *Programable Brick* possa interpretar.

O artigo **The Scratch Programming Language and Environment** (MALONEY *et al.*, 2010) escrito por quatro dos desenvolvedores do Scratch. Nele encontra-se uma apresentação dos padrões visuais que Scratch usa. Esses padrões foram a principal influência para a linguagem de programação desenvolvida neste trabalho.

O artigo **A Low-Cost and Simple Arduino-Based Educational Robotics Kit** (JUNIOR *et al.*, 2014) descreve a implementação de um robô educacional programável em linguagem de blocos. O robô desse trabalho tem o mesmo objetivo, porém eles diferem pois este usa o ESP32 ao invés de um Arduino UNO, este se movimenta a partir de comandos do programador, o outro usa seguidor de linha, e neste trabalho tem-se uma interface de programação hospedada no próprio robô, que pode ser acessada por qualquer computador com navegador via WI-FI sem a necessidade de softwares adicionais.

O trabalho **Development and Design of Educational Robot Using ESP32 Microcontroller** (KAMPOURIDIS, 2023) em que é também implementado um robô educacional, neste caso usando uma ESP32. Ao contrário do anterior, este robô se movimenta a partir de comandos, e usa um ESP32. Porém ele não tem interface de programação em blocos e usa apenas botões no próprio robô. Este robô também não tem um controle de direção com feedback, usa apenas um tempo pré-determinado para realizar curvas de 90 graus.

# CAPÍTULO 3: DESENVOLVIMENTO DO SISTEMA

## 3.1 Considerações Iniciais

Para o desenvolvimento deste trabalho foi necessário projetar e construir um robô de pequeno porte que fosse capaz de realizar a movimentação proposta, desenvolver o software para controlá-lo e então realizar os experimentos propostos. Partes do robô foram reaproveitadas de um outro projeto do professor Eduardo Simões com o grupo de extensão Principia - Robôs na Escola do ICMC-USP, inclusive seu chassi e alguns componentes eletrônicos (SIMÕES, 2021). Porém, são originais deste trabalho a interface de programação, a linguagem de blocos, o interpretador desta linguagem, o controle de direção e o controle de movimentação. Vê-se na figura 6 uma imagem frontal e uma imagem superior do robô. Seu projeto é simples e de baixo custo (cerca de R\$ 300,00) e pode ser construído por professores e crianças a partir de 10 anos de idade.

**Figura 6 - Fotografia Frontal e Superior do Robô**

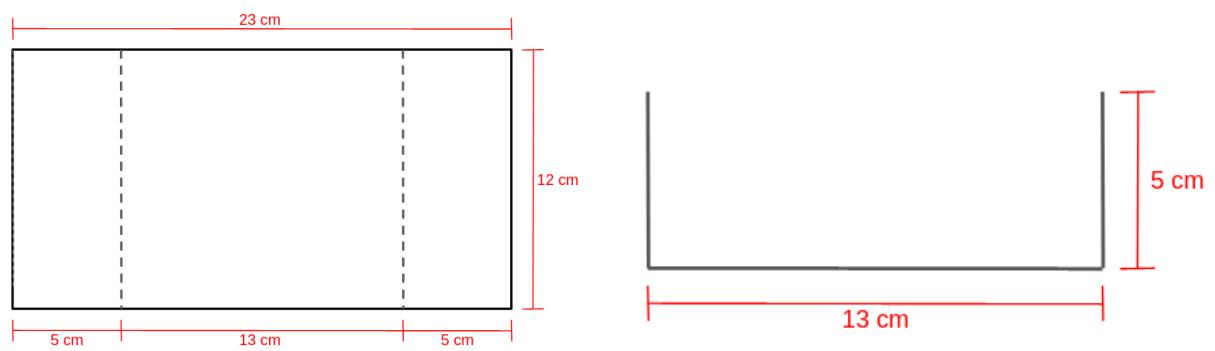


## 3.2. Estrutura

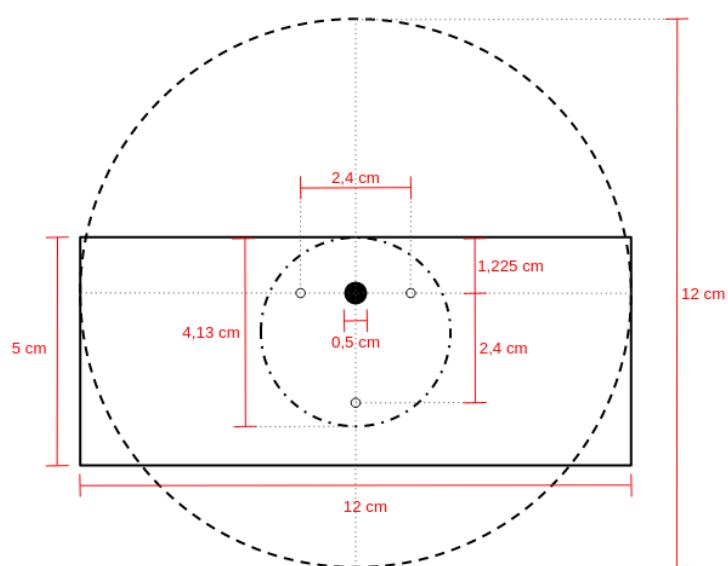
O componente principal da estrutura do robô é uma placa de alumínio de 12 cm x 22 cm. Uma aba foi dobrada 90° nos dois lados conforme a figura 7. Em cada aba foi parafusado um motor. O motor utilizado foi o MICRORED com microreduutor modelo A, na versão com 6,05 cm de comprimento, cujas especificações podem ser vistas em (MICRORED). Para fixar o motor foram feitos três furos em cada aba alinhados com os furos rosados do motor

representados por um circulado vazio na figura 8; um quarto furo foi realizado para passar o eixo do motor, que está representado por um círculo preenchido na figura 8. Para servir como roda foram usados dois DVDs de cada lado, que por padrão tem 12 cm de diâmetro. Cada par de DVD foi colado com cola quente, de forma que ficasse bastante cola quente no centro dos discos, depois da cola endurecer foi feito um furo no centro do monte, com uma broca um pouco menor que a espessura do eixo do motor. Esses furos foram usados para encaixar as rodas nos motores, como os furos são menores que o eixo, a elasticidade da cola aperta o eixo mantendo assim as rodas fixas.

**Figura 7 - Dimensões Estrutura do Robô**



**Figura 8 - Projeto Estrutura Roda e Motor do Robô**

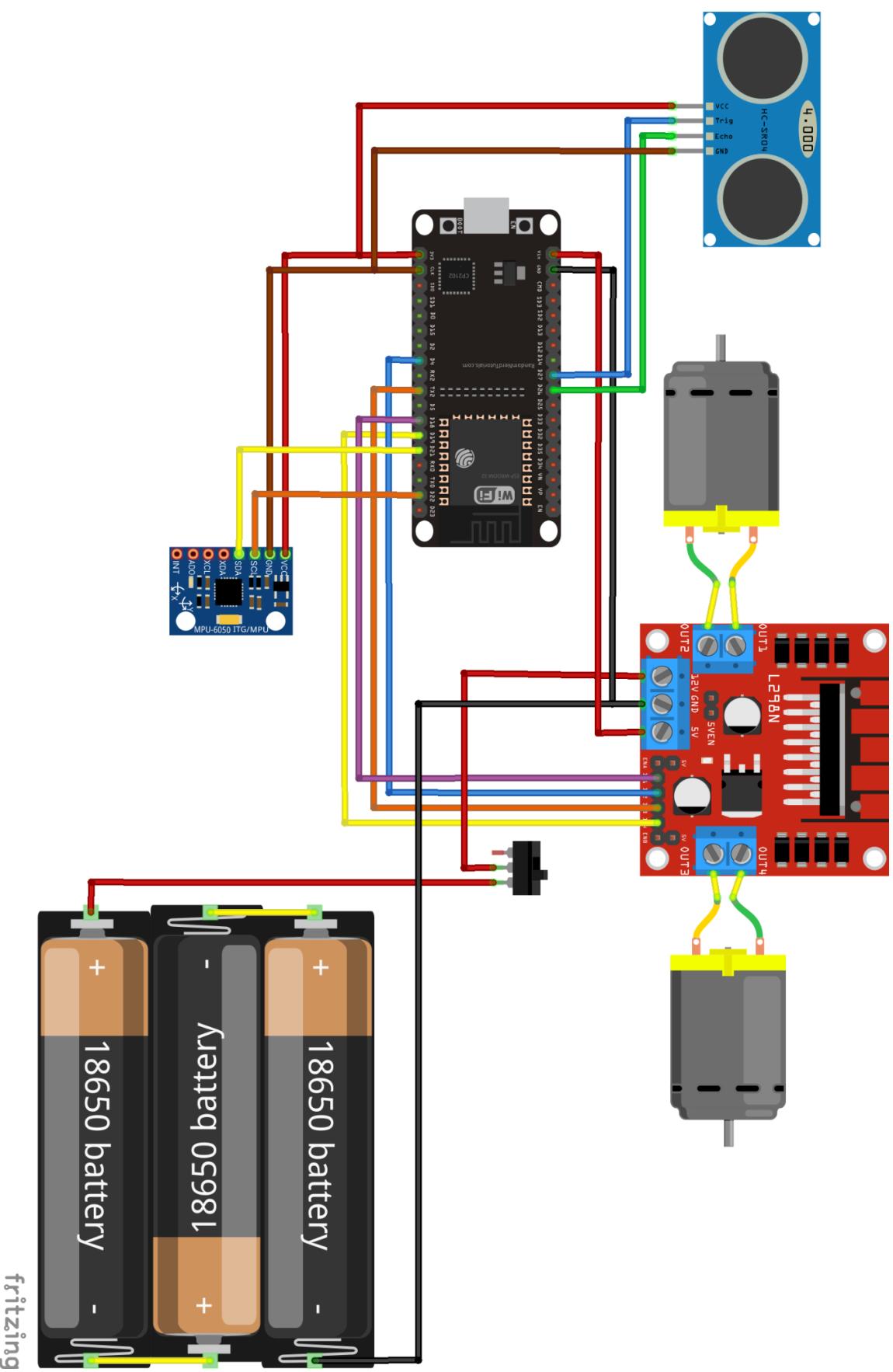


Uma terceira roda menor foi adicionada em baixo da parte traseira do robô. O eixo da terceira roda é uma haste metálica, com três aros de plástico que podem girar livremente. Esta haste foi fixada na parte de baixo da placa com cola quente. Com a terceira roda a placa metálica se mantém na horizontal sem tocar no chão. Além da placa de alumínio, também compõe a estrutura do robô uma caixa de sorvete virada para baixo, que foi recortada para encaixar com os motores. Com exceção dos motores e da bateria, os demais componentes elétricos foram colados em cima da caixa de sorvete também com cola quente. Foram realizados furos na caixa para passar os fios que conectam os componentes de cima com os componentes de baixo da caixa.

### 3.3. Circuito do Robô

Toda energia do robô vem de três baterias recarregáveis de íon de lítio de 4.2V modelo 18650, elas estão ligadas em série em um suporte de baterias totalizando 12.6V. Essa alimentação é enviada para uma Ponte H dupla (modelo L298N). Esta ponte H tem um regulador de tensão de 5V que é usado para alimentar o microcontrolador ESP32, os motores são conectados diretamente na ponte H, e os demais componentes elétricos (Sensor ultrassônico e giroscópio) são alimentados pelo ESP32, pelos pinos 3.3V e GND. Além disso, o ESP32 usa 4 pinos com PWM para controlar as duas direções dos dois motores, 2 pinos para ler o giroscópio usando I2C e 2 pinos para ler o sonar, uma para o gatilho e o outro para o eco. Uma imagem feita no programa Fritzing, representando todas as ligações, pode ser vista na figura 9.

**Figura 9 - Conexões Elétricas e Componentes do Robô**

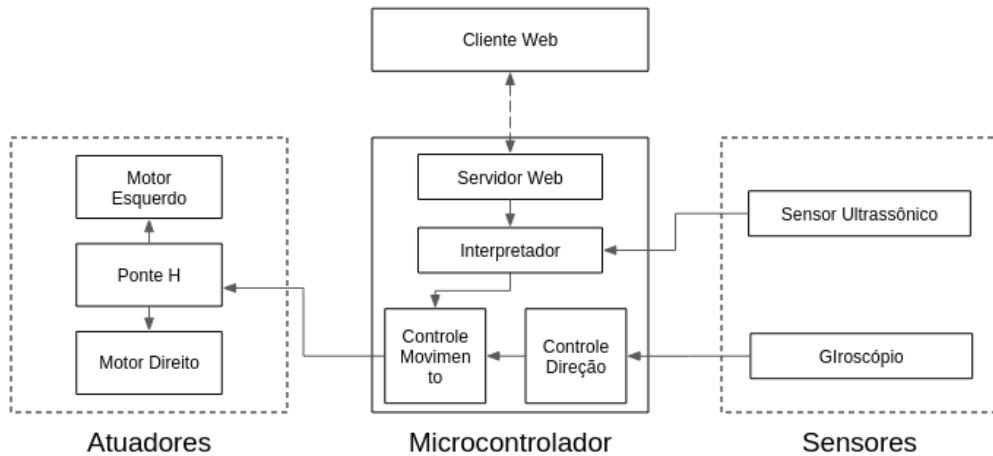


fritzing

### 3.4. Software

O código do robô foi feito em C++, e foi organizado em módulos. O arquivo principal do código *programable\_robot.ino* usa a macro `#include` para juntar os demais módulos: *server.h*, *interpreter.h*, *gyroscope.h*, *control.h* e *page.h*. O código completo pode ser visto no gitlab [<https://gitlab.com/simoesusp/robo-programavel-em-blocos>].

**Figura 10 - Organização dos Módulos de Software do Robô**



Fonte: Ilustração do autor desta monografia

Na figura 10 vê-se como as diferentes partes do software do robô se relacionam entre si e com os outros componentes eletrônicos. Ao iniciar, o microcontrolador gera uma rede de Wi-Fi e um **Servidor Web**, desta forma um dispositivo externo pode se conectar e acessar dois endpoints. Em um deles o robô envia uma página web com interface gráfica de programação do robô, e no outro o cliente faz uma requisição POST enviando o código a ser executado. Paralelamente a isso, o robô executa o **controle de direção**, uma tarefa em outro núcleo do processador, que constantemente lê os valores do **giroscópio** e calcula para qual direção o robô está apontando. Quando o servidor recebe o código do dispositivo externo, ele inicia o **interpretador** e muda de estado para ocupado para não receber novas requisições até a execução acabar. Por sua vez, o **interpretador**, dependendo da instrução que estiver executando, pode ler o **sensor ultrassônico** ou chamar uma das funções do **controle de movimento** que regula os **motores**, acionados por meio da **Ponte H**. Ele é um controle PID que regula a velocidade dos motores baseando-se na direção que o robô está apontando.

### **3.4.1. Inicialização**

Na inicialização, a ESP32, primeiramente configura os pinos que serão usados como entrada e saída. Em seguida inicia a tarefa do controle do giroscópio que será explicada em detalhes na seção 3.4.4. Para gerar a rede Wi-Fi foi usada a função softAP da biblioteca WiFi da ESP32 (ESPRESSIF SYSTEMS, 2022b), que recebe como parâmetro um nome para o SSID e a senha. Neste trabalho foi usado o ssid “robo” e a senha “senha123”. Também foi inicializado o módulo ArduinoOTA (ESPRESSIF SYSTEMS, 2022a) que permite que o microcontrolador seja programado via WiFi, facilitando assim a realização de mudanças no software se necessário.

### **3.4.2. Servidor Web**

O Servidor Web foi implementado usando a biblioteca esp\_http\_server (ESPRESSIF SYSTEMS, 2016b), usando as configurações padrões de httpd. Para o funcionamento normal dois endpoints foram criados: O endereço “/” que recebe requisições GET e contém a página da interface de programação web; e o endereço “/code” que recebe requisições POST contendo os programas a serem executados. O corpo da requisição deve ser um vetor de bytes, na qual o primeiro byte é o tamanho N do programa e os seguintes N bytes são o programa. Essa requisição vai inicializar o interpretador e retornar a mensagem “ok”. Também foi implementado um terceiro endpoint com uri “/info” que foi usado para os testes e para debugar o robô.

### **3.4.3. Interpretador**

O interpretador recebe o programa a ser executado do Servidor Web, em formato de um vetor de bytes. Ele lê cada byte, salvando a posição atual da leitura na variável **pc** (program counter). A cada iteração o interpretador vai ler a instrução na posição **pc**, realizar as ações correspondentes e atualizar o valor de **pc**. Cada instrução exige uma quantidade de parâmetros constantes e variáveis. Parâmetros constantes ficam no código junto com as instruções, já os os parâmetros variáveis são recuperados do topo de uma pilha, o que faz eles funcionarem como notação pós fixa. Todos os parâmetros tem um 1 byte de tamanho, isso é, são inteiros de 0 a 255, mas que também podem ser interpretados como valores booleanos, nos quais 0 representa ‘falso’ e os demais números representam verdadeiro, ou podem ser endereços de memória indicando uma posição no código. Endereços de memória extraídos do

código são relativos à posição atual do **pc**, e os que vierem da pilha são absolutos. Sendo assim, cada instrução no código será composta por um byte contendo o número da instrução seguida pelos parâmetros constantes. Como o número de parâmetros é determinado pelo tipo instrução, ao ler o número da instrução o interpretador já sabe quantos parâmetros terão em seguida, e incrementa o PC para pular esses parâmetros e ler o próximo código de instrução.

Na tabela 2 vemos todas as instruções e seus parâmetros.

**Tabela 2 - Instruções e seus Parâmetros**

| Nome                | OP Code | Parâmetros Constantes          | Tipos parâmetros constantes  | Parâmetros Variáveis | Tipos parâmetros variáveis   |
|---------------------|---------|--------------------------------|------------------------------|----------------------|------------------------------|
| Repete              | 1       | Número de repetições           | Número                       |                      |                              |
|                     |         | Fim da cadeia                  | Endereço de memória relativo |                      |                              |
| Se                  | 2       | Fim da cadeia                  | Endereço de memória relativo | Condição             | Booleano                     |
| Enquanto            | 3       | Endereço do começo da condição | Endereço de memória relativo | Condição             | Booleano                     |
|                     |         | Fim do bloco                   | Endereço de memória relativo |                      |                              |
| Frente              | 4       | Duração                        | Número                       |                      |                              |
| Esquerda            | 5       |                                |                              |                      |                              |
| Direita             | 6       |                                |                              |                      |                              |
| Sensor Ultrassônico | 7       |                                |                              |                      |                              |
| Maior que           | 8       | Valor comparado                | Número                       | Valor comparativo    | Número                       |
| Menor que           | 9       | Valor comparado                | Número                       | Valor comparativo    | Número                       |
| Termina             | 254     |                                |                              |                      |                              |
| Fim de Cadeia       | 255     |                                |                              | Endereço de retorno  | Endereço de memória absoluto |

- **Repete:** Adiciona na pilha o endereço absoluto seguinte ao fim do bloco, em seguida adiciona N vezes o endereço absoluto do começo da cadeia, que é a próxima instrução. Assim, das primeiras N vezes que ele executar a instrução fim de cadeia ele vai retornar para o começo da cadeia, mas na enésima primeira ele vai para a primeira posição depois do fim da cadeia. Essa implementação não é eficiente no uso da pilha, mas foi escolhida pois não necessita que a instrução seja dividida em duas.
- **Se:** Lê o valor da condição no topo da pilha, se ele for zero pula o **pc** para a posição seguinte ao fim da cadeia, se não, aciona a posição seguinte ao fim do bloco na pilha e continua para a próxima posição. Como a instrução tenta ler um valor booleano, é esperado que ela venha em seguida de uma instrução que retorne um valor booleano, como ‘maior que’ ou ‘menor que’, porém o interpretador não faz essa verificação.
- **Enquanto:** Lê o valor da condição no topo da pilha, se ela for verdadeira (diferente de zero) ele adiciona o valor do começo das condições na pilha. Dessa forma, quando chegar no próxima instrução fim de cadeia vai voltar para o começo da condição e calculá-la novamente. Se o valor for falso (igual a zero), vai pular para o endereço seguinte do fim do bloco. Dessa forma vai ficar repetindo a cadeia interna enquanto a condição for verdadeira.
- **Frente:** Chama a função frente do controle de movimentos, que fará o robô andar em linha reta durante a quantidade de centésimos de segundos passada como parâmetro.
- **Esquerda:** Chama a função ‘esquerda’ do controle de movimentos, que fará o robô girar 90º em seu próprio eixo em sentido anti-horário.
- **Direita:** Chama a função ‘direita’ do controle de movimentos, que fará o robô girar 90º em seu próprio eixo em sentido horário.
- **Sensor Ultrassônico:** Adiciona na fila a distância em centímetros que o sensor ultrassônico mediu até um obstáculo.
- **Maior Que:** Lê um valor comparativo do topo da pilha se ele for maior que o valor comparado adiciona verdadeiro no topo da pilha, se não, adiciona falso.
- **Menor Que:** Lê um valor comparativo do topo da pilha se ele for menor que o valor comparado adiciona verdadeiro no topo da pilha, se não, adiciona falso.
- **Termina:** Encerra a execução do código.
- **Fim de Cadeia:** Lê o endereço de retorno no topo da pilha e direciona o **pc** para lá.

### 3.4.4. Controle de Direção

Manter o controle de quantos graus o robô gira em cada mudança de direção não é uma tarefa fácil. Existem muitos fatores que interferem no movimento quando se manda que o robô vire por um certo tempo. As rodas podem derrapar, com a bateria gasta a velocidade da curva pode mudar, ou até mesmo colisões com pequenos objetos podem interferir na manobra. Isso faz com que seja recomendável se utilizar um giroscópio para medir a quantidade de graus de cada giro. O giroscópio escolhido foi o modelo MPU-6050, que inclui 3 eixos de giroscópios e 3 eixos de acelerômetros, porém neste trabalho só o eixo z do giroscópio foi utilizado. O giroscópio foi fixado na horizontal de maneira que o eixo z fosse o eixo de rotação do robô ao fazer as curvas. Para se comunicar com o giroscópio se usa a interface I2C (Inter Integrated Circuit), que pode ser usada com a biblioteca Wire da ESP32, para tal é necessário configurar um pino de SDA (Serial Data) e um pino de SCL (Serial Clock). Escolheu-se usar um clock de 1MHz ao invés do padrão de 100KHz para diminuir a latência das leituras. O giroscópio é configurado para trabalhar com um escopo de - 250°/s até +250°/s.

Para manter a informação da direção do robô atualizada foi necessário criar uma tarefa específica com este propósito, foi usado a função xTaskCreatePinnedToCore do FreeRTOS, o sistema operacional que roda na ESP32. A tarefa começa fazendo 1.000 leituras de velocidade de rotação no eixo Z e pega uma média para saber se tem algum viés de erro na medição. Como o robô se encontra parado, todas as leituras deveriam ser zero. Este valor será salvo e subtraído de todas as futuras leituras como uma correção do erro. Após calcular a média do erro entra-se em um ciclo infinito de ler o tempo atual e a velocidade angular no eixo z e calcular o valor atual direção com a seguinte fórmula:

$$\Theta = \Theta' + \frac{\omega + \omega'}{2} (t - t')$$

Sendo que  $\Theta$  é a rotação atual,  $\Theta'$  é a rotação na iteração passada,  $\omega$  é a leitura atual da velocidade de rotação,  $\omega'$  leitura anterior,  $t$  é o tempo atual e  $t'$  é o tempo da iteração passada. Como o valor de  $\Theta$  é salvo na variável global rot\_z, o controle de movimento pode acessá-lo a qualquer momento. Quando o interpretador vai começar a ler um código novo, ele zera a variável global para que a direção que o robô esteja virado naquele momento seja considerada a frente.

### 3.4.4. Controle de Movimento

O controle de movimento mantém salvo quantos graus o robô deveria ter girado desde o começo da execução do código. Por convenção, usa-se a direção no sentido horário. Sendo assim, esse valor começa como zero e é subtraído 90 a cada curva para esquerda e somado 90 a cada curva para direita. A diferença entre esse valor da direção desejada e a direção calculada pelo controle de direção é o erro da direção  $e(t)$ , que será usado para ajustar as velocidades dos motores.

Os motores estão conectados na Ponte-H: o motor direito está na saída Out1 e Out2 da ponte-H e o motor esquerdo nas saídas Out3 e Out4. A ponte-H permite que os motores liguem nos dois sentidos, baseado no sinal que estiver chegando pelos pinos de entrada (In1, In2, In3 e In4) de acordo com a Tabela 3 (STMICROELETRONICS, 2000).

**Tabela 3 - Ativações dos Motores Pela Ponte-H**

| In1  | In2  | Out1 | Out2 | Motor Direito | In3  | In4  | Out3 | Out4 | Motor Esquerdo |
|------|------|------|------|---------------|------|------|------|------|----------------|
| Gnd  | Gnd  | Gnd  | Gnd  | Parado        | Gnd  | Gnd  | Gnd  | Gnd  | Parado         |
| Gnd  | 3.3V | 12V  | Gnd  | Sentido 1     | Gnd  | 3.3V | 12V  | Gnd  | Sentido 1      |
| 3.3V | Gnd  | Gnd  | 12V  | Sentido 2     | 3.3V | Gnd  | Gnd  | 12V  | Sentido 2      |
| 3.3V | 3.3V | 12V  | 12V  | Parado        | 3.3V | 3.3V | 12V  | 12V  | Parado         |

Os motores foram conectados de tal forma que quando o motor direito estiver ligado no sentido 1 e o esquerdo no sentido 2, os motores girem fazendo o robô ir para frente, e quando os dois motores estiverem no mesmo sentido o robô gire em torno do próprio eixo. Como os pinos que conectam o ESP32 na ponte-H tem PWM (Pulse-Width Modulation) pode-se usá-lo para controlar o torque de cada motor. Isso acontece em duas situações, uma quando o robô está indo para frente e deve manter uma relação entre a velocidade das duas rodas de tal forma que o robô siga em linha reta, e outra na qual o robô está girando em torno do próprio eixo e as rodas estão girando em direções opostas. No segundo caso o controlador faz com que o curva pare com o robô o mais próximo possível da posição desejada.

No primeiro caso, os dois motores são inicialmente considerados uma PWM base de 150, então calcula-se um fator de correção para ajustar a rota, e ajusta a velocidade do motor esquerdo para ser base somada do fator de correção (FC), e o motor direito para ser a base

subtraída do fator de correção. Para calcular o fator de correção usa-se um PID com as constantes KP, KI, e KD sendo respectivamente 1 0,002 e 15. No final de cada iteração, espera-se um tempo de 5 milissegundos para iniciar a próxima.

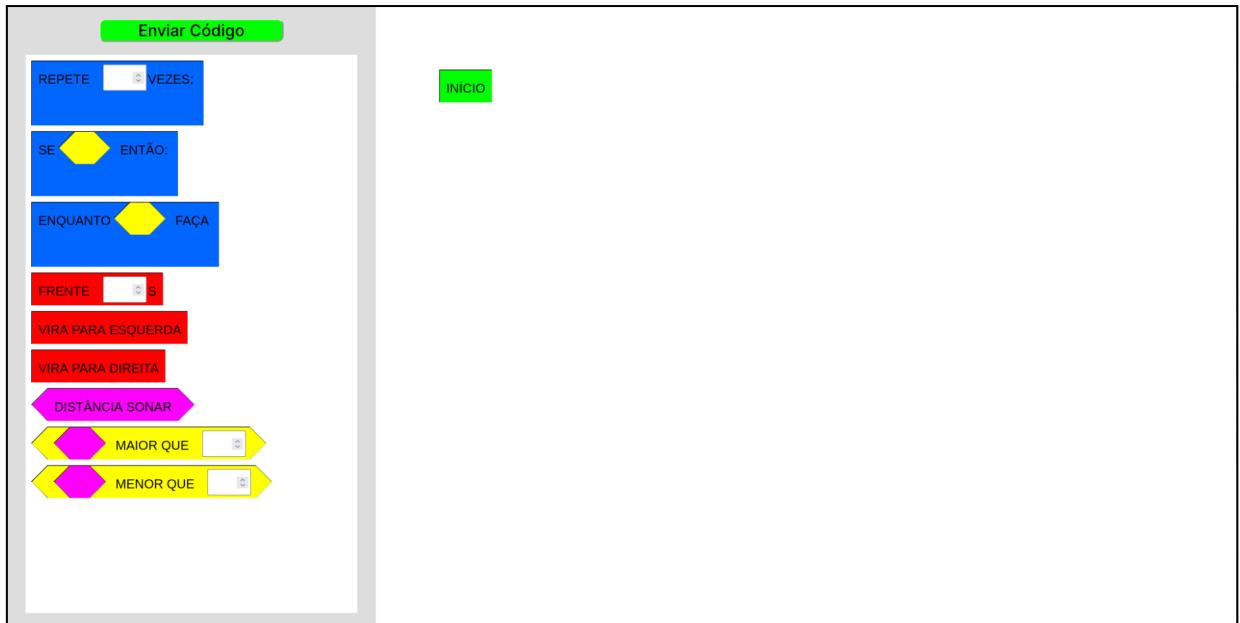
Já nas curvas os dois motores são ligados com o mesmo PWM até a direção calculada passar da direção desejada. Para que o robô não passe muito da direção desejada o controle impede que a velocidade angular fique muito grande, reduzindo o PWM baseado na derivada do erro. Para isso usa-se a seguinte fórmula com o KD valendo 100 :

$$Pwm = 150 + KD(e(t) - e(t - 1))$$

### 3.5. Interface de Programação

A interface de programação é uma página Web que o robô envia aos dispositivos conectados para que eles possam programá-lo. O protocolo Web foi escolhido pois ele é um padrão bastante consolidado, que permite transmitir páginas com interfaces gráficas e que executem código do lado do cliente. Esta página é composta por um único arquivo HTML contendo partes em CSS e Javascript. Visualmente a página é dividida em duas áreas: o menu à esquerda que ocupa 30% da página, e a área de programação à direita que ocupa os 70% restantes. No topo do menu tem um botão que envia o código para o robô, em baixo do botão tem um exemplo de cada bloco que pode ser arrastado para a área de programação. Inicialmente, na área de programação tem apenas um bloco de início, porém mais blocos podem ser arrastados para ela e encaixados para formarem o programa. Na figura 11 pode-se ver a interface de programação vazia, e na figura 12 com um programa.

**Figura 11 - A Interface de Programação Vazia**



**Figura 12 - A Interface de Programação com um Programa**



### 3.5.1. Padrões Visuais

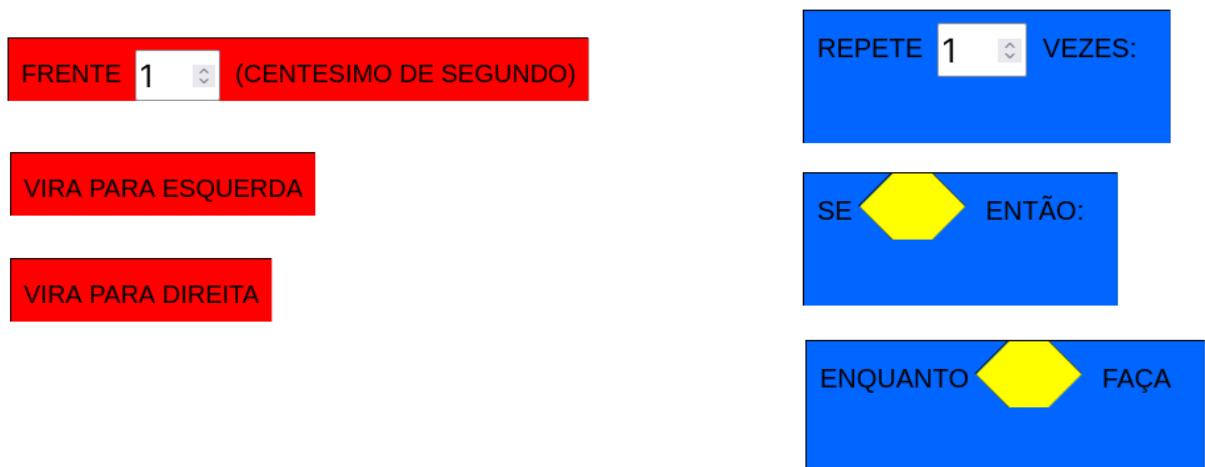
A interface tem os mesmos tipos de blocos que o Scratch, porém as formas de cada bloco são distintas. Neste trabalho foram implementadas formas mais simples, com menos saliências e cavidades.

Os padrões visuais são:

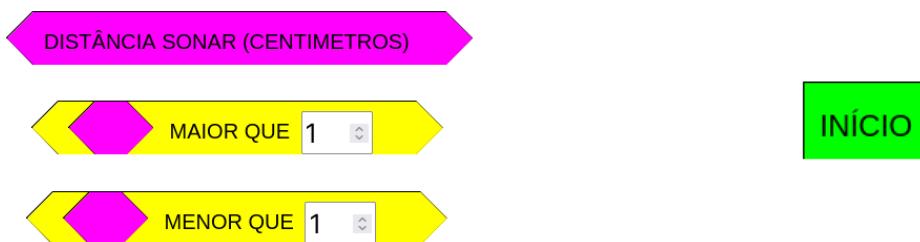
- Os blocos de comandos são retangulares e vermelhos (figura 13).

- Os blocos de controle são azuis e retangulares e são mais altos que os blocos de comando (figura 13).
- Os blocos de expressão são todos hexagonais e o tipo é determinado pela cor do bloco. Os amarelos são booleanos e os rosas são numéricicos. Os espaços onde eles podem se encaixar também seguem esse padrão de cor. (figura 14)
- O único bloco de gatilho é o início, que é igual a um bloco de comando, porém é verde. (figura 14)

**Figura 13 - Blocos de Comando e Blocos de Controle**



**Figura 14 - Blocos de Expressão e Blocos de Gatilho**



### 3.5.2. Gramática de Linguagem

A gramática da linguagem é definida a partir de um texto para cada bloco. Essa representação é que descreve como cada bloco é desenhado e como eles se encaixam. Esses textos usam palavras reservadas que representam características especiais do bloco. Para desenhar os blocos, o programa escreve esses textos substituindo as palavras reservadas. As palavras reservadas são:

- |**num**| significa que naquela posição o bloco espera é uma constante numérica.
- |**cmds**| significa que naquela posição o bloco espera uma cadeia de comandos.
- |**exp0**| significa que naquela posição o bloco espera uma expressão booleana.
- |**exp1**| significa que naquela posição o bloco espera uma expressão numérica.

Os textos dos blocos são:

- INÍCIO
- REPETE num VEZES: cmds
- SE exp0 ENTÃO: cmds
- ENQUANTO exp0 FAÇA cmds
- FRENTE num (CENTÉSIMOS DE SEGUNDOS)
- VIRA PARA ESQUERDA
- VIRA PARA DIREITA
- DISTÂNCIA SONAR (CENTÍMETROS)
- exp1 MAIOR QUE num
- exp1 MENOR QUE num

Com esses textos podemos definir a gramática da linguagem, o símbolo inicial é **pgr** que representa um programa completo.

**pgr** → INÍCIO cmds  
**cmds** → cmd cmds | cmd  
**cmd** → FRENTE num (CENTÉSIMOS DE SEGUNDOS) | VIRA PARA ESQUERDA | VIRA PARA DIREITA | SE exp0 ENTÃO: cmds | REPETE num VEZES: cmds | ENQUANTO exp0 FAÇA cmds

**exp0** → **exp1** MAIOR QUE **num** | **exp1** MENOR QUE **num**

**exp1** → DISTÂNCIA SONAR (CENTÍMETROS)

**núm** → Inteiros de 0 a 255

# CAPÍTULO 4: RESULTADOS EXPERIMENTAIS

## 4.1. Tempo de Envio do Código

Para comparar o tempo de envio do entre a interface de programação proposta e a porta de rede da IDE do Arduino, foram criados três programas de tamanhos distintos, cada um com duas versões: uma programada em blocos na interface de programação e outra programada na IDE do Arduino. As duas versões de cada código foram enviadas para o robô 2 vezes. As versões do Arduino incluíam além dos comandos a serem realizados, as bibliotecas Arduino OTA e WiFi para que o robô pudesse ser programado novamente depois de terminar a execução. Os tempos de envio da IDE do Arduino foram medidas com cronômetro e incluem o tempo de compilação e o tempo de envio. Porém não foi possível medir os tempos da interface de blocos com um cronômetro, pois eles eram muito pequenos. Então foi adicionado um contador do tempo na própria interface de programação. Os resultados obtidos podem ser vistos na tabela 4.

**Tabela 4 - Tempo de Envio nos Experimentos**

| Programa | Número de Blocos | Tempo Arduino IDE | Tempo Blocos |
|----------|------------------|-------------------|--------------|
| 1        | 1                | 41 s              | 19 ms        |
| 1        | 1                | 41 s              | 23 ms        |
| 2        | 7                | 41 s              | 47 ms        |
| 2        | 7                | 40 s              | 54 ms        |
| 3        | 11               | 38 s              | 31 ms        |
| 3        | 11               | 41 s              | 23 ms        |

Além do tempo, podemos medir o tamanho dos arquivos enviados. Para programar o ESP32 a IDE do Arduino envia dois conjuntos de dados diferentes, um que vai ser escrito na memória de programa do microcontrolador, e ou que consiste nas variáveis globais que se destinam à memória dinâmica. Já a interface em blocos envia um bloco único. O tamanho dos arquivos pode ser visto na tabela 5.

**Tabela 5 - Tamanho dos Arquivos Enviados nos Experimentos**

| Programa | Número de Blocos | Arduino IDE Memória de Programa (B) | Arduino IDE Memória Dinâmica (B) | Arduino IDE Memória Total (B) | Código em Blocos (B) |
|----------|------------------|-------------------------------------|----------------------------------|-------------------------------|----------------------|
| 1        | 1                | 803 721                             | 48 360                           | 852 081                       | 4                    |
| 2        | 7                | 804 057                             | 48 360                           | 852 417                       | 20                   |
| 3        | 11               | 804 337                             | 48 360                           | 852 697                       | 24                   |

Também foi realizado um teste sem as bibliotecas de Wi-Fi e Arduino OTA, porém sem elas é necessário usar o cabo USB na próxima vez que for programar o robô. O tempo ficou de 14 segundos e o tamanho dos arquivos 275.057 da memória de programa e 22.700 da memória dinâmica.

Pelos resultados obtidos pode-se ver que o tamanho dos programas foi pouco significativo para o resultado final, sendo que as vezes programas maiores foram mais rápidos que programas menores. Também pode-se ver que 65% do tamanho do arquivo enviado na IDE do Arduino são as bibliotecas de Wi-Fi e Arduino OTA, que não mudam de um envio para o outro. Sem elas o tempo de envio também diminui por volta de 65%. Como era esperado, com o interpretador no robô, enviando apenas as instruções, o tempo reduz várias ordens de magnitude, ficando imperceptível para o usuário.

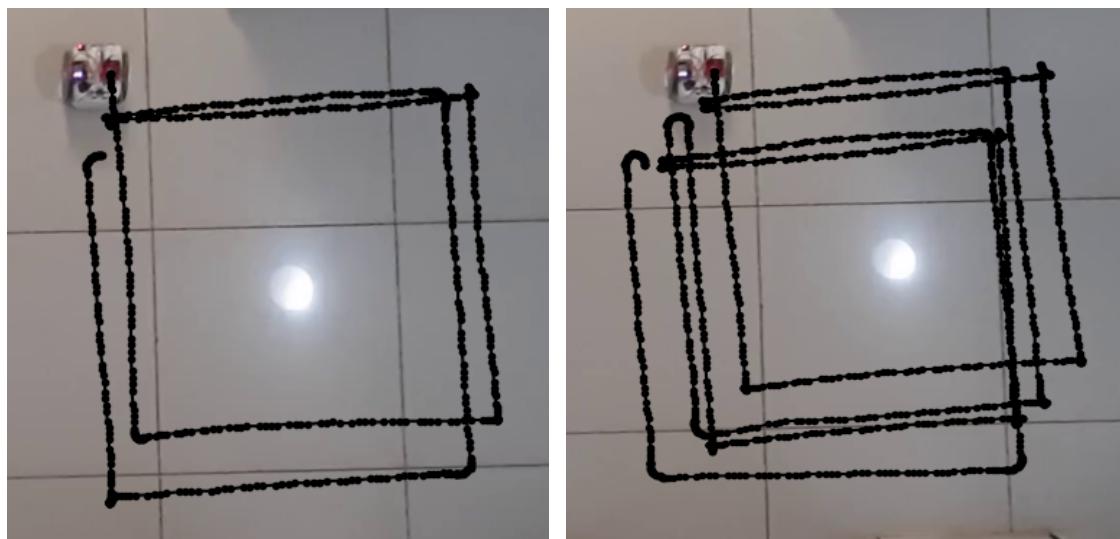
## 4.2. Trajetória do Robô

No segundo experimento, um vídeo foi filmado do robô executando diferentes programas, a câmera estava fixa em cima do robô, tendo assim uma visão superior. A cada quadro desse vídeo foi registrada a posição dos pixels do led do ESP32 e do led da Ponte-H. Para marcar as posições foi utilizado um programa em python que usava os canais de matiz, saturação e valor, para encontrar os pontos mais brilhantes da imagem. Nas figuras 15 e 16 está marcada trajetória do Led da ponte-H sobreposta com o primeiro quadro. Na figura 15 foi mantida a imagem inteira do vídeo, na figura 16 recortou-se apenas a parte do trajeto, para ser visto com mais clareza. Os vídeos completos dos experimentos, arquivos contendo a posição dos leds e o programa para encontrar os leds estão disponíveis no gitlab [<https://gitlab.com/simoesusp/robo-programavel-em-blocos>].

**Figura 15 - Trajeto Desviando de Obstáculo com Sonar**



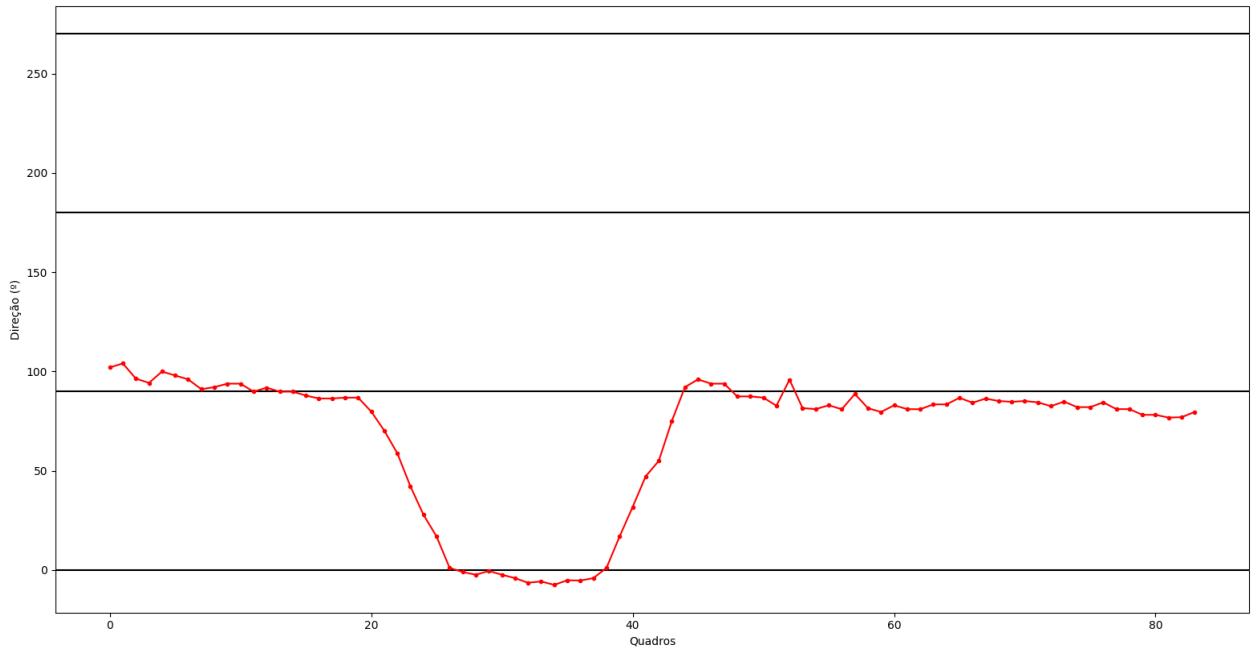
**Figura 16 - Trajeto Quadrado Ide a Volta e Trajeto Quadrado Ida e Volta com Duas Repetições**



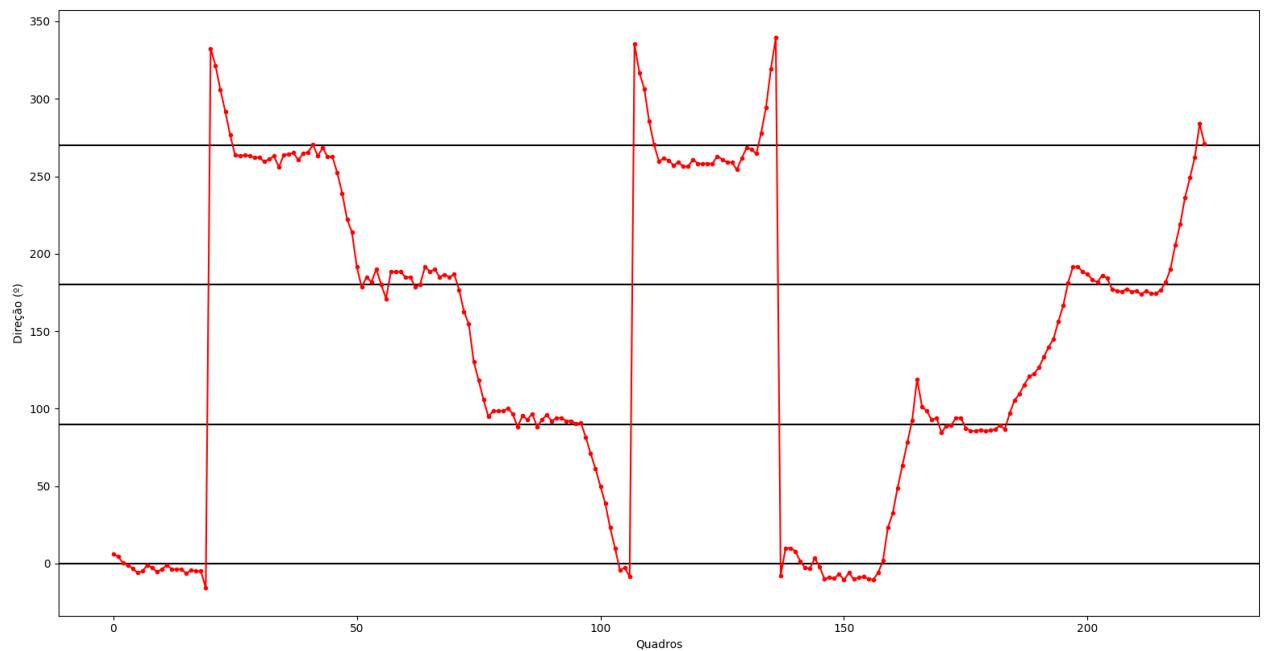
Pelas imagens vê-se que as formas programadas são reconhecíveis na trajetória do robô e as curvas formam ângulos retos. Porém a distância percorrida nas retas varia e os quadrados vão ficando cada vez mais deslocados da posição original. A partir dos vídeos,

usando a posição dos dois leds, também foi possível medir a direção do robô. Nos gráficos 1, 2 e 3 pode-se ver a direção do robô em cada trajetória a cada quadro. Os gráficos estão mostrando ângulos de  $-20^\circ$  e  $240^\circ$ , e os ângulos de 0, 90, 180, 270 foram destacados com linhas vermelhas. Foi considerado que o ângulo entre a linha que passa pelos leds e a frente do robô é de  $80^\circ$ .

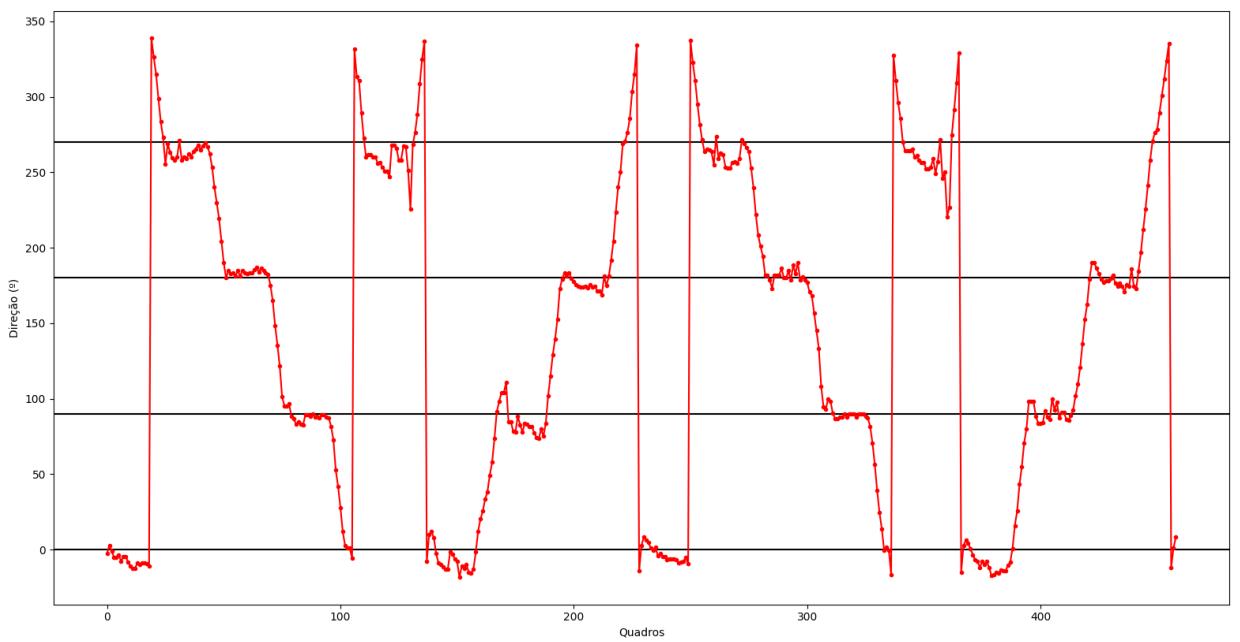
**Gráfico 1 - Direção do Robô de Obstáculo com Sonar em Graus**



**Gráfico 2 - Direção do Robô no Trajeto Quadrado Ida e Volta**



**Gráfico 3 - Direção do Robô no Trajeto Quadrado Ida e Volta com Duas Repetições**



Nesses gráficos pode-se ver que em algumas curvas o robô ultrapassa a direção desejada e que posteriormente corrige enquanto está andando para frente.

### 4.3. Considerações Finais

Analizando os resultados obtidos, pode-se considerar que o trabalho apresenta uma opção viável para ser usado na robótica educacional. Fica demonstrado que tendo um interpretador de código dentro do robô pode-se enviar arquivos muito menores. O tamanho dos arquivos enviados reduz da ordem das centenas de milhares de bytes para as dezenas. Com isso, o tempo de envio fica sempre inferior a algumas centenas de segundos. Porém, deve-se ressaltar que dessa maneira o robô só pode realizar atividades bem restritas, limitando-se a um conjunto pequeno de instruções que podem ser programadas pelos blocos.

Já quanto à utilização do giroscópio, vê-se pela sobreposição do gráfico nas imagens que é possível reconhecer as formas programadas nos trajetos realizados, porém elas ainda poderiam melhorar consideravelmente. Tem um acúmulo de erro considerável durante o percurso, como pode ser visto no *Trajeto Quadrado Ida e Volta duas Vezes*, no qual a cada quadradão o robô está mais deslocado para frente e para direita. A diferença entre a posição inicial do led da ponte-H é de 123 pixels, o que, comparando com outras distâncias, estimou-se equivalente a 27cm.

# CAPÍTULO 5: CONCLUSÃO

## 5.1. Contribuições

Neste trabalho demonstrou-se que implementar um interpretador e hospedar uma interface de programação WEB dentro do robô é uma opção bastante interessante para Kits Robóticos Educativos, permitindo que o robô possa ser controlado a partir de um navegador Web sem a necessidade de instalar Softwares adicionais, e praticamente zerando o tempo de envio de código para o robô. Além disso, fica demonstrado que esta solução não está além dos limites de hardware da atualidade, pois ela foi implementada usando um microcontrolador ESP32, que pode ser facilmente comprado e programado.

O código desenvolvido está disponível no Gitlab e pode ser livremente usado por qualquer pessoa. Porém, para este trabalho foi desenvolvido apenas uma versão mínima da solução que permitisse verificar seu funcionamento. Isso é principalmente notável no conjunto de blocos implementados. Entre as comparações só foram implementados blocos para as operações MAIOR-QUE e MENOR-QUE, mas não um para IGUAL. Também se pode notar a falta de um bloco SE-ENTÃO-SENÃO. Porém, a implementação desses blocos pode ser realizada sem maiores dificuldades a partir das técnicas discutidas neste trabalho. Além disso, encontram-se no mercado uma grande variedade de sensores e atuadores que podem ser usados com o Arduino e o ESP32, que poderiam ser adicionados no robô, como sensor de luminosidade, sensor de temperatura, buzinas etc. Uma vez adicionados esses componentes ao robô, seria simples criar novos blocos e instruções no interpretador para controlá-los. Além disso, seria bom implementar mais recursos como uma opção de apagar blocos da área de programação, salvar os códigos desenvolvidos e de consultar as medidas dos sensores. Para poder abranger todos esses objetivos, este projeto foi disponibilizado como software e hardware livres nos repositórios Github e Gitlab para fomentar uma comunidade de contribuidores que possam utilizar e melhorar as ferramentas produzidas.

## 5.2. Considerações Sobre o Curso de Graduação

Várias disciplinas do curso estão diretamente relacionadas com este trabalho. A elaboração de circuitos elétricos e o uso de microcontroladores e componentes eletrônicos é estudada em SSC0180 - Eletrônica para Computação. O desenvolvimento de páginas Web é estudado em SCC0219 - Introdução ao Desenvolvimento Web. O funcionamento das

linguagens de máquina, que inspiraram o código lido pelo interpretador, é estudada em SSC0902 - Organização e Arquitetura de Computadores. Já as definições de gramáticas formais e compilação entre distintas representações do programa é estudada nas disciplinas SCC0205 - Teoria da Computação e Linguagens Formais e SCC0217 - Linguagens de Programação e Compiladores. Sendo assim, pode-se dizer que o curso ofereceu conhecimentos bastante amplos sobre computação, o que permitiu encontrar soluções para problemas bastante diversos. Um exemplo desta versatilidade aplicado indiretamente neste trabalho foi o uso dos conhecimentos obtidos na disciplina *SCC0251 - Processamento de Imagens* para fazer um programa que encontra a posição dos leds em cada quadro dos vídeos dos experimentos.

### 5.3. Trabalhos Futuros

O projeto desenvolvido neste trabalho pode ser expandido em diversos aspectos. É interessante implementar uma série de ferramentas na interface de programação. Em seu estado atual, todos os códigos desenvolvidos se perdem ao recarregar a página, eles poderiam ser salvos em cookies para que isso não acontecesse. Também poderia-se ter a opção de baixar o programa e abrir os programas baixados. No estado atual também não é possível apagar um bloco da área de programação, outra mudança que poderia ser implementada.

Quanto aos blocos eles poderiam ser estendidos indefinidamente, podendo abranger qualquer sensor ou atuador compatível com o microcontrolador. Porém, pensando nas limitações da linguagem seria oportuno desenvolver opções de usar números não inteiros e maiores que 255, pois estes se mostraram muito restritos nos testes dos programas desenvolvidos. Também seria bom implementar variáveis e funções na linguagem, pois estas são partes importantes da programação que envolve o pensamento abstrato. Com elas seria possível realizar atividades mais complexas e oferecer um aprendizado mais completo sobre computação aos alunos que utilizarem o kit.

# Referências

ATMEL CORPORATION. **AVR221: Discrete PID controller**, 2006. Disponível em <[https://www.betzler.physik.uni-osnabrueck.de/Manuskripte/Elektronik-Praktikum/p3/doc255\\_8.pdf](https://www.betzler.physik.uni-osnabrueck.de/Manuskripte/Elektronik-Praktikum/p3/doc255_8.pdf)>. Acesso em 11 de Jun. de 2023.

AHO, Alfred V.; LAM, Monica S.; SETHI, Ravi; ULLMAN, Jeffrey D. **Compilers: Principles, Techniques, & Tools**. 2. ed. Pearson Addison Wesley. Boston, 2007.

ARDUINO COMPANY. **Arduino Uno Rev3**, 2021. Disponível em: <<https://store.arduino.cc/products/arduino-uno-rev3>>. Acesso em 11 de Jun. de 2023.

ARDUINO: entenda mais sobre essa versátil plataforma. **3E Unicamp**, 2021. Disponível em: <<https://3eunicamp.com/arduino-entenda-mais-sobre-essa-versatil-plataforma/>>. Acesso em 11 de Jun. de 2023.

ÅSTRÖM, Karl J.; HÄGGLUND, Tore. **PID Controllers: Theory, Design and Tuning**. Instrumentation Society of America. Durham, 1995.

ANWAR, Saira; BASCOU, Nicholas A.; MENESKE, Muhsin; KARDAGAR Asefah. **A Systematic Review of Studies on Educational Robotics**. Journal of Pre-College Engineering Education Research (J-PEER), 9(2), Article 2. 2019.

BEGEL, Andrew. **LogoBlocks: A Graphical Programming Language for Interacting with the World**. Epistemology and Learning Group MIT Media Laboratory. 1996

BELAND, Christopher; CHAN, Wesley; CLARKE, Dwayne; PARK, Richard; TRUPIANO, Michael. **LEGO Mindstorms: The Structure of an Engineering (R)evolution**. Massachusetts Institute of Technology, 2000.

BENCHOFF, Brian. **New Chip Alert: The ESP8266 WiFi Module (It's \$5)**. Hackaday, 2016. Disponível em: <<https://hackaday.com/2014/08/26/new-chip-alert-the-esp8266-wifi-module-its-5/>>. Acesso em 11 de Jun. 2023.

CURVELLO, André. **Programação do ESP8266 via OTA com wifi**. MakerHero, 2017. Disponível em: <<https://www.makerhero.com/blog/programacao-esp8266-ota-wifi/>>. Acesso em 11 de Jun. 2023.

DAVCEV, Kristijan; KOCESKA, Natasa; KOCESKI, Saso. **A Review of Robotic Kits Used for Education Purposes**. University of Navi Sad. Zrenjanin, 2019.

ESPRESSIF SYSTEMS. **Get Started**, 2016. Disponível em: <<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html>>. Acesso em 11 de Jun. 2023

ESPRESSIF SYSTEMS. **HTTP Server**, 2016. Disponível em: <[https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp\\_http\\_server.html](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_server.html)>. Acesso em 11 de Jun. 2023

ESPRESSIF SYSTEMS. **Memory Types**, 2016. Disponível em: <<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/memory-types.html>>. Acesso em 11 de Jun. 2023

ESPRESSIF SYSTEMS. **OTA Web Update**, 2022. Revisão 72c41d09. Disponível em:<[https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/ota\\_web\\_update.html](https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/ota_web_update.html)>. Acesso em 11 de Jun. 2023.

ESPRESSIF SYSTEMS. **Wi-Fi API**, 2022. Revisão 72c41d09. Disponível em:<<https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/api/wifi.html>>. Acesso em 11 de Jun. 2023.

HORN, Michael; BERS, Marina. **Tangible Computers**. In: FINCHER, Sally A; ROBINS, Anthony V. **The Cambridge Handbook of Computing Education Research**. Cambridge University Press. Cambridge, 2019.

JUNIOR, Luiz A; NETO, Osvaldo T; HERNANDEZ, Marli F; MARTINS, Paulo S; ROGER, Leonardo L; GUERRA, Fátima A; **A Low-Cost and Simple Arduino-Based Educational Robotics Kit**. Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Robotics and Control (JSRC), December Edition, 2013. Volume 3, Issue 12.

KAMPOURIDIS, Ilias. **Development and Design of Educational Robot Using ESP32 Microcontroller**. University of Macedonia, 2023.

LAMPRECHT, Patrick; HALLER-SEEBER, Simon; PIATER, Justus. **A Block-based IDE Extension for the ESP32**. In: LEPUSCHITZ, Wilfried; MERDAN, Munir; KOPPENSTEINER, Gottfried; BALOGH, Richard; OBDRZALEK, David (Eds.). **Robotics in Education - Methodologies and Technologies**. Springer. Cham, 2021.

MALMI, Lauri; UTTING, Ian; KO, Amy J. **Tools and Environments**. In: FINCHER, Sally A; ROBINS, Anthony V (Eds). **The Cambridge Handbook of Computing Education Research**. Cambridge University Press. Cambridge, 2019.

MALONEY, John; RESNICK, Mitchel; RUSK, Natalie; SILVERMAN, Brian; EASTMOND Evelyn. **The Scratch Programming Language and Environment**. ACM Transactions on Computing Education. Vol 10, Issue 4. Jun. 2010

MICRORED. **Microredutor Modelo A - 12V a 24V**. Disponível em : <<https://microred.com.br/page18.html>>. Acesso em 11 de Jun. 2023.

NOONE, Mark; MOONEY, Aidan; **Visual and Textual Programming Languages: A Systematic Review of the Literature**. Journal of Computers in Education. Vol. 5, Issue 2. Mar. 2018.

PAPADAKIS, Stamatios. **Robots and Robotics Kits for Early Childhood and First School Age**. University of Crete, 2020.

PAPERT, Seymour. **Mindstorms: Children, Computers and Powerful Ideas**. 2.ed. New York, 1993

PARALIJA, Kenan. **Making a 2WD Arduino Vehicle Drive Straight**. Hackerster.io, 2022. Disponível em <<https://www.hackster.io/Kenan-Paralija/making-a-2wd-arduino-vehicle-drive-straight-ae40e>>. Acesso em 11 de Jun. 2023.

PATIL; Dhiraj A.; AGIWAL Sakshi V. **Design and Implementation of Mapping Robot using Digital Magnetic Compass and Ultrasonic Sensor**. International Journal of Engineering Research & Technology (IJERT). Vol. 4, Issue 6. June, 2015.

SIMÕES, Eduardo. **Workshop de Robótica**. Gitlab, 2021. Disponível em : <<https://gitlab.com/simoesusp/workshop-robotica>>. Acesso em 11 de Jun. 2023.

SOLOMON, Cynthia; HARVEY, Brain; KAHN, Ken; LIEBERMAN, Henry; MILLER, Mark L.; MINSKY, Margaret; PAPERT, Artemis; SILVERMAN, Brian. **History of Logo**. Proceedings of the ACM on Programming Languages. Vol 4, Issue HOPL. 2020.

STMICROELETRONICS. **L298: DUAL FULL-BRIDGE DRIVER**, 2000. Disponível em : <<https://www.st.com/resource/en/datasheet/l298.pdf>>. Acesso em 11 de Jun. 2023.