UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

MANOEL MORAIS LEMOS NETO

PROPOSTA DE ATIVIDADES DIDÁTICAS EM MECATRÔNICA UTILIZANDO CONJUNTO EDUCACIONAL PARA ESCOLAS PÚBLICAS

Joinville
2023

MANOEL MORAIS LEMOS NETO

PROPOSTA DE ATIVIDADES DIDÁTICAS EM MECATRÔNICA UTILIZANDO CONJUNTO EDUCACIONAL PARA ESCOLAS PÚBLICAS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Dr. Anderson Wedderhoff Spengler

Joinville
2023

RESUMO

A programação e os sistemas mecatrônicos são duas das áreas que mais crescem no mercado mundial. No Brasil, a presença delas no ensino básico ainda não foi padronizada. No entanto, muitas escolas, majoritariamente de rede privada, já estão tentando integrá-las em conjunto com as matérias regulares. Nesse trabalho serão apresentadas as bases teóricas necessárias para a inicialização no mundo da robótica, bem como propostas de experimentos utilizando kits de baixo custo, viabilizando o investimento da rede pública de ensino. Os projetos apresentados abordarão conceitos como: temporização de processos utilizando *Light-Emiting Diodes* (LEDs), detecção de presença por meio de radiação infravermelha e controle de rotação via variação de tensão, uso e declaração de variáveis, uso de classes e objetos, inclusão de bibliotecas, estruturas de repetição e seleção, operadores entre outros aspectos necessários para a programação de microcontroladores. Com isso, foi possível criar um material que abrangesse toda a base introdutória da robótica de maneira simples, com baixo custo, que incentiva jovens a explorar e até mesmo seguir na área.

Palavras-chave: Sistemas mecatrônicos. Brasil. Ensino. Robótica. Programação. Kits didáticos. Microcontroladores.

ABSTRACT

Programming and mechatronic systems are two of the fastest-growing fields in the global market. In Brazil, their presence in basic education has not yet been standardized. However, many schools, mostly private ones, are already trying to integrate them together with regular subjects. This work will present the necessary theoretical foundations for getting started in the world of robotics, as well as proposals for experiments using low-cost kits, making it feasible for public education networks to invest. The presented projects will address concepts such as: process timing using Light-Emiting Diodes (LEDs), presence detection through infrared radiation, rotation control through voltage variation, variable usage and declaration, use of classes and objects, library inclusion, loop and selection structures, operators, and other aspects necessary for microcontroller programming. As a result, it was possible to create a material that covers the entire introductory basis of robotics in a simple and affordable way, encouraging young people to explore and even pursue a career in this field.

Keywords: Mechatronic systems. Brazil. Education. Robotics. Programming. Educational kits. Microcontrollers.

LISTA DE FIGURAS

Figura 1 – Ambiente de programação NTX-G 14
Figura 2 – Circuito elétrico simples. 16
Figura 3 – Arduino UNO R3 18
Figura 4 – Exemplos de resistores de filme metálico. 19
Figura 5 – Acendendo um LED. 19
Figura 6 – Potenciômetro linear. 20
Figura 7 – Servo motor. 21
Figura 8 – Buzzer ativo. 22
Figura 9 – Sensor de presença PIR. 23
Figura 10 – *breadboard*. 23
Figura 11 – Funcionamento de um botão. 24
Figura 12 – Usando um *pushbutton* no Arduino. 25
Figura 13 – Exemplo de fluxograma. 27
Figura 14 – Tabela-verdade do operador && (E lógico). 29
Figura 15 – Tabela-verdade do operador || (OU lógico). 30
Figura 16 – Operadores de igualdade e operadores relacionais. 31
Figura 17 – Operadores aritméticos. 32
Figura 18 – Uso inadequado de múltiplos returns. 35
Figura 19 – Selecionando placa e porta. 36
Figura 20 – Pisca. 41
Figura 21 – Semáforo. 42
Figura 22 – Exemplo de cruzamento. 43
Figura 23 – Estados para o cruzamento. 44
Figura 24 – Cruzamento com semáforo. 45
Figura 25 – Conectando um potenciômetro na protoboard. 46
Figura 26 – Cancela acionada via potenciômetro. 47
Figura 27 – Braço robótico controlado por potenciômetros. 50
Figura 28 – Braço montado e com os servos instalados. 51
Figura 29 – Alarme sonoro. 52

LISTA DE TABELAS

Tabela 1 – Tipos de dados comumente utilizados 29
Tabela 2 – Lista de Componentes. 39
Tabela 3 – Componentes para o blink. 40
Tabela 4 – Componentes para o semáforo. 41
Tabela 5 – Componentes para o cruzamento. 42
Tabela 6 – Cancela acionada por potenciômetro. 46
Tabela 7 – Braço robótico controlado por potenciômetros. 48
Tabela 8 – Lista de Componentes. 51

SUMÁRIO

1 INTRODUÇÃO 8
1.1 Objetivo 10
1.1.1 Objetivo Geral 10
1.1.2 Objetivos Específicos 10

2 FUNDAMENTAÇÃO TEÓRICA 11

3 METODOLOGIA 16
3.1 Materiais 17
3.1.1 Arduino UNO R3 17
3.1.2 Resistor 18
3.1.3 Diodo 19
3.1.4 Potenciômetro 20
3.1.5 Servo motor 21
3.1.6 Buzzer 21
3.1.7 Breadboard 23
3.1.8 Botões 24
3.2 Linguagem de programação C++ 25
3.3 Arduino 36

4 PROPOSTA DE EXPERIMENTOS E DISCUSSÃO 39
4.1 Blink 40
4.2 Semáforo 41
4.2.1 Semáforo para um cruzamento 42
4.3 Cancela acionada por potenciômetro 45
4.4 Braço robótico 48
4.5 Alarme 51
4.6 Avaliação de custos 54

5 CONCLUSÕES 55

REFERÊNCIAS 56

APÊNDICE A 59

APÊNDICE B 60

APÊNDICE C 61

APÊNDICE D **63**

APÊNDICE E **64**

APÊNDICE F **66**

1 INTRODUÇÃO

O Brasil é, em essência, um país agrário, e grande parte da contribuição do Produto Interno Bruto (PIB) é advinda desse setor. Tal fato decorre, em suma, da grande extensão territorial e da disponibilidade de recursos hídricos, o que favorece o criações de animais e cultivos em larga escala, cuja rentabilidade é alta ao ponto de negligenciarem o desenvolvimento de áreas como industria e comércio por muitos anos (CENTRO DE ESTUDOS AVANÇADOS EM ECONOMIA APLICADA, 2021). Entretanto, nos países desenvolvidos, adota-se uma postura econômica historicamente voltada para a indústria, desde a primeira revolução industrial e acompanhando as seguintes até a atual indústria 4.0.

Pode-se considerar que o Brasil encontra-se na terceira revolução industrial, um passo atrás das grandes potências mundiais, pois ainda é muito comum encontrar empresas que utilizam tecnologias desatualizadas em relação àquelas que usam ferramentas autônomas e com computação em nuvem (PORTAL DA INDÚSTRIA, 2022).

Considerando isso, Humberto Barbato, presidente da Associação Brasileira da Indústria Elétrica e Eletrônica (ABINEE) afirmou em uma entrevista: "A indústria é o setor que mais perde participação no PIB, então mudanças na Constituição, reformas tributárias e administrativas serão fundamentais para o país ter um crescimento efetivo.".O deputado Ricardo Barros, líder do governo na Câmara (2020), também se manifestou sobre atemática e mencionou que o governo está trabalhando na elaboração destas reformas, mas que o enfoque maior será na área administrativa (O QUE..., 2020).

Todavia, os esforços econômicos e administrativos, sozinhos, não são suficientes para favorecer o desenvolvimento industrial, pois ainda que o Brasil tenha mais de 11 milhões de desempregados, uma pesquisa de 2019 mostrou que 50% das empresas em solo nacional afirmam ter problemas com a falta de mão de obra qualificada (GERBELLI, 2020). Tal fato evidencia a necessidade de investimentos na capacitação profissional, além do que, quando se fala em modernização, é impossível não mencionar a robótica, tema que ainda não faz parte dos currículos de muitas escolas nacionais, principalmente os de escolas públicas.

Em contrapartida, hoje, existem projetos que aplicam a robótica no aprendizado escolar, como o *Robótica Livre*, que acontece nas escolas Estaduais no Rio Grande do Sul, visa formar grupos autossuficientes em pesquisas na área da robótica (NUCLEO DE TECNOLOGIA EDUCACIONAL DO RIO GRANDE DO SUL, 2014), uma iniciativa que proporciona aos jovens a oportunidade de desenvolver conhecimentos nas áreas associadas à robótica e à programação. Outro projeto que trabalha a temática da

robótica, desta vez sob um viés ambiental, é o *Robótica com Sucata*. Esse, por sua vez, desenvolve no público alvo uma consciência sustentável e ecológica para a construção de um futuro onde tecnologia e meio ambiente não sejam contrastes, mas que trabalhem juntos para o progresso (BRASIL, 2023).

Ainda não há obrigatoriedade da abordagem de tópicos relacionados a robótica em nenhum nos níveis de ensino da educação básica no Brasil, mas isso não impediu que muitas escolas públicas, como as que fazem parte do Projeto Robótica Livre, adquirissem kits didáticos para o ensino da robótica, como o pacote básico *microbit* da British Broadcasting Corporation (BBC) ou o kit LEGO® Mindstorm. Entretanto, nem todas tem condições para explorar didaticamente a usabilidade dos mesmos (MICRO:BIT EDUCATIONAL FOUNDATION, 2022).

A inclusão da robótica no ensino regular de maneira bem sucedida sem que haja padronização e suporte por parte do Ministério da Educação (MEC) demanda um grande engajamento de toda a instituição. Os principais obstáculos para isso são os custos dos equipamentos necessários, o longo tempo consumido pela natureza práticas das atividades e a dificuldade que os tópicos abordados possuem. (ALIMISIS, 2013). Então, é compreensível que não haja tantas instituições dispostas a adotar atividades voltadas para essa área.

As escolas que possuem robótica educacional como parte de seu currículo, costumam trabalhá-lo sob uma das três óticas (ALIMISIS, 2013):

- **Currículo por tema:** Esta abordagem consiste na seleção de temas específicos que possam ser trabalhados de maneira disciplinar ou interdisciplinar. O foco poderá ser na aprendizagem da robótica ou no uso da mesma para auxiliar no ensino de outras áreas do saber como, por exemplo, física e química.
- **Currículo por projeto:** Os estudantes trabalham em equipe no desenvolvimento de um projeto que tente solucionar problemas do mundo real.
- **Currículo por objetivo/competição:** Os estudantes desenvolvem suas atividades focados em participações em eventos e competições.

Nessa perspectiva, este trabalho apresenta a implementação da robótica educacional de um currículo por projeto, usando materiais de baixo custo. O enfoque será na criação de experimentos facilmente replicáveis e escaláveis, para que os professores interessados em repassar algum tema, de maneira mais prática, para seus alunos possam utilizá-los sem dificuldades.

Para isso, será abordada uma didática repleta de metáforas e exemplos. Isso porquê, busca-se neste material simplificar o entendimento, mediante uma comparações entre os aspectos construtivos e elementos do cotidiano. Desse modo, será possível construir uma ponte entre o conhecimento comum ao mecatrônico e o necessário para o entendimento dos componentes e sua utilização.

1.1 OBJETIVO

1.1.1 Objetivo Geral

Criar um conjunto de experimentos para a aprendizagem de robótica destinado à um público que não tenha conhecimentos aprofundados em eletrônica e programação, para que possam repassá-lo aos alunos, de escolas públicas.

1.1.2 Objetivos Específicos

- Apresentar propostas de aprimoramentos aos sistemas apresentados;
- Utilizar componentes de baixo custo para a confecção de um kit didático de robótica;
- Criar exemplos escaláveis, ou seja, cada experimento deverá ser capaz de receber melhorias por aqueles que desejam aplicar mais os conhecimentos obtidos;

2 FUNDAMENTAÇÃO TEÓRICA

O estudo da robótica consiste no desejo de sintetizar aspectos de seres vivos, como os humanos, a partir do uso de mecanismos, sensores, atuadores e computadores. Claro, não se limita apenas a isso e por essa razão recebe diversas ideias de áreas clássicas do conhecimento, como a física. A variedade de tópicos é tão vasta que, no geral, não é o caso de um único indivíduo ter domínio de toda a robótica, então é razoável dividi-la em quatro grandes áreas: manipulação mecânica, locomoção, visão computacional e inteligência artificial (CRAIG, 2012).

Conforme foi mencionado, é possível inferir que o conceito de robô também é muito amplo. Os leigos costumam defini-lo com base no que é mostrado na mídia, associando-os à categoria antropomórfica, que compreende os famosos braços robóticos e parte dos chamados humanoides. Segundo Craig:

> Um manipulador desse tipo consiste de duas juntas no “ombro” (uma para rotação em torno de um eixo vertical e uma de elevação para fora do plano horizontal), uma junta no “cotovelo” (cujo eixo é geralmente paralelo à junta de elevação do ombro) e duas ou três juntas no punho, na ponta do manipulador (CRAIG, 2012, p. 225).

Mas essas divergências de conceitos ocorriam até mesmo entre os cientistas. Em 1979, foi realizado um Workshop de pesquisa Robótica nos Estados Unidos, no qual foi definido robô como "Um manipulador reprogramável e multifuncional projetado para mover materiais, peças, ferramentas ou dispositivos especializados através de vários movimentos programados para o desempenho de uma variedade de tarefas." (ROBOT INSTITUTE OF AMERICA, 1979). Hoje, podemos encontrar definições mais amplas como a do dicionário online Priberam, que define robô como; "Aparelho capaz de agir de maneira automática numa dada função." (PRIBERAM, 2021), expandindo o conceito para aplicações mais voltadas para software.

No século XXI, é praticamente impossível dissociar a robótica da programação. A principio, a combinação dessas duas áreas consistia em um processo lento, tedioso e propenso a erros, pois utilizava-se a linguagem de máquina, muitas vezes referidas como **código-objeto**, que são basicamente cadeias de caracteres numéricos (em última instância reduzidas a 1s e 0s) que instruem os computadores a realizar suas operações mais elementares uma de cada vez. Além disso eram dependentes da máquina, ou seja, máquinas distintas possuíam codificações diferentes, ademais 0s e 1s não são muito intuitivas para seres humanos(DEITEL; DEITEL, 2011).

Com o passar dos anos foram surgindo as chamadas linguagens de alto nível, que utilizam uma sintaxe mais próxima da forma como os humanos se comunicam, facilitando imensamente o trabalho dos programadores. Dentre elas pode-se destacar o C++, que é amplamente utilizado na programação de microcontroladores como, por

exemplo, Arduino e Esp32. Essa conversão da linguagem de alto nível para a linguagem de baixo nível é feita por meio de compiladores.

Microcontroladores podem ser definidos como circuitos integrados que reúnem um núcleo de processador, memórias voláteis e não voláteis e diversos periféricos de entrada e de saída de dados, em outras palavras um computador, porém em uma escala menor e obviamente com poder de processamento inferior. Sua importância reside na sua alta demanda no mercado, principalmente por sistemas embarcados. Existe uma famosa analogia que os compara ao cérebro humano pois é o responsável pelo controle lógico do sistema (RAS UFCG, 2020).

Como pôde ser visto, tudo relacionado à robótica está em constante evolução, com o aprendizado não poderia ser diferente. Antes, a ideia de ensiná-la na educação básica era praticamente inexistente, pois os materiais requeridos eram muito caros, ademais a quantidade de profissionais capacitados para ministrar aulas de temas relacionados a essa área eram muito escassos. Hoje, tem-se salas de informática na maioria das escolas (VARELA, 2017), e, no contexto atual, os computadores representam o maior custo material para o ensino da robótica básica, visto que existem vários kits para iniciantes que utilizam como microcontrolador o Arduino.

Entretanto, trabalhar com esses kits requer um certo grau de conhecimento em eletrônica e programação. Logo, para utilizá-los seria necessário um investimento na contratação de professores exclusivos para ministrar essas aulas, além de reservar horários específicos para isso, o que para a maioria das escolas é inviável.

Pensando nesse mercado, a LEGO®, atualmente a maior empresa de brinquedos do mundo, em uma parceria com Massachusetts Institute of Technology (MIT), em 1998 lançou o LEGO Mindstorms Robotic Invention Kit. Composto por 717 peças, incluindo peças LEGO, motores, engrenagens, diferentes sensores e um RCX Brick que contém três portas de entrada e três portas de saída conectadas a um microcontrolador Hitachi H8/3292. O produto foi um sucesso e hoje suas versões mais novas Spike e EV3 são as mais utilizadas pelas escolas, no Brasil e no mundo (ÜÇGÜL, 2013).

Um dos maiores responsáveis pelo sucesso da linha foi o matemático Seymour Papert, que juntamente com sua equipe iniciaram uma pesquisa no MIT para tentar compreender como as crianças pensam e aprendem. O resultado dessa pesquisa foi a linguagem LOGO, que implementa de certa forma o construtivismo pela visão de Perpertque, co-fundador da linguagem juntamente com Cynthia Solomon e Wally Feurzeig (ÜÇGÜL, 2013).

A primeira versão do LOGO foi criada em 1967, ela era usada para se comunicar com a Tartaruga, nome dado a um robô com formato de cúpula do tamanho de uma bola de basquete que podia se mover pelo chão com os comandos do tipo FORWARD, BACKWARD, LEFT and RIGHT. Segundo Papert, a Tartaruga era vista como um

"objeto para pensar", pois permitia que as crianças aprendessem sobre programação de computadores de maneira prática e visual (PAPERT, 1980).

Por volta da metade da década de 1980, o grupo de desenvolvimento do LOGO iniciou sua colaboração com o grupo LEGO. Eles criaram o sistema LEGO/LOGO que combinava a técnina de produção da LEGO (que inclui engates, engrenagens e motores) com a linguagem LOGO. Na versão produzida pela LOGO em 1967, a Tartaruga já vinha pronta para uso, e as crianças só tinham que programá-las, já no sistema LEGO/LOGO elas podem criar suas próprias máquinas tais como uma roda gigante, elevador e robôs criatura, antes de os programar. Nomeado de “LEGO tc LOGO” pelo grupo LEGO, o kit já havia sido vendido pra mais de 15.000 escolas de nível fundamental e médio nos Estados Unidos já estavam utilizando os kits (ÜÇGÜL, 2013).

Mas o “LEGO tc LOGO” era limitado, os robôs construídos pelas crianças tinham que estar conectados a um computador por meio de fios, e ,em projetos que tinham mobilidade, os fios muitas vezes se embaraçavam e criavam nós. Além disso, o cabeamento tornava difícil de pensar no LEGO/LOGO como máquinas autônomas enquanto conectadas a um computador (DRUIN; HENDLER, 2000).

Fred Martin, membro do *MIT Media Laboratory* e seu grupo de pesquisa conseguiram solucionar essa limitação com o desenvolvimento do primeiro bloco programável em 1987. Aproximadamente do tamanho de uma caixa de suco, com processador baseado no Motorola 6811 e 32 kilobytes de memória de acesso aleatório não volátil, e tem uma ampla variedade de possibilidades de entrada-saída, sendo capaz de controlar quatro motores ou luzes de cada vez, e pode receber entradas de seis sensores. O bloco para ser programado deveria estar conectado ao computador, mas uma vez "baixado", o bloco poderia ser desconectado e carregado para qualquer lugar (RESNICK et al., 1996).

Após o uso em larga escala do bloco programável, fora do ambiente laboratorial, um grupo de escolas de Rhode Island criaram uma exibição chamada de "Robotic Park". O tema da exposição era animais, os estudantes então fizeram seus robôs criaturas, para isso, estudaram os aspectos biológicos e comportamentais dos animais escolhidos por seus grupos (equipes de três ou quatro alunos), dentre os projetos destacaram-se o robô caranguejo que possuía um par de pinças e começava a movê-las enquanto se dirigia a um alvo, outro destaque foi a tartaruga de lego que escondia a cabeça sempre que seu nariz tocava algo (RESNICK et al., 1996).

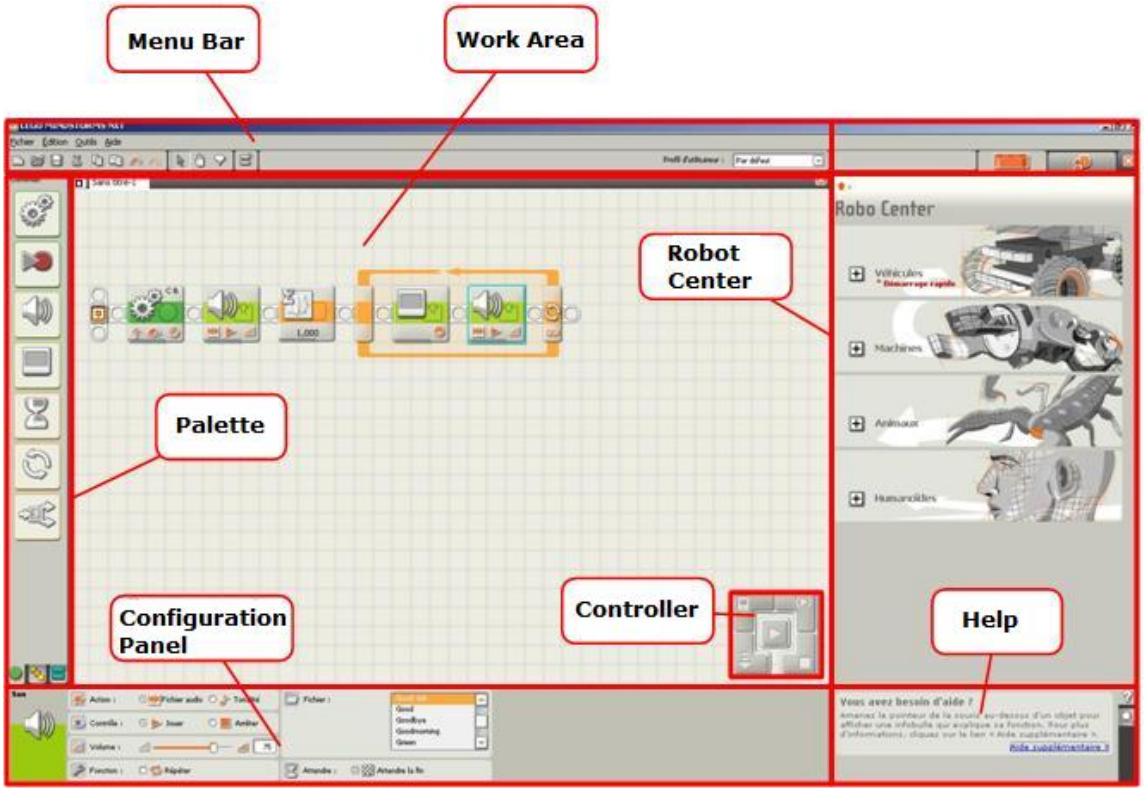
Livre da conexão direta com o computador, as possibilidades de pesquisa e aproveitamento dos sensores atingiram um novo patamar. Com isso, a popularidade do kit aumentou muito e o produto começou a se popularizar fora dos Estados Unidos. Mas, foi em 2006 que a internacionalização do produto se consolidou, com as atualizações dos componentes do kit e principalmente no bloco programável, o LEGO Mindstrom kit

foi substituído pelo LEGO Mindstorm NTX kit.

Bloco NXT, é um controlador multifuncional que se conecta-se facilmente com um computador gráfico. O processador principal do NXT é um processador Atmel® ARM® de 32 bits operando a 48 MHz, com 256 kB de memória flash e 64 kB de RAM; um coprocessador de 8 bits e 8 MHz fornece funcionalidade adicional. Possui interface de quatro botões e tela LCD de 100 x 64 pixels (26 x 40,6 mm). Isto pode se comunicar com um computador de mesa ou laptop com a porta USB 2.0 integral (12 Mbit/s) ou a porta sem fio Bluetooth, baseada no chip único CSR BlueCore™, 4 sensores (toque, ultrassônico, som e luminosidade). O kit ainda incluía o software NXT-G, uma interface gráfica para a programação do bloco que permitia a criação e download de programas para o NTX.

O ambiente de programação NTX-G (Figura 1) trabalha com o conceito de programação em bloco. Esse modelo de programação consiste na junção de blocos que executam funções específicas de modo a construir a lógica do programa, dentre os blocos, nele presentes, pode-se dividi-los em sete famílias principais, sendo elas: comuns, ação, sensores, blocos de fluxo, avançados e customizados.

Figura 1 – Ambiente de programação NTX-G



Fonte: Generation Robots (2015).

Em 2013, a Lego lançou o sucessor do NXT, o Lego Mindstorms EV3. O EV3 era semelhante ao NXT em muitos aspectos, mas apresentava várias melhorias

importantes. Por exemplo, o bloco programável do EV3 tinha mais memória e um processador mais rápido do que o NXT, permitindo que os robôs fossem programados para realizar tarefas mais complexas e respondessem mais rapidamente aos comandos.

O software do EV3 também apresentava melhorias significantes em relação ao seu antecessor, NTX-G. Permitindo duas formas de programação, uma baseada em blocos chamada "Scratch" ou em python, uma das linguagem de programação orientada a objetos de alto nível. Com isso tornou-se possível a criação de programas mais sofisticados.

O sucesso do EV3 foi global, diversas instituições de ensino pelo mundo passaram a adotar robótica educacional utilizando-o como material didático, mas para muitas mantê-los não é viável. Apesar do investimento inicial não ser absurdo para as escolas, estudantes relatam imprecisão em alguns dos sensores e pouca durabilidade, acarretando custos de manutenção.

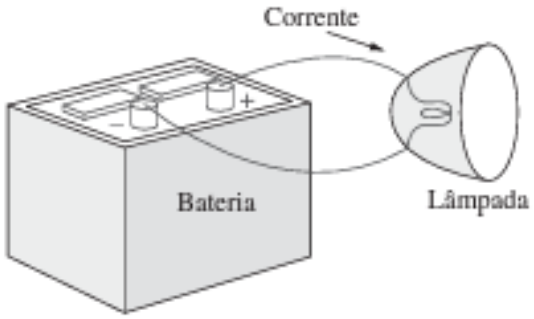
Somado a isso temos o custo, que pode não ser um problema tão grande para as escolas, porém para os estudantes, sim. O preço da versão Spike é bastante inacessível considerando o valor do salário mínimo no Brasil. E a disponibilidade do material apenas dentro das instituições limita o tempo de contato do aluno com a robótica. Por isso a criação de um material de baixo custo se faz tão importante, pois além possibilitar a replicabilidade no ambiente doméstico, também estimula o aprendizado da eletrônica, já que todas as conexões dos componentes devem ser feita manualmente e não em blocos como nos modelos da LEGO.

Apesar de kits de baixo custo serem bastante atrativos, utilizá-lo pode ser um pouco problemático no início, pois eles não contam com uma interface convidativa como o NTX-G ou utilizam uma linguagem de programação simplificada como Scratch. O correspondente ao bloco programável nessa abordagem são microcontroladores como Arduino e Esp32, que utilizam como linguagem de programação o C++.

3 METODOLOGIA

Para a montagem física dos projetos é necessário conhecer o básico sobre a teoria dos circuitos elétricos. Circuitos elétricos são interconexões entre elementos elétricos. Por mais simples que essas conexões possam ser, por exemplo, uma lâmpada conectada a uma bateria é considerado um circuito elétrico. A Figura 2 exibe esse exemplo, juntamente com outros dois aspectos fundamentais de um circuito elétrico: tensão e corrente(ALEXANDER; SADIKU, 2013).

Figura 2 – Circuito elétrico simples.



Fonte: Alexander e Sadiku (2013, p. 4).

Corrente elétrica (i) é o fluxo de carga por unidade de tempo, medido em ampères (A). Fluxo de carga nesse caso refere-se ao deslocamento dos elétrons pelo material. Se o valor da corrente não muda com o tempo e permanece constante, ela ganha o nome de corrente contínua (CC), caso seu valor se altere ao longo do tempo seguindo a forma de onda senoidal, é chamada de corrente alternada (CA). Nos projetos apresentados neste trabalho só será utilizada a contínua (ALEXANDER; SADIKU, 2013).

Tensão ou diferença de potencial (ddp), é a energia necessária para deslocar uma carga unitária através de um elemento, medida em volts (V). Assim como a corrente, a tensão pode ser CC ou CA, sendo a CC comumente proveniente de baterias como na Figura 2, e a CA proveniente de geradores, como a rede de energia doméstica (ALEXANDER; SADIKU, 2013).

Sob esse viés, é importante salientar que existem termos importantes e comumente utilizados na eletrônica: *Voltage at the Collector* (VCC), usado em diferentes contextos, como tensão no coletor em aplicações que utilizem transistores, ou tensão de alimentação em geral, outro muito utilizado é terra, do inglês *ground* (GND). Para simplificar, pense em uma bateria, o polo positivo seria o VCC e o negativo o GND.

Para compreender melhor esses dois conceitos, encha uma mangueira transparente com água, tape com as mãos as pontas, se você segurá-las na mesma

altura e deixar de tapá-las, a água não sairá, não haverá fluxo de água para fora, agora teste deixar as mãos em diferentes alturas com as pontas destampadas, se o fizer verá que a água sairá, e quanto maior a diferença de altura entre as pontas maior será o fluxo de água. A diferença de potencial é como essa diferença de altura, é a "força impulsionadora", e a mangueira é como um condutor elétrico, por inferência, a água fluindo é a corrente.

É comum que os materiais apresentem uma propriedade intrínseca de resistir a corrente elétrica. Essa propriedade física, ou habilidade, é conhecida como resistência e é representada pelo símbolo R. Sua unidade é o ohm (Ω) em homenagem a Georg Simon Ohm, físico alemão. Ela pode ser calculada conforme a Equação 1 , onde ρ é conhecida como resistividade do material em ohms-metro, e de acordo com seu valor os materiais podem ser classificados em três tipos: condutores, os semicondutores e os isolantes (VLACK, 1988).

$$R = \rho \frac{l}{A} \tag{1}$$

Sabendo os conceitos de tensão, corrente e resistência, o próximo passo seria compreender como essas grandezas se relacionam entre si. As Leis de Ohm, foram formuladas por Georg Simon Ohm em 1827, são fundamentais no estudo dos circuitos elétricos. A primeira lei de Ohm estabelece que a corrente elétrica que flui em um circuito é diretamente proporcional à diferença de potencial aplicada entre seus terminais e inversamente proporcional à resistência do circuito. Essa relação é expressa pela Fórmula 2, onde I é a corrente em amperes, V é a diferença de potencial em volts e R é a resistência em ohms.

$$V = Ri \tag{2}$$

Nem sempre deseja-se utilizar toda a corrente que a fonte fornece, alguns componentes são bastante sensíveis à altas correntes, então é muito importante saber exatamente com quais valores de corrente e tensão o seu circuito está preparado para suportar. Para aplicações simples, saber apenas a primeira lei de Ohm já resolve a maior parte dos problemas relacionados a isso.

3.1 MATERIAIS

Os componentes que compõe os projetos serão apresentados em subtópicos, a fim de fornecer uma visão clara e organizada sobre cada um deles.

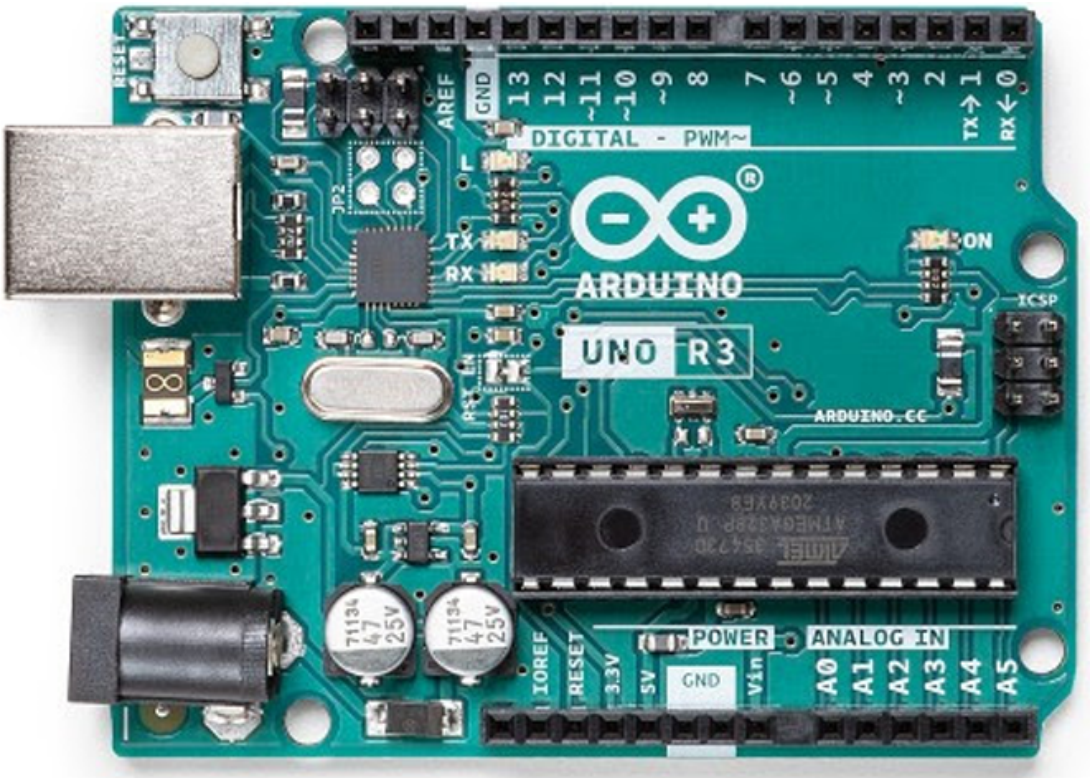
3.1.1 Arduino UNO R3

Para o desenvolvimento dos projetos o microcontrolador utilizado será o Arduino UNO R3. Ele possui 14 pinos de entrada/saída digital, 6 entradas analógicas, uma

conexão USB, um conector de energia, dentre uma série de outras características que envolvem um nível mais avançado de eletrônica (ARDUINO, 2022).

A transferência de dados é feita pelo cabo USB tipo A/B, essa conexão serve como alimentação para a placa, caso deseje que a mesma não fique presa ao computador, basta alimentá-la com uma fonte externa, uma bateria, por exemplo, ou ligada a uma tomada residencial, considerando o uso de uma fonte que tenha conversor de corrente alternada para continua.

Figura 3 – Arduino UNO R3



Fonte: Arduino (2022).

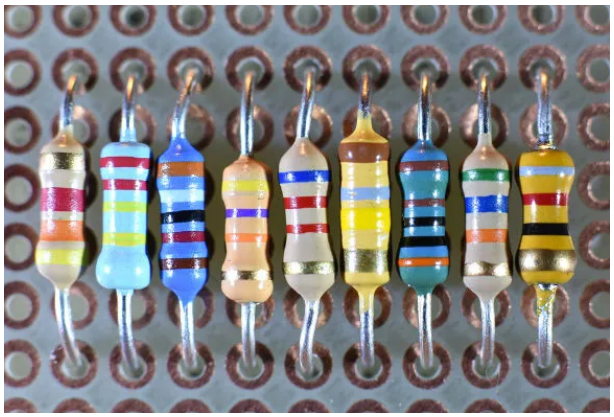
A Figura 3 exibe o modelo UNO R3, nela é possível ver toda a pinagem, e sua numeração correspondente, bem como os pinos de tensão continua 3.3V, 5V e GND. Esses pinos são especiais pois são utilizados para a energização do circuito. Nem sempre os de tensão continua são utilizados, mas o mesmo não pode ser dito do GND, não importa se a alimentação do circuito venha de uma fonte externa ou se vem da própria placa, o pino GND deverá fazer parte do circuito.

3.1.2 Resistor

Resistores, são componentes eletrônicos que resistem a passagem de corrente elétrica. São como gargalos, que ao serem adicionados ao fio condutor, diminuem a corrente e provocam uma queda na tensão. Existem diversos tipos de resistores no mercado, inclusive alguns que são utilizados como sensores, os termoresistores,

também conhecidos como termistores, possuem sua resistência elétrica variando de acordo com a temperatura. Mas, os mais usuais são os de filme metálico como os da Figura 4 (HELERBROCK, 2023).

Figura 4 – Exemplos de resistores de filme metálico.

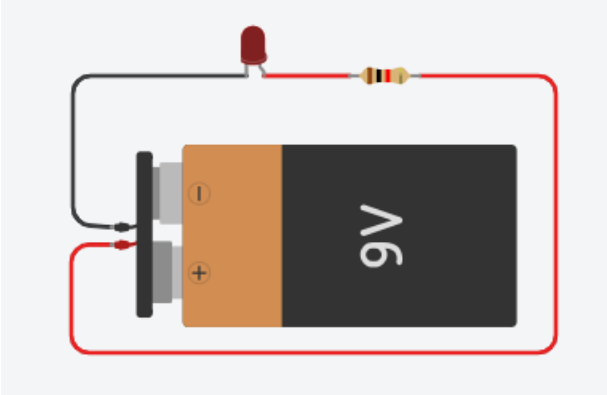


Fonte: Helerbrock (2023).

3.1.3 Diodo

Um diodo é um dispositivo eletrônico que permite o fluxo de corrente elétrica em apenas uma direção. Ele é composto por dois terminais, conhecidos como anodo (terminal positivo) e catodo (terminal negativo). O fluxo de corrente no diodo ocorre quando o anodo está em um potencial mais positivo do que o catodo.Um exemplo de diodo, e talvez o mais conhecido, é o LED, que é um diodo que emite luz visível ou invisível (infravermelha) quando energizado (BOYLESTAD; NASHELSKY, 2013).

Figura 5 – Acendendo um LED.



Fonte: Autor (2023).

Caso a ligação seja invertida, não haverá passagem de corrente, portanto o LED não ligará. Além disso, conforme consta na Figura 5, usualmente será necessário a presença de um resistor, pois a corrente suportada por esses componentes é muito baixa, tolerando uma máxima de 20 mA. Quanto mais próximo do valor máximo de corrente, mais intensamente o LED irá brilhar, e se o valor da resistência for muito alto

a corrente será miníma e o nível de brilho também. Para encontrar o valor ótimo para a resistência a ser posta no circuito, utiliza-se a Equação 3.

$$R = \frac{(V_{fonte} - V_{led})}{i} \tag{3}$$

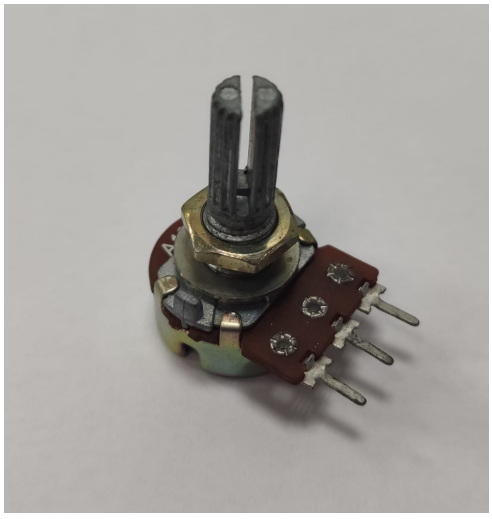
- R: resistência em ohms do resistor adequado para o LED.
- V_{fonte}: tensão da fonte usada.
- V_{led}: tensão em volts do LED, depende da cor, mas a fim de facilitar as contas 2 V já traz um resultado satisfatório e seguro.
- i: Corrente máxima, 20 mA.

Para o exemplo apresentado na Figura 5, o valor ótimo de R é (9 - 2)/0.02, que é igual a 350 Ω. A simplificação foi feita, não por que agregaria complexidade na conta, mas para facilitar a memorização. Ademais são componentes produzidos em larga escala, portanto, a confiabilidade em suas características é reduzida, inclusive no quesito corrente máxima, que pode ser maior ou menor.

3.1.4 Potenciômetro

Potenciômetro é, basicamente, uma resistência elétrica que varia conforme a posição do cursor. Eles são muito úteis para medir posição, pois são fabricados de modo que a resistência entre dois de seus terminais varie com o posicionamento do cursor. Quanto a geometria, podem ser lineares ou circulares. Para o desenvolvimento das atividades propostas será utilizado apenas o modelo linear, tal qual o da Figura 6 (AGUIRRE, 2013).

Figura 6 – Potenciômetro linear.



Fonte: Autor (2023).

3.1.5 Servo motor

Servos motores são motores CC ou AC, com seu eixo ligado a uma caixa de redução, fator que permite um controle simples de sua rotação. Os servos CC são os mais usuais, devido a facilidade em controlá-los. A quantidade de projetos nos quais eles podem ser inseridos é enorme, o que o torna um dos principais componentes de automação na atualidade, seja a nível doméstico ou industrial (FIROOZIAN, 2014).

Figura 7 – Servo motor.

Fonte: Eletrogate (2023).

Para os projetos que serão realizados nesse trabalho o modelo 9g apresentado na Figura 7 é a opção mais barata. Mas leve em consideração que o torque desse modelo não é grande, e sua rotação é limitada. Existem modelos mais fortes e com rotação contínua, tudo depende da sua necessidade.

3.1.6 Buzzer

Um *Buzzer* baseia seu funcionamento no efeito piezoelétrico reverso. O mesmo funciona a partir de uma diferença de potencial aplicada que gera uma deformação mecânica variável, produzindo assim uma onda sonora.

Pode-se encontrar dois modelos desse dispositivo no mercado, um ativo e outro passivo, apesar de ambos não serem complexos de se utilizar.

Sob essa ótica, o primeiro funciona imediatamente após ser energizado, tornando-o o mais adequado para alarmes. O segundo requer mais que apenas energização para funcionar, ele necessita também de um controle da nota a ser reproduzida, por meio da alteração da frequência com que é aplicada a diferença de potencial em seus terminais, fazendo com que ele seja o mais adequado para a criação de melodias. É importante ressaltar que há vários tipos de CI que podem substituí-lo

no processo de emissão sonora, mas neste projeto, a fim de reduzir custos e por questões de praticidade, será utilizado um *Buzzer* ativo. O mesmo está ilustrado na Figura 8, apesar de não estar muito evidente nesta imagem, ele possui o terminal do VCC ligeiramente maior que o GND (BEZERRA, 2021).

Figura 8 – Buzzer ativo.

Fonte: Bezerra (2021).

Sensor de Presença PIR (*Passive Infrared Sensor*): O movimento pode ser captado por vários tipos de sensores, dentre eles, o escolhido para esta aplicação será o sensor de presença PIR, pois o mesmo é capaz captar movimentos com até 7 metros de distância (OPEN IMPULSE, 2023) e pode ser calibrado com bastante facilidade.

Ele é composto internamente por duas faixas com material sensível ao infravermelho. Na parte externa, uma espécie de capa/tampa, que na verdade é uma lente fresnel que atua ampliando o ângulo de "visão" do sensor. Quando o diferencial de sinal entre os dois sensores muda, a saída é acionada por um determinado tempo. Este tempo, juntamente com a sensibilidade do sensor podem ser calibradas por meio de potenciômetros contidos no próprio módulo (ARDUINO E CIA, 2014).

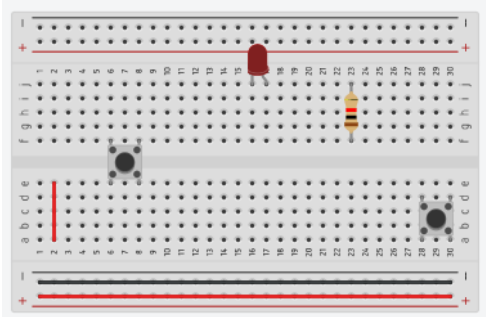
Figura 9 – Sensor de presença PIR.

Fonte: Arduino e Cia (2014).

3.1.7 Breadboard

Uma *breadboard*, também conhecida como placa de ensaio ou protoboard, é um componente eletrônico utilizado para criar e testar circuitos sem a necessidade de soldagem. Ela é uma ferramenta essencial para montagem rápida e temporária de circuitos eletrônicos. A *breadboard* possui uma base de plástico com uma matriz de furos, geralmente em padrão retangular. Esses furos são conectados internamente por trilhas condutoras, permitindo que os componentes eletrônicos sejam inseridos e conectados sem a necessidade de solda. Além disso, as trilhas são independentes entre si.

Figura 10 – *breadboard*.



Fonte: Autor (2023).

Para facilitar a compreensão do funcionamento da *breadboard*, a Figura 10 mostra como as conexões são feitas e alguns erros comuns. As extremidades de ambos os lados possuem dois símbolos, GND (-) e VCC (+), que correspondem a toda a linha horizontal a qual indicam, e do ponto de vista elétrico a linha inteira equivale a um único ponto. Visualize a linha inteira como se fosse um polo de uma bateria. Na região

central, tem-se um pequeno fio vermelho para ilustrar o tamanho das trilhas e seu direcionamento, e assim como nas extremidades a sua trilha equivale a apenas um ponto no circuito.

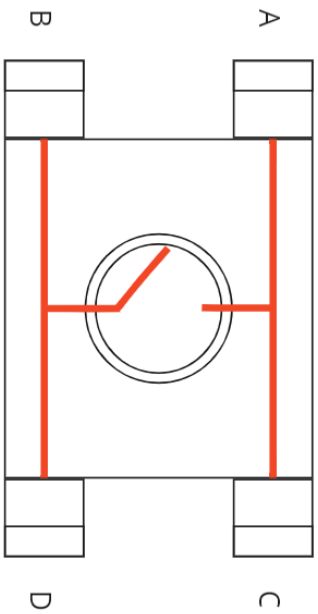
Ainda analisando a Figura 10, é possível ver que os dois botões foram posicionados de maneira distinta, o primeiro (mais a esquerda) entre os dois caminhos de trilha, portanto nenhum de seus conectores está na mesma trilha, logo, ele funcionará perfeitamente, assim como o LED vermelho, mas o mesmo não pode ser dito do botão mais a direita e o resistor, ambos tem seus terminais na mesma trilha, causando um curto-circuito.

Um curto-circuito ocorre quando há uma conexão direta e não intencional entre dois pontos de um circuito que normalmente não deveriam estar diretamente ligados. Essa conexão direta cria um caminho de baixa resistência para a corrente elétrica, resultando em um fluxo excessivo de corrente, e como já foi mencionado anteriormente, corrente demais "queima"os componentes.

3.1.8 Botões

Dentre todos os componentes apresentados até então os botões são provavelmente os mais comuns no dia a dia. Em quase toda aplicação que envolva algum controle por parte do usuário terá um botão. O tipo mais comumente utilizado em microcontroladores é o *pushbutton*, possui um mecanismo, que ao ser apertado conecta dois pontos do circuito.

Figura 11 – Funcionamento de um botão.

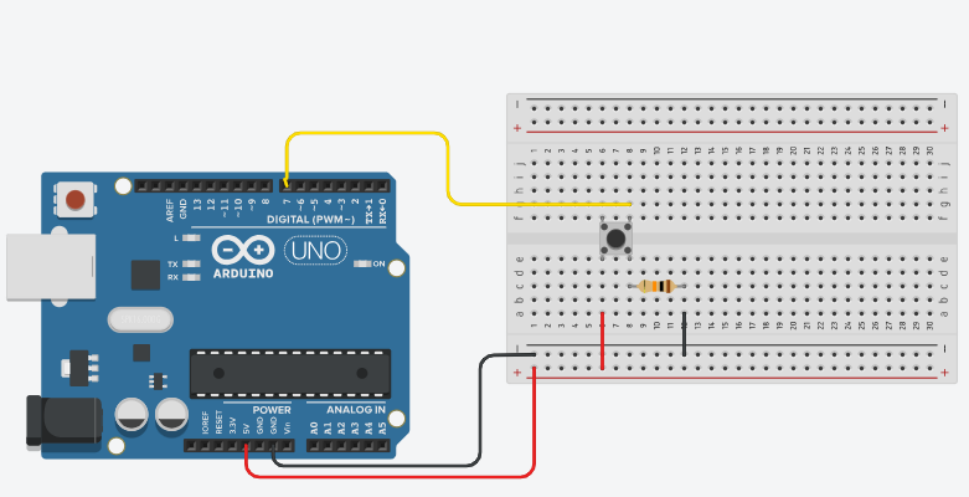


Fonte: Autor (2023).

Na figura 11, quando um botão está com contato aberto (não pressionado),

não há conexão entre os terminais B e D com os terminais A e C , então o pino está conectado ao GND, por meio de um resistor com elevada resistência,isso se chama *Pull Down*. Quando o botão tem seu contato fechado (pressionado), ele faz uma ligação entre seus dois terminais, ligando o pino à 5 volts, de forma à promover o nível lógico *High*. A Figura 11 mostra como o botão funciona internamente e a Figura 12 exemplifica seu uso no Arduino.

Figura 12 – Usando um *pushbutton* no Arduino.



Fonte: Autor (2023).

3.2 LINGUAGEM DE PROGRAMAÇÃO C++

Antes de partir para uma linguagem propriamente dita como o c++, é importante entender alguns conceitos básicos e e cruciais para o entendimento da área e, principalmente, a lógica de programação.

Para falar de programação é essencial entender o que são algorítimos. Segundo Mathias (2017, p. 12), algorítimos são: "a descrição de um conjunto de ações que, obedecidas, resultam numa sucessão finita de passos atingindo um objetivo". Nessa frase também aparece o termo **ação** que, segundo ele: "É um acontecimento que, a partir de um estado inicial, após um período de tempo finito, produz um estado final previsível e bem definido" (MATHIAS, 2017, p. 12).

Simplificando, podemos comparar algorítimos à uma receita de bolo. Você tem um objetivo (fazer o bolo), e para isso você deve seguir uma sequência de passos bem definidos, preparar os ingredientes, misturar, bater e etc. É importante ressaltar a importância da ordem das ações na construção de um algoritmo. Ainda usando o exemplo do bolo, não se pode assar se a massa não tiver sido preparada.

Uma sugestão para treinar a criação de algoritmos é pensar em situações cotidianas como, por exemplo, tomar banho e cozinhar. O interessante dessa abordagem é que caso várias pessoas sejam submetidas a tentar criar a sequência de

passos pra essas atividades, dificilmente haverão algorítimos iguais. Isso é esperado, pois a interpretação do problema é algo individual. Para ilustrar isso abaixo têm-se um exemplo de algorítimo para o ato de tomar banho.

1 Tirar a roupa
2 Abrir o chuveiro
3 Verificar se já está completamente molhado
4 Fechar o chuveiro
5 Passar sabonete
6 Esfregar-se com uma esponja
7 Abrir o chuveiro
8 Verificar se já retirou todo o sabão
9 Fechar o chuveiro
10 Enxugar-se

Observe que foram elaborados 10 passos para a atividade, em algumas situações, seguir os passos à risca é imprescindível, enquanto em outros casos, etapas podem ser saltadas . O passo [6], por exemplo, em alguns locais não é comum de ser realizado, então, em um algorítimo feito pelos habitantes desses locais provavelmente esse passo não existiria e o objetivo (banho) seria alcançado da mesma maneira. Cabe ao programador o discernimento sobre quais passos são de fato cruciais para garantir a completude dos objetivos levando em consideração os requisitos desejados.

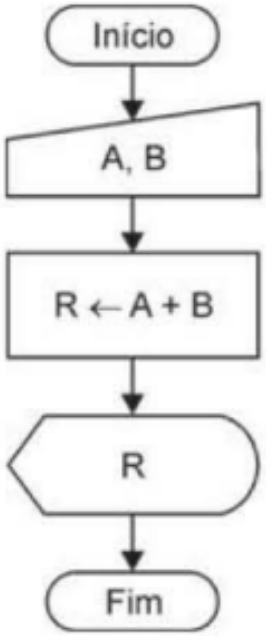
A forma apresentada acima para se criar um algorítimo é chamada de descrição narrativa, caracterizada por ser muito verbal e muito próxima da forma natural de pensamento. Existem outras maneiras de criar algorítimos que são mais usuais, como o fluxograma e o pseudocódigo, que são bem mais sucintos e mais próximos de como uma máquina funciona.

O fluxogramas são mais visuais, com eles é possível visualizar rapidamente o funcionamento de algorítimos. Contudo, diferente da descrição narrativa, para criá-lo o deve-se obedecer a norma ISO 5807: 1985 (E) (MANZANO, 2016). Nela consta os tipos de blocos e linhas de fluxo que podem ser usadas e para quais finalidades estão destinadas .

Na Figura 13, temos um fluxograma que representa a soma de dois valores A e B em conformidade com a norma. A fim de minimizar a bagagem de conteúdos portada neste trabalho, não será aprofundado os conhecimentos dessa abordagem de confecção de algorítimos.

Um programa de computador também conhecido como *software* (do inglês, programa) pode ser visto como um uma codificação de um algoritmo em uma determinada linguagem de programação. Programas podem ser interpretados como formulações concretas de algoritmos abstratos, baseados em representações e estruturas específicas de dados (MATHIAS, 2017).

Figura 13 – Exemplo de fluxograma.



Fonte: Manzano (2016, p. 33).

Para sair do algorítimo para uma linguagem de programação é necessário que se tenha um certo grau de domínio da lógica. É comum ver pessoas afirmarem que possuem e sabem utilizar o raciocínio lógico, mas quando confrontados com situações que demandem isso acabam perdendo a lina de raciocínio, pois inúmeros fatores são necessários para completá-lo, tais como conhecimento, versatilidade, experiência, criatividade, responsabilidade, ponderação, calma, autodisciplina, entre outros (MANZANO, 2016).

Um pseudocódigo é uma forma de representar um algoritmo utilizando uma linguagem de programação simples e informal. Ele não segue as regras sintáticas e estruturais de uma linguagem de programação específica, mas descreve as etapas lógicas e a sequência de instruções que um programa deve executar. Fazendo dele uma das abordagens mais escolhidas para o ensino da lógica de programação.

Aprender lógica de programação diretamente em uma linguagem também é uma possibilidade. Apesar da diferença de idiomas(as principais linguagens de programação são estruturadas em inglês), a abordagem direta é mais rápida, pois dessa forma só é necessário aprender a sintaxe de da linguagem desejada, já que mesmo pseudolinguagens tem suas próprias particularidades.

Neste trabalho será seguida a abordagem direta, e a linguagem escolhida é o C++. Várias simplificações serão aplicadas na metodologia, pois o foco será a programação de microcontroladores que a nível introdutório exige bem pouco conhecimento se comparado a toda a vastidão da linguagem. Para ajudar na associação foi preparado um exemplo de código que trabalha com basicamente todos os conceitos

básicos necessários para a programação de microcontroladores: saída serial, variáveis, funções, estruturas condicionais e de repetição. Tudo isso usando como hardware um LED, um resistor, um botão, jumpers/fios, e logicamente uma placa de Arduino.

```
// Definindo os numeros dos pinos do LED e do botao
int pinLED = 13; //global
int pinButton = 2; //global

void setup() {
  // Configurando o pino do LED como saada
  pinMode(pinLED, OUTPUT);
  // Configurando o pino do botao como entrada
  pinMode(pinButton, INPUT);

  // Iniciando a comunicacao serial
  Serial.begin(9600);
}

void loop() {
  // Verificando se o botao foi pressionado
  if (digitalRead(pinButton) == HIGH) {
    // Loop para acender e apagar o LED 5 vezes

    Serial.println("Vai comecar a piscar"); // Exibe no monitor serial a
    mensagem "Vai comecar a piscar"
    int i;//local
    for (i = 0; i < 5; i++) {
      digitalWrite(pinLED, HIGH);  // Acende o LED
      delay(500);                  // Aguarda 0,5 segundos
      digitalWrite(pinLED, LOW);   // Apaga o LED
      delay(500);                  // Aguarda 0,5 segundos
    }
  }
}
```

Listing 3.1 – Código Arduino para acender um LED.

O C++ é uma linguagem tipada, ou seja, ao declarar uma variável você deve tipificá-la. Variáveis, são identificadores criados a fim de armazenar valores ou referências a objetos no código. Na prática, são locais nomeados na memória RAM do computador, onde os valores são armazenados para posterior processamento. Outra característica marcante dessa linguagem é o uso de ponto e virgula para indicar o fim de uma linha de comando.

Para declarar uma variável em C++ basta identificar o tipo e um nome válido para a mesma. Os tipos mais básicos estão listados no Quadro 1, existem outros tipos, mas que são bem menos utilizados. Não é necessário inicializá-las com algum valor

(linha 21), mas é uma boa prática fazê-lo. Para usá-las não se deve usar o tipo, apenas o nome, como pode ser visto, por exemplo, na linha 7.

As variáveis podem ser classificadas quanto ao seu escopo de duas formas: locais e globais, as locais são limitadas ao escopo nas quais foram criadas, já as globais podem ser lidas de qualquer lugar do código. Para declarar uma variável globalmente basta que a declaração seja feita fora do escopo, no Arduino, fora de setup() ou loop().

Tabela 1 – Tipos de dados comumente utilizados

Tipo	Característica	Exemplo de uso
int	Usado para identificar números inteiros	int idade = 25;
float	Usado para identificar números de ponto flutuante(reais). A separação do número deve ser feita por ponto.	float altura = 1.72;
char	Usado para identificar caracteres. Devem ser indicados utilizando aspas simples.	char letra = 'm';
bool	Valor booleano, pode ser apenas dois valores "true"ou "false"	bool vivo = true;
String	Cadeia de caracteres, texto. Um ou mais caracteres, deve ser declarado com o uso de aspas duplas	String nome = "Igor";

Fonte: H.M. Deitel e P.J. Deitel (2011).

As estruturas condicionais no C++ são: if(se), else(senão), else if(se não se). Elas são usadas para definir o direcionamento do código com base na veracidade ou não de uma expressão lógica. Essa condição segue a lógica de Boole, também conhecida como álgebra booleana, é um sistema matemático que lida com valores lógicos e operações lógicas. Foi desenvolvida pelo matemático e lógico George Boole no século XIX e é amplamente utilizada na ciência da computação, eletrônica digital e programação (ESCOLA, Equipe Brasil).

Na Álgebra de Boole existem apenas três operadores E, OU e NÃO, no c++ são respectivamente &&, || e !. o Não (!) inverte um valor booleano, para o E(&&) e o OU(||) são aplicadas as Tabelas verdade das Figuras 14 e 15 14 e 15 (ESCOLA, Equipe Brasil).

Figura 14 – Tabela-verdade do operador && (E lógico).

expressão1	expressão2	expressão1 && expressão2
false	false	false
false	true	false
true	false	false
true	true	true

Fonte: H.M. Deitel e P.J. Deitel (2011, p. 162).

Figura 15 – Tabela-verdade do operador || (OU lógico).

expressão1	expressão2	expressão1 || expressão2
false	false	false
false	true	true
true	false	true
true	true	true

Fonte: H.M. Deitel e P.J. Deitel (2011, p. 162).

Não é preciso decorá-las, na tabela do "E", caso um valor seja falso, o resultado é falso então a única possibilidade para verdadeiro é quando todos os valores são verdadeiros. Para o "OU" basta que uma das condições seja verdadeira para o resultado ser verdadeiro, então somente na situação em que todos os valores são falsos o resulto é falso.

De volta as estruturas condicionais propriamente ditas, o *if* testa se uma dada expressão relacional é verdadeira, caso seja, o trecho de código delimitado pelo seu escopo(simplificando bastante, escopo é o código entre o par de chaves) é executado. Caso a expressão seja falsa, todo o escopo do *if* é ignorado durante a execução do código.

O *if* pode ser utilizado sozinho sem a presença de um else, mas o contrário não é possível. O escopo do *else* só é lido caso a condição do *if* ou *else if* anterior a ele seja falsa. O *else if*, por sua vez, como o nome sugere é a mistura do *else* com o *if*, ele é executado somente se a condição do *if* ou *else if*(pode ter vários) que o antecede seja falsa, mas diferente do *else* ele também tem uma condição tal qual o *if*. Além disso, dentro do escopo de qualquer um deles podem ter outros esquemas de estruturas condicionais, esses casos são nomeados como aninhamento (DEITEL; DEITEL, 2011).

```
1  if(temperatura < 20){
2      Serial.println("frio")
3  }else if(temperatura > 20 && temperatura < 40){
4      Serial.println("morno")
5  }else{
6      if(temperatura > 100){
7          Serial.println("Pegando fogo")
8      }else{
9          Serial.println("Quente")
10     }
11 }
```

Listing 3.2 – Exemplo de estruras aninhadas.

O código acima exemplifica as três estruturas de maneira aninhada. Nas linhas um, três e seis, compara-se o valor da variável utilizando os chamados operadores lógicos, a Figura 16 exibe todos os operadores lógicos em c++. Um erro comum com iniciantes na programação é confundir o operador de atribuição "=" com o de igualdade

"==", pois a mente ainda está habituada com o uso desse sinal apenas na aritmética.

Figura 16 – Operadores de igualdade e operadores relacionais.

Operador algébrico de igualdade ou relacional padrão	Operador de igualdade ou relacional em C++	Exemplo de condição em C++	Significado da condição em C++
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior que ou igual a y
≤	<=	x <= y	x é menor que ou igual a y
Operadores de igualdade			
=	==	x == y	x é igual a y
≠	!=	x != y	x não é igual a y

Fonte: H.M. Deitel e P.J. Deitel (2011, p. 38).

Códigos para Arduíno por padrão devem apresentar duas funções: setup() e loop(). Dentro do escopo da setup são feitas as conexões com o hardware e por isso são executadas apenas uma vez ao ligar a placa a uma fonte de alimentação, já a loop é repetida até que a alimentação seja removida. No fluxo normal de execução as linhas são executadas em ordem uma por vez, mas como foi visto, o trecho dentro de estruturas condicionais só é lido caso uma dada condição seja verdadeira. Muitas aplicações demandam que certos trechos do código repitam várias vezes antes que o resto do programa seja executado, para isso são usadas as chamadas estruturas de repetição.

Uma instrução de repetição, também chamada de laço de repetição ou simplesmente *loop* permite ao programador especificar um conjunto de instruções enquanto uma condição (chamada de condição de parada) permanecer verdadeira. No C++ tem-se três formas de se criar um laço de repetição sendo elas: **do while**, **while** e **for**. O *while* e o for funcionam da mesma forma, antes de executarem seu bloco de código verificam a veracidade da condição de parada, já o do *while* executa a primeira iteração sem checar a condição de parada, e caso a condição não tenha sido atingida ele realiza a próxima iteração (DEITEL; DEITEL, 2011).

O *do while* foi exposto apenas a nível de curiosidade, são poucas as vezes em que ele se faz necessário, mas é importante saber sua existência. A sintaxe das outras duas estruturas está apresentada dentro do trecho de código abaixo. Um detalhe a ser pontuado é a possibilidade de *loops* infinitos, caso sua condição de parada nunca seja alcançada, o trecho do *loop* será executado.

```
1 //for(iterador;condicao de parada;incremento){}
2 for(inti i=0;i<10;i++){ //o i++ e a mesma coisa de i+=1 e que tambem e
     igual a i=i+1;
3     Serial.println(i);//exibe no monitor serial a contagem de 0 a 9
4 }
```

```
int j=0;
//while(condicao)
while(j<10){
    Serial.println(j);//exibe no monitor serial a contagem de 0 a 9
    j = j+1;
}
```

Listing 3.3 – For e while

A maioria dos programas possuem cálculos aritméticos. As operações devem ser realizadas do lado direito do operador de atribuição, do lado esquerdo só é aceita a existência de uma única variável, um exemplo simples de expressão algébrica em C++ é: a = b * (b+2*c). A Figura 17 contém todos os operadores aritméticos presentes no C++ (DEITEL; DEITEL, 2006).

Figura 17 – Operadores aritméticos.

Operação C++	Operador aritmético C++	Expressão algébrica	Expressão C++
Adição	+	$f + 7$	f + 7
Subtração	-	$p - c$	p - c
Multiplicação	*	bm ou $b \cdot m$	b * m
Divisão	/	x / y ou $\frac{x}{y}$ ou $x \div y$	x / y
Módulo	%	$r \bmod s$	r % s

Fonte: H.M. Deitel e P.J. Deitel (2011, p. 35).

Como foi visto na apresentação dos conceitos, os programas podem escalar em termos de tamanho muito facilmente, imagine que você deva monitorar variáveis, realizar cálculos aritméticos e *loops* em uma mesma aplicação. Por isso existem as funções, elas ajudam a modularizar o código, dividindo-o em partes menores, essa técnica chama-se dividir para conquistar (DEITEL; DEITEL, 2006).

Uma função é declarada de maneira similar a uma variável, ela deve ter um tipo para o valor que ela retorna(caso não tenha retorno, usa-se void) e pode ou não receber parâmetros. Um exemplo de definição de função com retorno é: "int soma_inteiros(int parametro1, int parametro2);", para adicionar novos parâmetros basta colocar uma virgula, tipo e nome para o novo parâmetro. Porém, além de declará-las é preciso defini-las (se nada for definido a função não fará nada), isso nada mais é do que criar o bloco de código que deve ser executado sempre que a função for chamada. No código abaixo tem-se alguns exemplos de funções.

```
bool meChamou = false;
float getTemperatura(){// sem parametros
    float temp = 37.5;// observe que tipo eh o mesmo do retorno da
   funcao, caso contrario haveria um erro de incompatibilidade de tipo.
    return temp;
}
void fuiChamado(){
```

```
7       meChamou = true;
8   }
9   int soma(int numeroA, int numeroB){
10      int valorSomado = numeroA + numero B;
11      return valorSomado;
12  }
13  setup(){//funcao sem retorno e que nao precisa de argumentos
14  //nenhuma definido aqui
15  }
16  loop(){
17      fuiChamado();// muda o valor de meChamou para true
18      int testeSoma = soma(1,1);
19      float temperaturaAtual = getTemperature();// A variavel
        temperaturaAtual recebe o retorno da funcao getTemperature
20  }
```

Listing 3.4 – Declaração e definição juntas.

```
1   bool meChamou = false;
2   void fuiChamado();
3   float getTemperature();
4   int soma(int numeroA, int numeroB);
5   setup(){//funcao sem retorno e que nao precisa de argumentos
6   //nenhuma definido aqui
7   }
8   loop(){
9       fuiChamado();// muda o valor de meChamou para true
10      float temperaturaAtual = getTemperature();// A variavel
        temperaturaAtual recebe o retorno da funcao getTemperature
11  }
12
13  float getTemperatura(){// sem parametros
14      float temp = 37.5;// observe que tipo eh o mesmo do retorno da
        funcao, caso contrario haveria um erro de incompatibilidade de tipo.
15      return temp;
16  }
17  void fuiChamado(){
18      meChamou = true;
19  }
20  int soma(int numeroA, int numeroB){
21      int valorSomado = numeroA + numero B;
22      return valorSomado;
23  }
24  //Podem ser feito aritimetica dentro diretamente no return
25  //Para a funcao soma poderia ser apenas: return numeroA+numeroB;
```

Listing 3.5 – Declaração e definição separadamente

Para chamar uma função, primeiro verifique se ela possui algum retorno, em caso afirmativo, certifique-se de receber esse valor numa variável caso deseje utilizá-lo posteriormente. Em seguida abra parênteses e passe os argumentos necessários para a execução da função. Argumentos são valores condizentes com a tipagem dos parâmetros, que são passados na chamada da função para que a mesma execute o que você deseja. Devem ser colocados em ordem.

Caso a função seja do tipo *void* não é necessário armazenar seu retorno, até por que não há retorno. Mas se a função não tiver nenhum parâmetro ainda é necessário pôr os parenteses vazios. É interessante saber que em uma mesma função podem ter vários *returns*. Além disso em funções *void*, o programador pode colocar um retorno vazio, basta digitar "return;", de primeiro momento parece completamente sem significado retornar "nada", mas isso pode permitir uma saída prematura do escopo da função, e em muitas situações pode ser útil.

O *return* faz parte da sintaxe das funções e não pode ser utilizado fora do escopo delas, e assim que executado ele retorna o código para a o ponto em que a função foi chamada, mesmo que ainda houvesse outras instruções a serem executadas. Então, apesar de não ser uma regra ou um erro, é uma má prática colocar muitos returns em uma mesma função, pois quebra a linearidade do código.

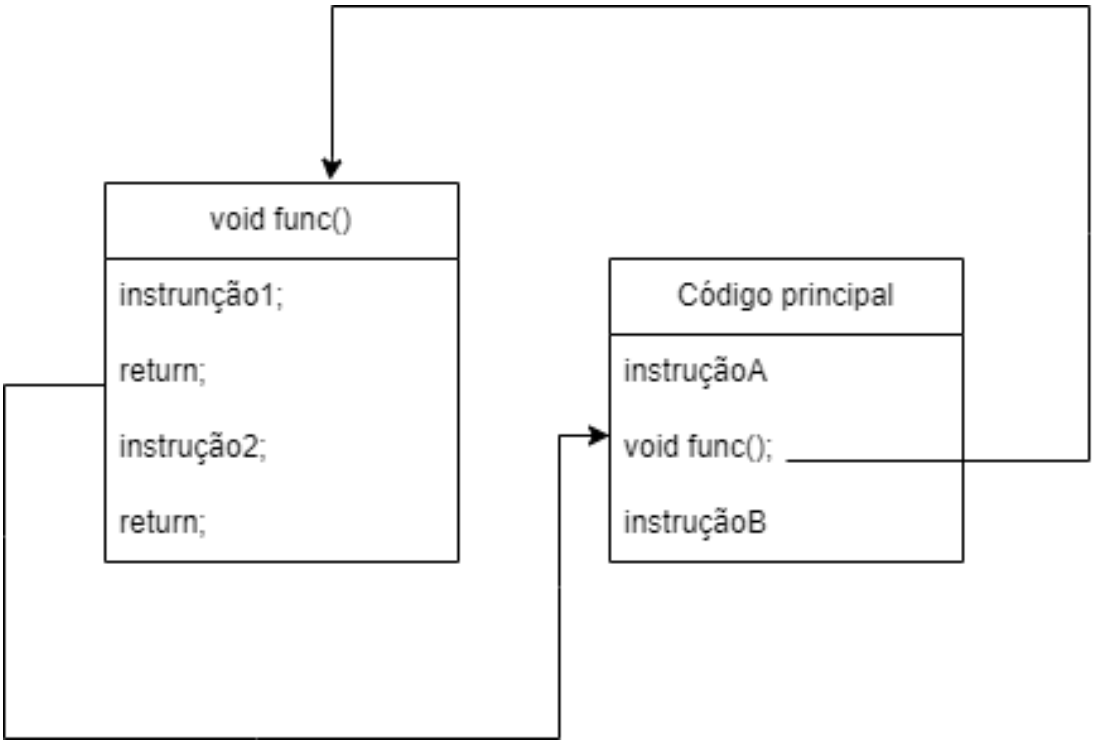
A Figura 18 exemplifica um fluxo de um algorítimo que tenha chamada de função. Ao executar a chamada da função o código parte para a leitura do escopo da mesma, lá há a presença de um *return* antes da instrução2, por conta dessa estruturação mal feita de *returns*, a instrução2 em hipótese alguma é realizada.

Por fim, o tópico que é provavelmente o mais importante para a programação de microcontroladores, classes. Elas são a base para a programação orientada a objetos, por meio delas consegue-se criar abstrações de objetos reais no meio virtual. Essas abstrações consistem em pegar as características e funcionalidades que são necessárias para sua aplicação e aglomerá-las juntas, as características são chamadas de atributos(variáveis) e as funcionalidades de métodos(funções).

A criação dessas estruturas será omitida, pois possuem uma série de detalhes muito avançados para iniciantes, portanto o foco será em seu uso, já que boa parte dos sensores e atuadores possuem bibliotecas específicas, e inserindo-as no código basta apenas utilizar o que for necessário para sua aplicação, sem se preocupar em como os métodos dela foram criados.

Objetos são instâncias(ou ocorrências de uma classe), para compreender isso pense em uma classe chamada **Pessoa**, caso você deseje criar um objeto dessa classe, você deve declará-lo tal qual declara uma variável, por exemplo: "Pessoa Manoel;". O exemplo só é válido para classes que possuem um construtor(método especial que inicia a classe) vazio, mas a maioria das aplicações o possuem.

Figura 18 – Uso inadequado de múltiplos returns.



Fonte: Autor (2023).

Para acessar métodos basta utilizar um "."imediatamente após o nome do objeto e em seguida digitar o método da mesma forma que é feito com funções, colocando parenteses e passando argumentos, caso seja necessário. Quanto aos atributos, por conta de uma característica chamada encapsulamento das linguagens orientadas a objeto, o desenvolvedor da classe usualmente não permite que o usuário tenha acesso aos atributos da classe diretamente, e só permite modificá-los e acessá-los por meio de métodos*get*(pegar) e *set*(definir).

Para acessar a um suposto atributo idade do exemplo da classe pessoa, a chamada seria algo do tipo: "Manoel.getIdade();". E Para alterar esse valor: "Manoel.setIdade(25);". Para descobrir os nomes exatos dos métodos de uma classe que não foi construída por você, busque a documentação da mesma na Internet. Documentações são como manuais, que contém informações detalhadas sobre um software ou código-fonte, criado com o intuíto de fornecer orientação para os usuários.

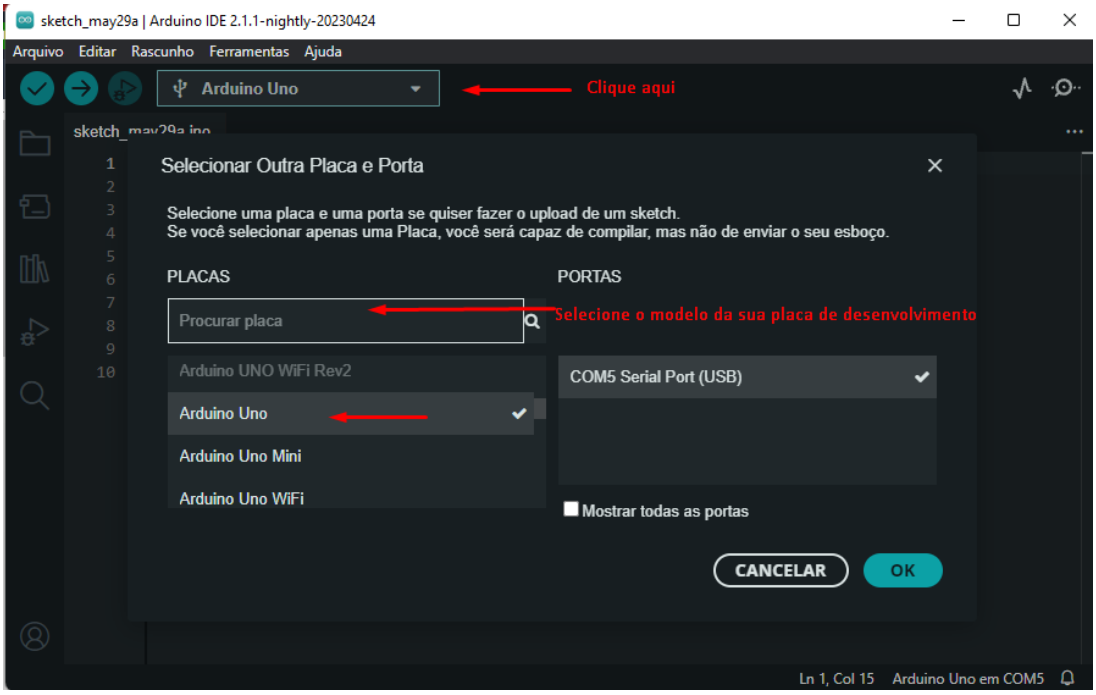
Classes de sensores e atuadores por padrão não vem embutidas no Arduino. Mas elas estão disponíveis em bibliotecas que podem ser baixadas gratuitamente da Internet. Biblioteca é um conjunto de código pré-compilado que contém um conjunto de funções, classes e outros componentes que podem ser reutilizados em diferentes programas. As bibliotecas fornecem funcionalidades específicas para facilitar o desenvolvimento de software, permitindo que os programadores utilizem implementações prontas ao invés de terem que escrever todo o código do zero.

Para adicionar uma biblioteca no seu código em C++ é muito simples, se for uma biblioteca nativa da sua versão do g++(compilador do C++) basta colocar: #include <nomeDaBoblioteca>. Caso seja uma biblioteca de terceiros, a inclusão é : #include "nomeDaBiblioteca". Observe que na inclusão de bibliotecas não é necessário colocar o ";"no final da linha.

3.3 ARDUINO

Para programar na placa Arduino, usualmente utiliza-se sua própria *Integrated Development Environment* (IDE) que é um ambiente de desenvolvimento integrado, ou seja, um software que oferece um conjunto de ferramentas e recursos para facilitar o desenvolvimento de software. Ela está disponível gratuitamente no site oficial da Arduino. Após baixar e instalar a IDE, crie um novo esboço e selecione a sua placa de desenvolvimento.

Figura 19 – Selecionando placa e porta.



Fonte: Autor (2023).

Se sua placa e porta(entrada USB na qual o seu Arduino está conectado ao computador) já estiverem selecionados conforme a Figura 19, então o ambiente de programação está pronto para desenvolver sua aplicação.

A plataforma Arduino independentemente do modelo da placa, utiliza a mesma estrutura de programação, uma função setup() para pré-definições que serão executadas no momento em que a alimentação for conectada e uma função loop() que se repetirá até que a alimentação seja cancelada.

A função setup() (em português, configurar), como o nome sugere, é uma

função de configuração, nela você deve configurar pinos, inicializar sensores/atuadores e protocolos de comunicação, conforme a necessidade do seu projeto. Para configurar pinos usa-se a função pinMode(), que recebe dois argumentos, respectivamente numero do pino (valor inteiro) e se o pino é de saída (OUTPUT) ou de entrada (INPUT) (ARDUINO, 2023).

Uma boa prática é atribuir um "apelido" aos pinos, utilizando o #define. Sua sintaxe é muito simples: #define apelido valor_Apelidado, sem ponto e vírgula no final. Com isso a leitura do código fica mais simples e intuitiva. Pinos OUTPUT, são aqueles enviam corrente da placa, já os de INPUT são os que recebem corrente. Na prática, pinos de saída são usados para atuadores, já os de entrada para sensores. É importante ressaltar que só são configurados os pinos digitais, não é necessária essa definição para os analógicos.

```
#define ledPin  13
#define motor 7
#define PIR 8
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(motor, OUTPUT);
  pinMode(PIR, INPUT);
}
void loop() {
}
```

Listing 3.6 – Exemplo de configuração de pinos.

A inicialização de sensores/atuadores dependem de qual sensor/atuador está sendo configurado. Como já foi mencionado no aprendizado do C++, os sensores/atuadores são abstraídos e encapsulados em uma biblioteca, interna ou externa à IDE, então é de se esperar que possuam métodos diferentes. Abaixo tem-se um exemplo de inicialização de um servo e de um sensor de temperatura e umidade dht11.

```
#include <Servo.h>
#include <DHT.h>
#define DHTPIN 2  // Pino ao qual o sensor esta conectado
#define pinServo 9

DHT dht(DHTPIN, DHT11);  // Criar uma instancia global do sensor DHT11

Servo myservo; //Criar uma instancia global do servo

void setup() {
  myservo.attach(pinServo);//funcao que inicializa um servo
  dht.begin();// inicia o dht11
```

```
13 }
14
15 void loop() {
16 }
```

Listing 3.7 – Exemplo de configuração de pinos.

Na função *loop*() é onde a lógica da aplicação deve ser inserida. Para isso as funções primordiais são: digitalWrite, digitalRead, analogWrite e analogRead. As digitas trabalham apenas com valores digitais (0 ou 1) de sinal alto (HIGH) ou sinal baixo (LOW), e as analógicas com valores de tensão que variam de 0 a 5V (no modelo UNO) (ARDUINO, 2023).

A digitalWrite tem a seguinte sintaxe: digitalWrite(pino, valor). O pino é o número do pino e o valor, poder ser HIGH ou LOW. É do tipo void, portanto não tem retorno. Mentalize-a como um interruptor, se você desliga um interruptor a lâmpada apaga e se você liga, ela acende. A digitalRead por sua vez tem a drguinte sintaxe: digitalRead(pino), ela possui um retorno, que poder ser HIGH ou LOW.

Para as funções entrada e saída analógica, temos as seguintes sintaxes: analogRead(pino) e analogWrite(pino, valor). A analogRead realiza leitura analógica do pino, no entanto esta é limitada a resolução do conversor analógico digital (0-1023 para 10 bits ou 0-4095 para 12 bits).

A analogWrite pode ser usada para variar o brilho de um LED ou acionar um motor a diversas velocidades. Ela faz isso utilizando *Pulse Width Modulation* (PWM), mas modulação de pulso é uma discussão a nível de graduação. Em termos práticos, "valor" deve ser um número inteiro entre 0 (desligado) e 255 (ligado no máximo).

4 PROPOSTA DE EXPERIMENTOS E DISCUSSÃO

Os projetos foram desenvolvidos na plataforma tinkercad, disponível gratuitamente no site: www.tinkercad.com. Nela é possível criar circuitos, com ou sem Arduino e simulá-los. Além disso, nos projetos que utilizam o Arduino, ela habilita um editor de texto para que o usuário possa programar dentro da plataforma e já verificar os resultados.

Com o tinkercad é possível simular os projetos sem que haja o risco de danificar os componentes reais. Ademais, ele possui uma série de componentes eletrônicos que podem ser utilizados para testar ideias. Dessa forma, o que deve ou não ser comprado para o modelo físico fica completamente definido já na simulação, evitando excesso de gasto com componentes desnecessários.

Foram elaborados cinco projetos, sendo dois deles continuações diretas de outros dois, visando mostrar a escalabilidade de ideias e suas aplicações, que possam ser usadas para resolver problemas reais. O último projeto foi criado tendo em mente algumas expansões que podem ser produzidas utilizando componentes de outras aplicações.

Primeiramente, todos os componentes utilizados, bem como os valores médios de preço obtidos em lojas online, foram listados na Tabela 2. Contudo, cada projeto apresentado possuirá uma tabela com apenas os componentes neles utilizados e o valor total do kit individualmente.

Tabela 2 – Lista de Componentes.

Componente	**Quantidade**	**Preço**
Arduino + cabo	1	R$ 95,00
Led	12	R$ 6,00
Jumpers	65	R$ 12,73
Servos	4	R$ 71,00
Potenciômetros	1	R$ 2,10
Bateria 9V	1	R$ 7,00
Buzzer ativo	1	R$ 2,75
Botão	1	R$ 1,80
Resistor 220 Ω	6	R$ 1,32
Resistor 10k Ω	1	R$ 0,20
Sensor PIR	1	R$ 12,95
Protoboard 830 Pontos	1	R$ 16,90
Estrutura do braço em MDF	1	R$ 16,90
Total		**R$ 246,65**

Fonte: Autor (2023).

4.1 BLINK

O projeto mais básico utilizando o arduino é o *Blink*, seu código está disponível nos exemplos que a própria IDE do Arduino fornece e no Apêndice A deste trabalho. Nessa aplicação tem-se a presença de dois componentes, um LED e um resistor conectados em série. Para fazer o LED "piscar", você deve ligá-lo e de desligá-lo em um intervalo de tempo de sua preferência, em outras palavras mandar um sinal HIGH (ligado) e um sinal LOW (desligado) após um *delay* (intervalo de espera).

Tabela 3 – Componentes para o blink.

Componente	**Quantidade**	**Preço**
Arduino + cabo	1	R$ 95,00
Led	1	R$ 0,50
Jumpers	3	R$ 0,60
Protoboard 830 Pontos	1	R$ 16,90
Resistor 220 Ω	1	R$ 0,22
Total		**R$113,22**

Fonte: Autor (2023).

A Figura 20 apresenta a montagem do circuito na *breadboard*, o LED utilizado que tem a cor vermelha etá ligado em série com um resistor de 220 Ω. Valor ideal da resistência conforme as simplificações assumidas seria de 150 Ω, conforme é mostrado em 4, porém resistores de 220 Ω são mais facilmente encontrados em lojas de eletrônica, e quase não há diferença visível no nível de brilho.

$$R = \frac{(V_{\text{fonte}} - V_{\text{led}})}{i} = \frac{(5-2)}{0,02} = 150\Omega \tag{4}$$

A alimentação é feita pelo pino digital 7, que em sinal alto fornece uma tensão de 5 V. Note que no tinkercad a conexão feita com o 5 V está no conector que possui uma curvatura, no componente físico, os dois terminais tem comprimentos diferentes, o maior é o anodo (ligue no positivo, 5 V, 3.3 V ou pino digital escolhido) e o menor o catodo (ligue no GND).

A construção lógica do código foi muito simples. Observou-se o objetivo (piscar), compreendeu-se o fenômeno (acender e apagar). Então aplicou-se o conhecimento da linguagem já apresentado. Os detalhes estão comentados em verde diretamente no código.

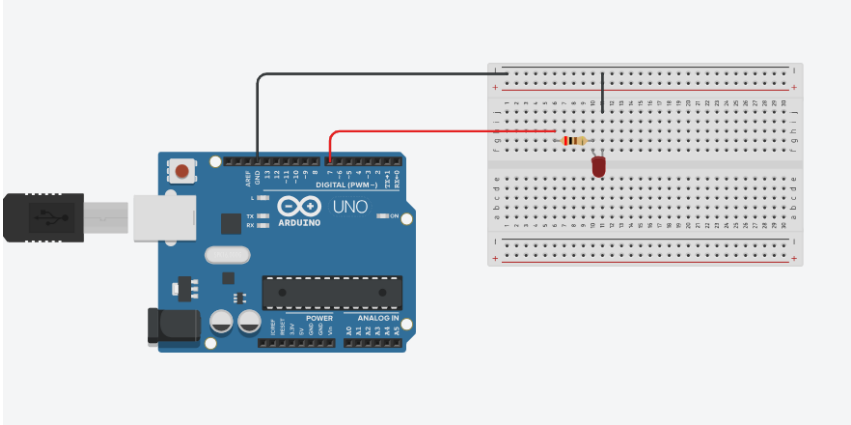
```
1 #define LED 7
2 void setup()
3 {
4   pinMode(LED, OUTPUT);//Configura o pino LED como de saida
```

```
}

void loop()
{
  digitalWrite(LED, HIGH);//Acende o LED
  delay(1000); // Espera 1000 millisecond(s), equivalente a 1 segundo
  digitalWrite(LED, LOW);//Apaga o LED
  delay(1000); //Espera 1000 millisecond(s)
}
```

Listing 4.1 – Código para o blink.

Figura 20 – Pisca.



Fonte: Autor (2023).

4.2 SEMÁFORO

Pensando de maneira objetiva, um semáforo é composto por luzes que acendem e apagam seguindo uma ordem fixa com intervalos definidos. Então, podemos compreendê-lo como uma aplicação prática do *blink*. O circuito para um único semáforo é um blink triplicado, conforme consta na Figura 21, o diferencial é o controle de estados, que são idênticos a um semáforo real, sendo eles: siga, atenção e pare. A Tabela 4 apresenta o custo do experimento.

Tabela 4 – Componentes para o semáforo.

Componente	**Quantidade**	**Preço**
Arduino + cabo	1	R$ 95,00
Led	3	R$ 1,50
Jumpers	10	R$ 2,00
Protoboard 830 Pontos	1	R$ 16,90
Resistor 220 Ω	3	R$ 0,66
Total		**R$116,06**

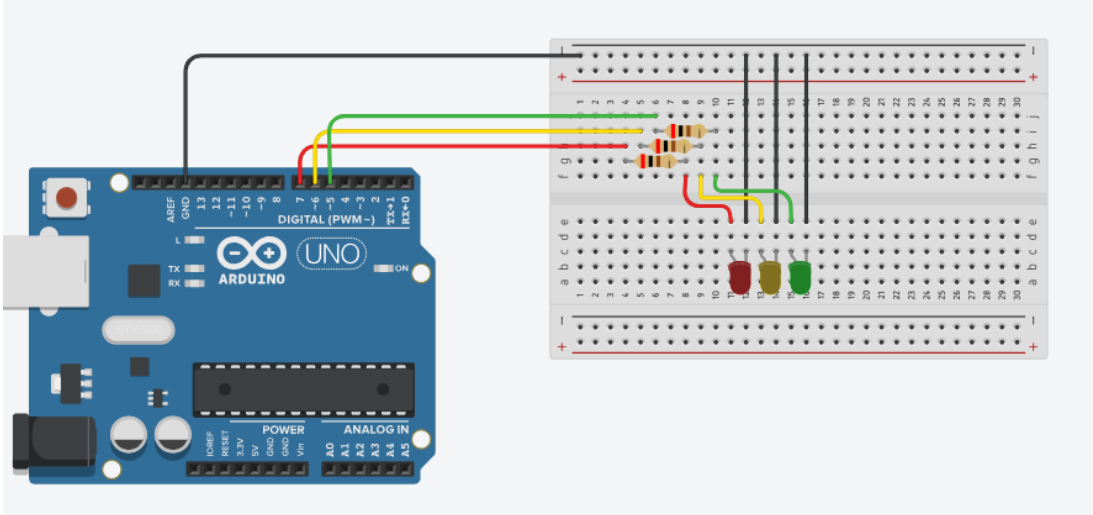
Fonte: Autor (2023).

Como pode ser visto na Figura 21, a montagem segue o mesmo modelo do blink. O maior diferencial entre os dois projetos é a programação, que requer um pouco mais de controle lógico, pois agora existem três estados a serem considerados: verde (siga), amarelo (atenção) e vermelho (pare). Cada estado consiste em dar um sinal de HIGH para o LED que deve ficar aceso e LOW para os demais. Na implementação do código, define-se os três pinos para os LEDS e no *setup()* os configura como OUTPUT. Na função *loop()*, os estados foram implementados utilizando as funções *digitalWrite()* para acender e apagar os LEDS e *delay()* para controle da duração de cada estado. Abaixo mostra como foi implementado o estado de siga, o código completo encontra-se no Apêndice B.

```
1  //Acende o verde e apaga os outros LEDS -> Siga
2  digitalWrite(ledVermelho, LOW);
3  digitalWrite(ledVerde, HIGH);
4  digitalWrite(ledAmarelo, LOW);
5  delay(3000); // Espera 3000 millisegundos, equivalente a 3 segundos
```

Listing 4.2 – Siga.

Figura 21 – Semáforo.



Fonte: Autor (2023).

4.2.1 Semáforo para um cruzamento

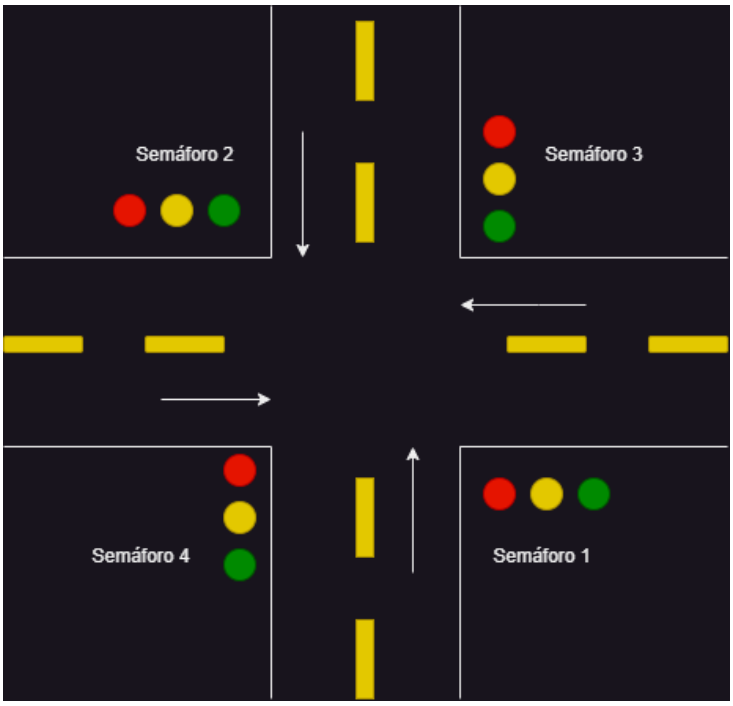
Semáforos comumente são associados a outros para o controle do fluxo de veículos em cruzamentos. Neste subtópico será abordada a aplicação deles em conjunto, o objetivo será criar e associar semáforos para reproduzir um cruzamento como o da Figura 22, respeitando os fluxos representados pelas setas.

Tabela 5 – Componentes para o cruzamento.

Componente	Quantidade	Preço
Arduino + cabo	1	R$ 95,00
Led	12	R$ 6,00
Jumpers	26	R$ 5,20
Protoboard 830 Pontos	1	R$ 16,90
Resistor 220 Ω	6	R$ 1,32
Total		**R$124,42**

Fonte: Autor (2023).

Figura 22 – Exemplo de cruzamento.



Fonte: Autor (2023).

Avaliando a Figura 22, nota-se dois fluxos: vertical (semáforo 1 e 2) e horizontal (semáforo 3 e 4). A abordagem natural com base no que já foi apresentado, seria definir 12 pinos para os LEDs dos quatro semáforos e definir seus comportamentos (ligado, desligado e intervalos). Apesar de ser uma solução, essa abordagem é bem precária, primeiramente pela grande quantidade de pinos utilizada, segundo a programação seria desnecessariamente longa, para cada estado seriam necessários 12 chamadas de *digitalWrite()*. Com isso, percebe-se que a abordagem imediata e mais "simples" nem sempre é a mais correta.

Uma abordagem mais interessante parte de uma observação mais crítica do problema. Para o cruzamento apresentado, deseja-se garantir o fluxo de veículos, com base nisso percebe-se que os semáforos do fluxo vertical devem ser síncronos entre si, e o mesmo vale para os da horizontal. Em outras palavras os sinais dos semáforos de

um mesmo fluxo devem ser iguais, por exemplo, se está verde no semáforo 1 então deve está verde no semáforo 2 para que o fluxo do trânsito nessa direção seja liberado. Logo, em termos de programação o problema pode ser abstraído para apenas dois semáforos.

Estudando os estados possíveis para os dois fluxos, chegou-se a um total de quatro estados necessários para o funcionamento da aplicação. Cada estado requer um total de seis chamadas de *digitalWrite()*. Eles estão exibidos na Figura 23, nela a seta que sai do estado "Semáforo 2 - Amarelo ligado" e aponta de volta ao início é apenas para identificar que será algo cíclico, ou seja, que estará dentro do *loop()*.

Figura 23 – Estados para o cruzamento.

Fonte: Autor (2023).

Implementar esses estados é uma tarefa simples, como já foi dito, basta utilizar a *digitalWrite()* para ligar ou desligar um dado LED, escolher um tempo para cada estado usando a função *delay()* e fazer isso para todos os estados apresentados. Abaixo está exemplificado a implementação de um estágio. O código completo para essa aplicação está disponível no Apêndice C.

```
...
void loop()
{
  //Estado 1 - Verde do semaforo 1 ligado
  digitalWrite(verde1, HIGH);
  digitalWrite(amarelo1, LOW);
  digitalWrite(vermelho1, LOW);

  digitalWrite(verde2, LOW);
  digitalWrite(amarelo2, LOW);
  digitalWrite(vermelho2, HIGH);

  delay(3000);//tempo em sinal verde no semaforo 1
  ...
```

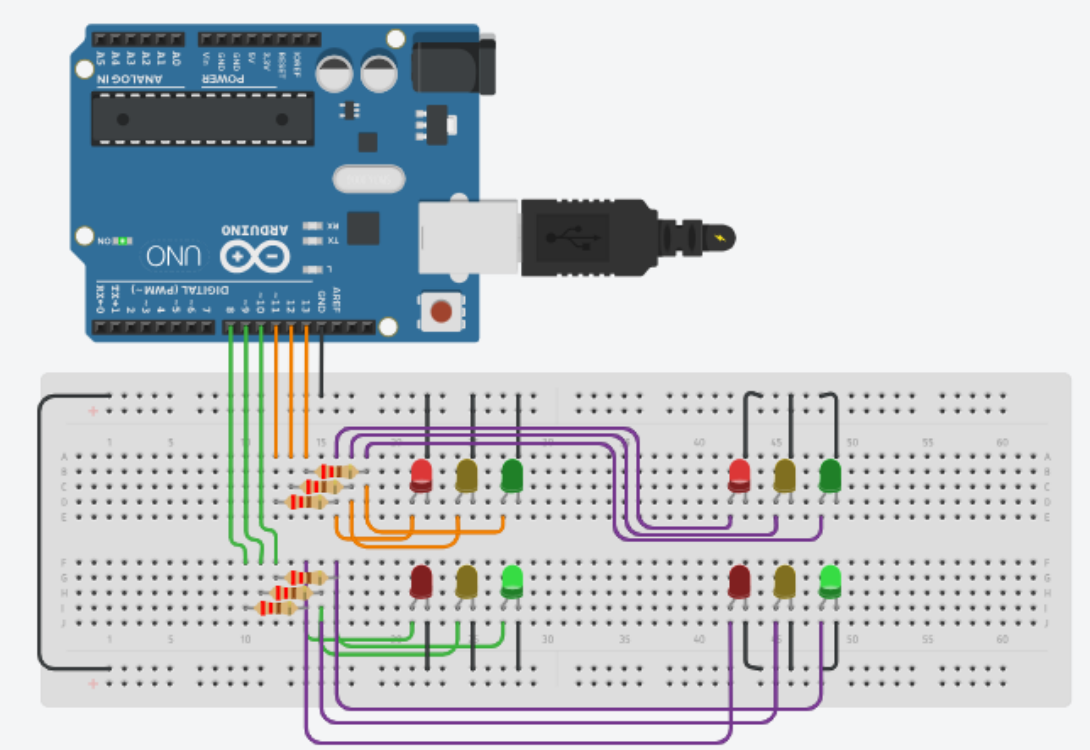
```
15 }
```

Listing 4.3 – Estado 1.

O hardware, apesar de parecer apenas uma versão quadruplicada do semáforo, possui suas próprias especificidades. A Figura 24 mostra a simulação do circuito, nela é possível perceber que os grupos de LEDs à direita da protoboard compartilham a mesma ligação com o Arduino que os da sua esquerda. Na prática isso significa que o valor escrito/lido será compartilhado entre eles.

O motivo pelo qual a ligação foi feita após o resistor está relacionada a economia de componentes, caso a conexão fosse posta antes da corrente passar pelo resistor, seria necessário adicionar resistores para os LEDs da direita também, adicionando uma complexidade desnecessária ao circuito. Vale ressaltar que da maneira que foi realizada a montagem, haverá uma perda de brilho nos LEDs, pois a corrente que antes era fornecida para apenas um deles, agora deverá ser dividida para dois. Se achar que a intensidade está fraca, substitua o resistor por um com menor resistência.

Figura 24 – Cruzamento com semáforo.



Fonte: Autor (2023).

4.3 CANCELA ACIONADA POR POTENCIÔMETRO

Neste projeto tem como objetivo criar uma cancela que abra e feche conforme um potenciômetro linear seja girado. Para realizar utilizou-se um servo motor do modelo

9g, que são muito populares em pequenos projetos de automação, e como trata-se apenas de uma miniatura, seu torque pequeno não é um problema. A Tabela 6 contém todos os elementos utilizados na criação do projeto.

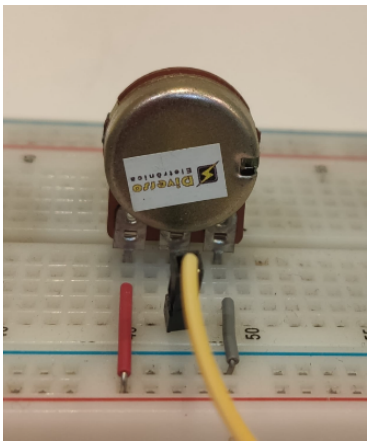
Tabela 6 – Cancela acionada por potenciômetro.

Componente	**Quantidade**	**Preço**
Arduino + cabo	1	R$ 95,00
Jumpers	11	R$ 2,20
Servos	1	R$ 17,75
Potenciômetros	1	R$ 2,10
Bateria 9V	1	R$ 7,00
Protoboard 830 Pontos	1	R$ 16,90
Total		**R$ 140,95**

Fonte: Autor (2023).

O potenciômetro possui três terminais: GND, sinal e VCC. Para conectá-lo à protoboard reproduza o exemplo da Figura 25, nele o fio cinza corresponde ao GND, o vermelho o VCC e o amarelo o terminal de sinal. Os terminais VCC e GND foram ligados as linhas de positivo e negativo da protoboard, que estão alimentadas pelo VCC e GND do arduino. O pino de sinal deve ser conectado à porta analógica, para que a leitura do valor da queda ou aumento de tensão seja lido.

Figura 25 – Conectando um potenciômetro na protoboard.

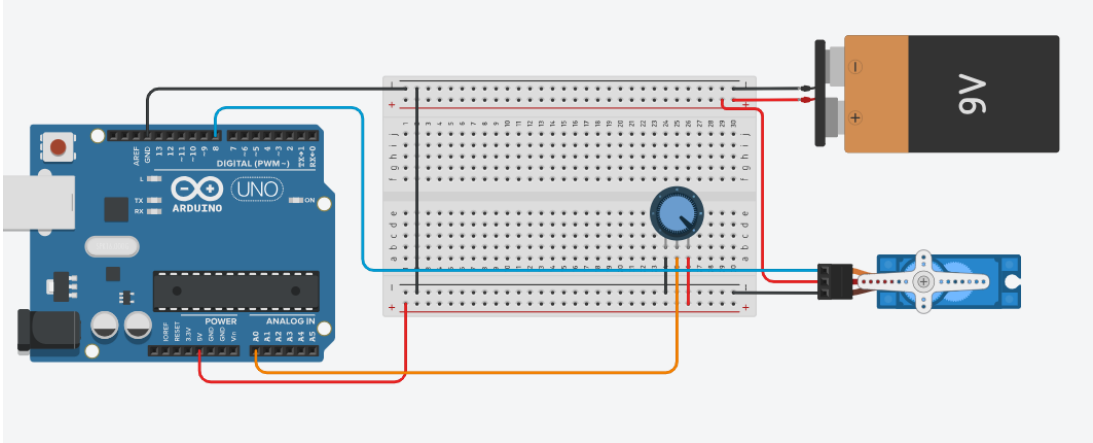


Fonte: Autor (2023).

Para utilizar um servo motor no Arduino, assim como o potenciômetro, é preciso conectar três fios. O servo 9g usa o seguinte padrão: vermelho(VCC), marrom (GND) e laranja (sinal). Devido ao consumo de corrente do servo, o mesmo demanda uma fonte de energia externa para funcionar corretamente. Para isso, utiliza-se uma bateria,pois apesar da tensão de funcionamento desses componentes ser entre 4,8 V (sem carga) e 7,2 V, a bateria de 9 V se mostrou um desempenho superior. Então, a

montagem ficou conforme a Figura 27. Observe que o GND da bateria está conectada ao do Arduino para o fechamento do circuito.

Figura 26 – Cancela acionada via potenciômetro.



Fonte: Autor (2023).

Para a programação, primeiro incluiu-se a biblioteca <Servo.h> que contém a classe Servo utilizada para se trabalhar com esse componente. Em seguida foi feita uma instância da classe Servo denominada meuServoMotor. Definiu-se "apelidos" para os pinos de sinal, tanto do servo quanto do potenciômetro. Na função *setup()*, foi realizada a inicialização do servo, utilizando o método *attach()*.

```
#include <Servo.h>

Servo meuServoMotor; // Cria um objeto da classe Servo

#define potenciometro  A0 // Pino analogico usado pelo potenciometro
#define pinServo 8 // Pino digital usado pelo servo motor

void setup() {
  meuServoMotor.attach(pinServo); // Inicializa o servo motor
}
```

No *loop()*, o valor do potenciômetro é lido e armazenado na variável do tipo inteiro, valorDoPotenciometro. No entanto, ela é lida em termos de tensão e não de ângulo, portanto não deve ser atribuída ao servo diretamente, para fazer isso é necessário utilizar a função *map()* nativa do Arduino. Ela é uma função utilizada para realizar uma conversão linear entre um intervalo de entrada e um intervalo de saída. Ela possui a seguinte sintaxe geral: *map(valor, valorMinimoEntrada, valorMaximoEntrada, valorMinimoSaida, valorMaximoSaida)*.

Com ela é possível converter o valor de tensão proveniente da porta analógica, equivalente em graus, que, em seguida, pode ser atribuído ao servo, por meio do método *write()* do servo e não o convencional *digitalWrite()*, no caso: *meuServoMotor.write(angulo)*.

```
void loop() {
  int valorDoPotenciometro = analogRead(potenciometro);  // Le o valor
   do potenciometro
  int angulacao = map(valorDoPotenciometro, 0, 1023, 0, 180);  // Mapeia
    o valor lido para o angulo do servo (0 a 180)

  meuServoMotor.write(angulacao); // Move o servo para a posicao
   desejada

  delay(15); // Pequeno atraso para evitar oscilacoes rapidas
}
```

4.4 BRAÇO ROBÓTICO

Para criar um braço robótico, utilizando potenciômetros, têm-se o principio idêntico ao da cancela, a grande diferença é o trabalho na hora da montagem. Como pode ser visto na Tabela 7, foram utilizados quatro servos, para que o braço robótico em questão consiga se mover em três dimensões e também movimentar sua garra.

Tabela 7 – Braço robótico controlado por potenciômetros.

Componente	**Quantidade**	**Preço**
Arduino + cabo	1	R$ 95,00
Jumpers	29	R$ 5,80
Servos	4	R$ 71,00
Potenciômetros	4	R$ 8.40
Bateria 9V	1	R$ 7,00
Protoboard 830 Pontos	1	R$ 16,90
Estrutura do braço em MDF	1	R$ 16,90
Total		**R$ 221**

Fonte: Autor (2023).

A base da programação é idêntica ao projeto da cancela, a única diferença é que ao invés de controlar um servo, agora são quatro. Para realizar a implementação, o procedimento é o mesmo, inclui a biblioteca Servo.h, instancie quatro objetos da classe Servo, defina pinos para cada um deles e defina os potenciômetros que irão controlá-los. Na função *setup()*, inicialize os servos com o método *attach()*. Não esqueça de definir variáveis com nomes diferentes.

```
#include <Servo.h>
// Cria um objeto da classe Servo para cada parte do braco
Servo meuServoMotor1;
Servo meuServoMotor2;
```

```
Servo meuServoMotor3;
Servo meuServoMotor4;

//Definindo os pinos analogicos dos potenciometros
#define potenciometro1  A0
#define potenciometro2  A1
#define potenciometro3  A2
#define potenciometro4  A3
//Definindo os pinos de sinais dos servos
#define pinServo1 7
#define pinServo2 6
#define pinServo3 5
#define pinServo4 4

void setup() {
  //Inicializando todos os servos
  meuServoMotor1.attach(pinServo1);
  meuServoMotor2.attach(pinServo2);
  meuServoMotor3.attach(pinServo3);
  meuServoMotor4.attach(pinServo4);
}
```

Na função *loop()*, declara-se variáveis inteiras **valorDoPotenciometro** para receber a leitura analógica de um pino conectado a um potenciômetro, e **angulação** que recebe o valor mapeado em graus de **valorDoPotenciometro**. Após isso realiza-se a escrita da angulação no servo utilizando o método *write()* da classe servo, em seguida é feita uma pequena parada utilizando o a função *delay()* para que a próxima leitura possa ser feita. O processo se repete para cada um dos servos declarados.

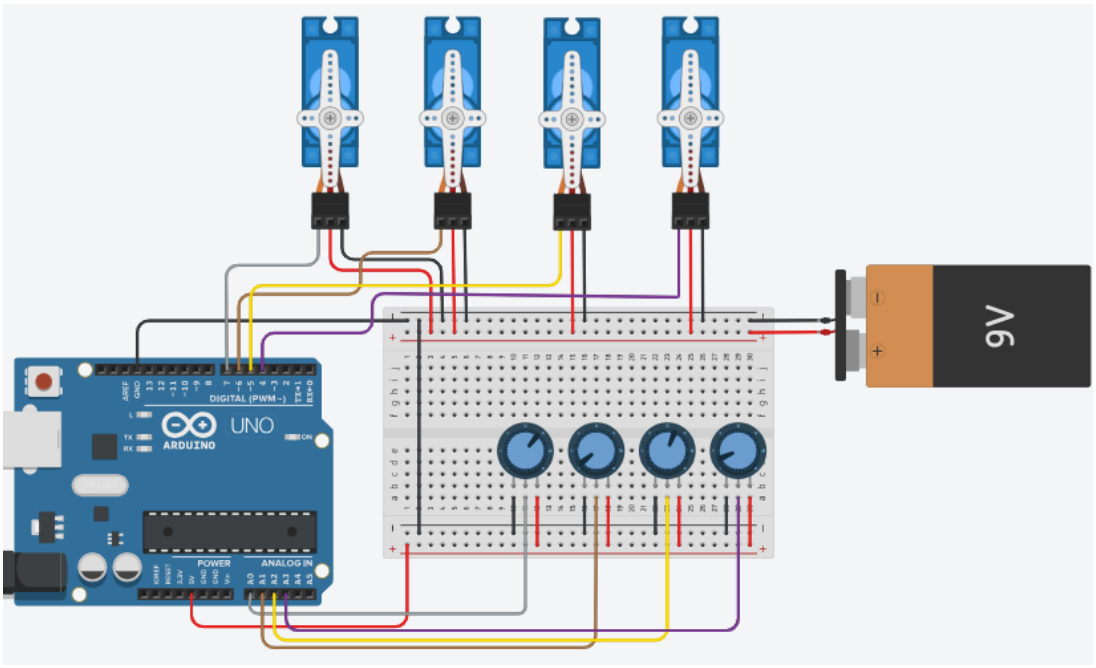
```
void loop() {
  //Repete o mesmo processo de apenas um potenciometro: leitura,
   mapeamento de valor e escrita no servo
  int valorDoPotenciometro1 = analogRead(potenciometro1);  // Le o valor
    do potenciometro
  int angulacao1 = map(valorDoPotenciometro1, 0, 1023, 0, 180);  //
   Mapeia o valor lido para o angulo do servo (0 a 180)
  meuServoMotor1.write(angulacao1);  // Move o servo para a posicao
   desejada
  delay(15);

  int valorDoPotenciometro2 = analogRead(potenciometro2);  // Le o valor
    do potenciometro
  int angulacao2 = map(valorDoPotenciometro2, 0, 1023, 0, 180);  //
   Mapeia o valor lido para o angulo do servo (0 a 180)
  meuServoMotor2.write(angulacao2);
  delay(15);
  ...
```

```
13 }
```

Uma forma de aprimorar ainda mais o projeto é tornar o braço independente de potenciômetros, criando rotinas dentro da função *loop()* por meio do uso de *delay()* e *write()*, de forma similar à apresentada no projeto do semáforo. No entanto, ao invés de da *digitalWrite()*, utilizar o método *write()* do servo, passando um valor de angulação como argumento.

Figura 27 – Braço robótico controlado por potenciômetros.

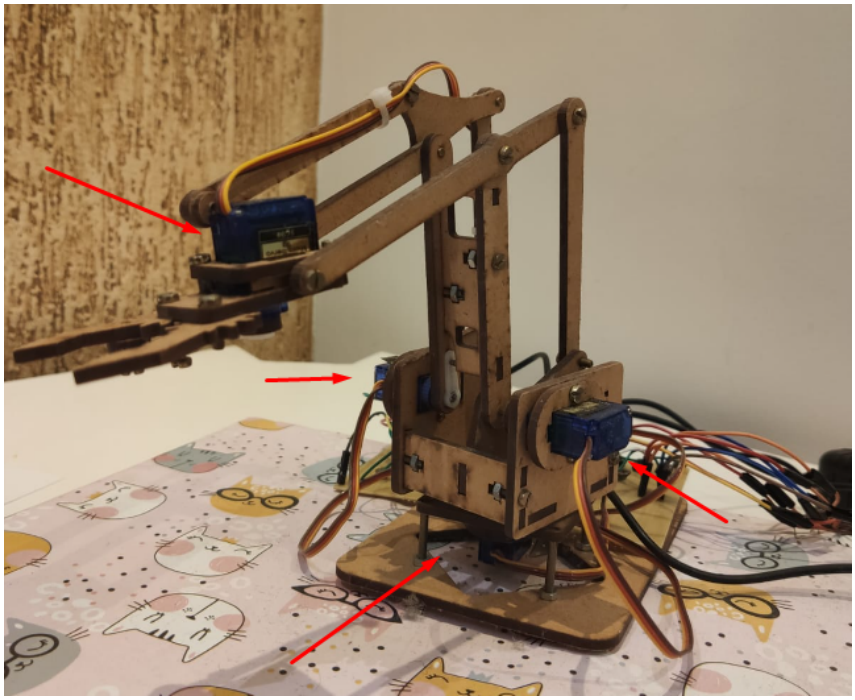


Fonte: Autor (2023).

A montagem física desse experimento é, sem dúvidas, a mais difícil dentre todos os projetos aqui apresentados, e a que demanda mais tempo. Pois, além do circuito requerer uma grande quantidade de conexões, a estrutura vem completamente desmontada, aumentando a complexidade do experimento, porém esse aspecto pode ser visto como uma oportunidade para testar habilidades como, coordenação motora e lógica. Desse modo, o braço pode ser adquirido separadamente ou construído A Figura 28 mostra o braço montado e a localização de cada servo.

Ao realizar atividades práticas com o braço robótico, como movimentar objetos, levantar pesos ou equilibrar objetos em determinadas posições, os estudantes podem adquirir um entendimento intuitivo dos conceitos de torque. Eles podem observar as consequências das variações no torque, como o impacto na estabilidade do braço e a necessidade de equilíbrio adequado para evitar quedas ou falhas no controle.

Figura 28 – Braço montado e com os servos instalados.



Fonte: Autor (2023).

4.5 ALARME

O objetivo deste projeto é atuar como um alarme de segurança. Quando algo for detectado pelo sensor de presença PIR, o buzzer deve emitir um som alertando a presença de um intruso. Mas para dificultar um pouco a construção da lógica de programação, foi adicionado ao projeto um botão e um LED. O botão deverá ligar ou desligar o alarme, e o LED deve estar acesso caso o alarme esteja ligado e apagado caso contrário. A Tabela 8 exibe todos os componentes utilizados no projeto.

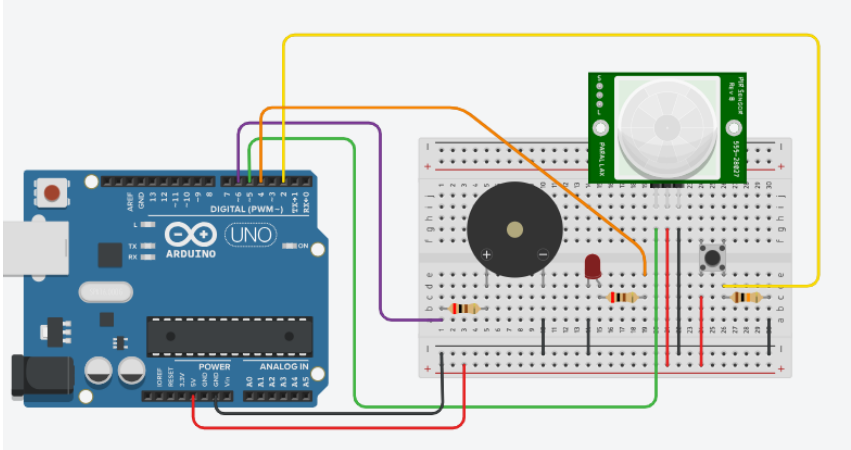
Tabela 8 – Lista de Componentes.

Componente	**Quantidade**	**Preço**
Arduino + cabo	1	R$ 95,00
Led	1	R$ 0,50
Jumpers	12	R$ 2,40
Buzzer ativo	1	R$ 2,75
Botão	1	R$ 1,80
Sensor PIR	1	R$ 12,95
Resistor 220 Ω	2	R$ 0,44
Resistor 10k Ω	3	R$ 0,20
Protoboard 830 Pontos	1	R$ 16,90
Total		**R$ 132,94**

Fonte: Autor (2023).

A montagem do circuito traz, novamente, o uso de um LED, e apresenta outra estrutura para o uso dos botões. O Buzzer, assim como o LED, precisa de um resistor de 220 Ω para ser utilizado. O sensor PIR tem seus pinos nomeados no próprio CI, então é bem simples conectá-lo. O circuito completo está exibido na Figura29.

Figura 29 – Alarme sonoro.



Fonte: Autor (2023).

Para o código, tem-se uma lógica de estados (alarme ligado e desligado) e o uso da função *loop()*. Além disso, ainda conta com estruturas de seleção e comunicação serial. Fazendo desse projeto, o mais completo em termos de programação dentre todos os outros apresentados. Abaixo tem-se as declarações de pinos e variáveis globais, e a função *setup()*. As novidades são o uso do buzzer, PIR e botão, configurados respectivamente como OUTPUT, INPUT e INPUT, além disso tem-se a inicialização da comunicação serial por meio do método begin(speed), speed nesse caso são quantos bits podem ser enviados na comunicação por segundo.

```
1  // Definicao dos pinos
2  #define pinBotao  2     // Pino do botao
3  #define pinBuzzer 6     // Pino do buzzer
4  #define pinPIR 5        // Pino do sensor PIR
5  #define pinLed 4
6
7  // Variaveis
8  int botaoApertado = 0;// Estado atual do botao
9  int alarme = 0;// Estado do alarme (ligado/desligado)
10
11 void setup() {
12   // Configuracao dos pinos
13   pinMode(pinBotao, INPUT);
14   pinMode(pinPIR, INPUT);
15   pinMode(pinBuzzer, OUTPUT);
```

```
16   pinMode(pinLed, OUTPUT);
17   Serial.begin(9600); // Inicia a comunicacao serial
18 }
```

O ponto chave na confecção do algorítimo foi a divisão de estados. Para isso era preciso primeiro coletar o valor do fator que efetuaria a troca de estado, no caso o botão. Se o mesmo for pressionado, o valor 1 será atribuído à variável global alarme, e se alarme for igual a 1, o código entra no *while(1)*, que é um loop cuja condição jamais será falsa, portanto, a única forma de sair dele é com o uso do break, que deve ocorrer apenas se o botão for pressionado novamente.

```
1 void loop() {
2   // Leitura do estado do botao
3   botaoApertado = digitalRead(pinBotao);
4   if(botaoApertado == 1){// Se o retorno da funcao for HIGH
5     alarme = 1;// Atribui 1 a alarme para indicar que esta ligado
6   }
7   delay(400);
8   if(alarme == 1){ //Se o alarme estiver ligado entra em um loop while
9     while(1){//Colocar 1 no argumento torna o while
10 }
```

Dentro do *while()*, ou seja, dentro do estado em que o alarme está ligado, primeiro é a mensagem "ligado" é exibida no monitor serial, em seguida acende-se o LED para indicar que o alarme está ligado. Por fim, armazena-se o valor da leitura do sensor PIR, e logo depois verifica se algo foi detectado, em caso afirmativo faz-se soar o alarme (liga o buzzer) e escreve "detectou" no monitor serial, utilizando o método *println()*, que aceita como argumento Strings (texto), int e float. No fim do escopo do *while()* desliga-se o buzzer, um delay não se faz necessário pela característica do sensor PIR, de manter a detecção por um certo período de tempo.

```
1   if(alarme == 1){ //Se o alarme estiver ligado entra em um loop while
2     while(1){//Colocar 1 no argumento torna o while infinito, ate que a
    estrutura break seja chamada para quebra-lo.
3       Serial.println("ligado");//Imprime a saida serial no monitor
    serial apenas para fins de indicacao
4       digitalWrite(pinLed,HIGH);// led que mostra fisicamente se o botao
     foi pressionado
5       int detectou = digitalRead(pinPIR);
6         if(detectou == 1){//verifica se o sensor detectou algo, ou seja,
     recebeu um sinal HIGH
7           Serial.println("Detectou");//Texto indicativo no monitor
    serial
8           digitalWrite(pinBuzzer, HIGH); // Liga o buzzer, faz soar o
    alarme
9         }
10      botaoApertado = digitalRead(pinBotao);
```

```
11        if(botaoApertado == 1){// Caso o botao tenha sido pressionado o
      while sera quebrado, e o fluxo do codigo voltara para o fluxo normal
      de loop()
12            break;
13        }
14        digitalWrite(pinBuzzer, LOW);  // Desliga o buzzer, para o alarme
15      }
16    }
```

Fora do *while()*, é o estado de alarme desligado, que desliga o LED, monitora se o botão foi pressionado e exibe no monitor serial a mensagem "desligado". O programa em toda sua integridade encontra-se no Apêndice E.

```
1  void loop() {
2    // Leitura do estado do botao
3    botaoApertado = digitalRead(pinBotao);
4    if(botaoApertado == 1){// Se o retorno da funcao for HIGH
5      alarme = 1;// Atribui 1 a alarme para indicar que esta ligado
6    }
7    delay(400);//Faz uma pequena espera
8    alarme = 0;//torna alarme zero para evitar a entrada no while sem que
      o botao tenha sido pressionado novamente
9    digitalWrite(pinLed,LOW);//Apaga o LED
10   Serial.println("desligado");//Indicador no monitor serial de que o
      alarme esta desligado
11 }
```

4.6 AVALIAÇÃO DE CUSTOS

Se comparados ao valor dos kits LEGO, que custam na casa dos milhares, o kit completo que abrange todos os projetos apresentados é pelo menos dez vezes menor. Além disso, após avaliar os custos individuais dos projetos, constatou-se que a maior parte do custo é proveniente do Arduino. E que sensores e atuadores possuem preços bastante acessíveis.

5 CONCLUSÕES

Este trabalho teve como intuito criar um material de apoio para auxiliar na disseminação do ensino da robótica e da programação no país. Para isso, utilizou-se de uma linguagem didática que tenta se aproximar do leitor, utilizando-se de exemplos explicativos após as discussões de novos conceitos.

A programação foi, sem dúvidas, a parte mais trabalhada, abordando o básico de C++ aplicado ao Arduino. Partindo desde a formulação de algorítimos até uma linguagem de alto nível, apresentando mesmo que de forma rasa, os conceitos fundamentais para que um iniciante consiga dar os primeiros passos nessa área.

A apresentação dos componentes eletrônicos e a base de conhecimento em elétrica foram desafiadoras. Pois, na tentativa de simplificar as informações básicas necessárias para fundamentar o conhecimento, sempre apareciam termos cuja explicação eram temas de graduação, ou das séries finais do ensino médio, por exemplo, divisor de tensão e corrente, PWM, Leis de Ohm, entre outros tópicos que fugiriam do objetivo deste trabalho.

O custo do kit completo foi muito mais baixo que o preço de um da LEGO. Contudo, poderia ter sido mais barato, visto que o Arduino, que representa quase 40% do valor total do kit, pode ser trocado por um ESP32, que custa por volta de R$ 32. Ademais, vale ressaltar que não foram apresentados projetos relacionados a deslocamento, o que coloca os conjuntos LEGO em vantagem nesse aspecto.

Em projetos futuros, poderão ser abordados os componentes voltados para a mobilidade, bem como fazer uma revisão bibliográfica dos grupos de sensores e atuadores mais utilizados no mundo da robótica e automação. Além disso, poderá ser apresentado uma abordagem mais voltada para o público infantil, trabalhando linguagens como o *Scratch* ou a programação em bloco disponível no tinkercad.

Por fim, é crucial destacar a importância da aplicação prática desse material para o ensino da robótica, transmitindo-o aos professores. Ao fazer isso, será possível adaptar os experimentos a aplicações práticas vistas em sala de aula, garantindo que sejam realmente utilizáveis e eficazes no ambiente de ensino.

REFERÊNCIAS

AGUIRRE, L. A. **Fundamentos de instrumentação**. São Paulo: Pearson, 2013.

ALEXANDER, C. K.; SADIKU, M. N. O. **Fundamentos de Circuitos Elétricos**. 5. ed. Porto Alegre: AMGH, 2013. ISBN 978-85-8055-173-0.

ALIMISIS, D. Educational robotics: Open questions and new challenges. **Themes in Science Technology Education**, Greece, v. 6, p. 63–71, 2013. Disponível em: https://files.eric.ed.gov/fulltext/EJ1130924.pdf. Acesso em: 17 jun. 2023.

ARDUINO. **Datasheet Arduino Uno R3**. 2022. Disponível em: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf. Acesso em: 13 jun. 2023.

ARDUINO. **Language Reference**. 2023. Disponível em: https://www.arduino.cc/reference/en/. Acesso em: 13 jun. 2023.

ARDUINO E CIA. **Como usar um sensor de presença PIR com arduino**. 2014. Disponível em: https://www.arduinoecia.com.br/sensor-presenca-arduino-modulo-pir-dyp-me003/. Acesso em: 13 jun. 2023.

BEZERRA, V. **Buzzer ativo ou passivo. Qual é o ideal para o seu projeto?** 2021. Disponível em: encurtador.com.br/jzCX5. Acesso em: 12 Ago. 2021.

BOYLESTAD, R. L.; NASHELSKY, L. **Dispositivos eletrônicos e teoria de circuitos**. 11. ed. São Paulo: Pearson Education do Brasil, 2013.

BRASIL. **Robótica com sucata, promovendo a sustentabilidade**. 2023. Disponível em: http://basenacionalcomum.mec.gov.br/implementacao/praticas/caderno-de-praticas/ensino-fundamental-anos-finais/172-robotica-com-sucata-promovendo-a-sustentabilidade-2. Acesso em: 20 jul. 2023.

CENTRO DE ESTUDOS AVANÇADOS EM ECONOMIA APLICADA. **Participação no PIB brasileiro chega a 27.4%**. São Paulo, 2021. Disponível em: encurtador.com.br/ELQR0. Acesso em: 12 jul. 2022.

CRAIG, J. J. **Robótica**. 3. ed. São Paulo: Pearson Education do Brasil, 2012.

DEITEL, H.; DEITEL, P. **C++**: Como programar. São Paulo: Pearson Prentice Hall, 2006. Revisão técnica por Fábio Lucchini.

DEITEL, P.; DEITEL, H. **C++**: How to program. 8th. ed. USA: Prentice Hall Press, 2011.

DRUIN, A.; HENDLER, J. A. **Robots for kids**: Exploring new technologies for learning. San Francisco, CA, USA: Morgan Kaufmann, 2000.

ELETROGATE. **Micro Servo 9g SG90 TowerPro**. Belo Horizonte, 2023. Disponível em: https://www.eletrogate.com/micro-servo-9g-sg90-towerpro. Acesso em: 13 jun. 2023.

ESCOLA, E. B. **Álgebra Booleana**. Equipe Brasil. Disponível em: https://brasilescola.uol.com.br/informatica/algebra-booleana.htm.

FIROOZIAN, R. **Servo Motors and Industrial Control Theory**. [S.l.]: Springer, 2014.

GENERATION ROBOTS. **NXT-G: the development environment supplied with Lego Mindstorms, NXT-G**. 2015. Disponível em: https://www.generationrobots.com/blog/en/nxt-g-the-development-environment-supplied-with-lego-mindstorms-nxt-g/. Acesso em: 13 jun. 2023.

GERBELLI, L. G. **Falta de mão de obra qualificada afeta metade das indústrias do país. G1 Economia**. 2020. Disponível em: https://g1.globo.com/economia/concursos-e-emprego/noticia/2020/02/11/falta-de-mao-de-obra-qualificada-afeta-metade-das-industrias-do-pais.ghtml. Acesso em: 13 jul. 2022.

HELERBROCK, R. **Resistores**. 2023. Disponível em: https://brasilescola.uol.com.br/fisica/resistores.htm. Acesso em: 13 jun. 2023.

MANZANO, J. A. N. G. **Algoritmos**: Lógica para desenvolvimento de programação de computadores. 28. ed. São Paulo: Érica, 2016.

MATHIAS, I. M. **Algoritmos e programação I**. Ponta Grossa: UEPG/NUTEAD, 2017.

MICRO:BIT EDUCATIONAL FOUNDATION. **BBC micro:bit**. 2022. Disponível em: https://microbit.org/. Acesso em: 13 jul. 2022.

NUCLEO DE TECNOLOGIA EDUCACIONAL DO RIO GRANDE DO SUL. **Projeto robótica livre educacional na escola pública estadual do rio grande do sul**. 2014. Disponível em: https://nte-poa.weebly.com/projeto-roboacutetica.html. Acesso em: 13 jul. 2022.

O QUE... **O QUE o Brasil precisa para crescer e como a indústria pode ajudar.Revista Exame**. 2020. Disponível em: https://exame.com/bussola/o-que-o-brasil-precisa-para-crescer-e-como-a-industria-pode-ajudar/. Acesso em: 13 jul. 2022.

OPEN IMPULSE. **Datasheet do sensor PIR**. [S.l.], 2023. Disponível em: https://sicccIber.com.br/wp-content/uploads/2020/06/FTC-PIR.pdf. Acesso em: 17 jun. 2023.

PAPERT, S. **Mindstorms**: Children, computers, and powerful ideas. New York: Basic Books, 1980.

PORTAL DA INDÚSTRIA. **Indústria 4.0**: Entenda seus conceitos e fundamentos. São Paulo, 2022. Disponível em: https://www.portaldaindustria.com.br/industria-de-a-z/industria-4-0/. Acesso em: 13 jul. 2022.

PRIBERAM. **robô**. 2021. Disponível em: https://dicionario.priberam.org/rob\%C3\%B4. Acesso em: 13 jul. 2022.

RAS UFCG. **O Que É Um Microcontrolador?** 2020. Disponível em: https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador/. Acesso em: 23 jun. 2023.

RESNICK, M. et al. **Programmable Bricks**: Toys to think with. Massachusetts, EUA: IBM Systems Journal, 1996. 443 - 452 p. Disponível em: https://ieeexplore.ieee.org/abstract/document/5387219/metrics#metrics. Acesso em: 17 jun. 2023.

ROBOT INSTITUTE OF AMERICA. **Technical paper AD 79**. 1. ed. Detroit, Michigan: Dearborn, 1979.

VARELA, G. **Há laboratórios de informática em 81% das escolas públicas, mas somente 59% são usados**. Rio de Janeiro, 2017. Disponível em: https://epoca.globo.com/educacao/noticia/2017/08/ha-laboratorios-de-informatica-em-81-das-escolas-publicas-mas-somente-59-sao-usados.html. Acesso em: 30 jun. 2023.

VAN VLACK, L. H. **Princípios de ciência e tecnologia dos materiais**. Rio de Janeiro: Campus, 1988. 567 p.

ÜÇGÜL, M. **History and educational potential of LEGO Mindstorms NXT**. Mersin, Turquia: Mersin University Journal of the Faculty of Education, 2013. 127-137 p. Disponível em: https://dergipark.org.tr/en/download/article-file/160881. Acesso em: 27 mar. 2023.

APÊNDICE A - PISCA

```
#define LED 7
void setup()
{
  pinMode(LED, OUTPUT);//Configura o pino LED como de saida
}

void loop()
{
  digitalWrite(LED, HIGH);//Acende o LED
  delay(1000); // Espera 1000 millisecond(s), equivalente a 1 segundo
  digitalWrite(LED, LOW);//Apaga o LED
  delay(1000); //Espera 1000 millisecond(s)
}
```

Listing 1 – Blink

APÊNDICE B - SEMÁFORO

```
#define ledVermelho 7
#define ledAmarelo 6
#define ledVerde 5
void setup()
{
  //Configura os pinos dos LEDs como de saida
  pinMode(ledVermelho, OUTPUT);
  pinMode(ledVerde, OUTPUT);
  pinMode(ledAmarelo, OUTPUT);
}

void loop()
{
  //Acende o verde e apaga os outros LEDS -> Siga
  digitalWrite(ledVermelho, LOW);
  digitalWrite(ledVerde, HIGH);
  digitalWrite(ledAmarelo, LOW);
  delay(3000); // Espera 3000 millisegundos, equivalente a 3 segundos

  //Acende o Amarelo e apaga os outros -> Atencao
  digitalWrite(ledVermelho, LOW);
  digitalWrite(ledVerde, LOW);
  digitalWrite(ledAmarelo,HIGH);
  delay(1500); //Espera 1000 millisegundos

  //Acende o vermelho e apaga os outros -> Pare
  digitalWrite(ledVermelho, HIGH);
  digitalWrite(ledVerde, LOW);
  digitalWrite(ledAmarelo,LOW);
  delay(3000);//Espera 3000 milisegundos, equivalente a 3 segundos
}
```

Listing 2 – Semáforo

APÊNDICE C - CRUZAMENTO

```
#define verde1 13
#define amarelo1 12
#define vermelho1 11

#define verde2 10
#define amarelo2 9
#define vermelho2 8

void setup()
{
  pinMode(verde1, OUTPUT);
  pinMode(amarelo1, OUTPUT);
  pinMode(vermelho1, OUTPUT);

  pinMode(verde2, OUTPUT);
  pinMode(amarelo2, OUTPUT);
  pinMode(vermelho2, OUTPUT);

}

void loop()
{
  //Estado 1 - Verde do semaforo 1 ligado
  digitalWrite(verde1, HIGH);
  digitalWrite(amarelo1, LOW);
  digitalWrite(vermelho1, LOW);

  digitalWrite(verde2, LOW);
  digitalWrite(amarelo2, LOW);
  digitalWrite(vermelho2, HIGH);

  delay(3000);//tempo em sinal verde no semaforo 1

  //Estado 2 - Amarelo do semaforo 1 ligado
  digitalWrite(verde1,LOW);
  digitalWrite(amarelo1, HIGH);
  digitalWrite(vermelho1, LOW);

  digitalWrite(verde2, LOW);
  digitalWrite(amarelo2, LOW);
  digitalWrite(vermelho2, HIGH);

```

```
43    delay(1500);//tempo em sinal amarelo no semaforo 1
44    //Estado 3 - Vermelho do semaforo 1 ligado
45
46    digitalWrite(verde1,LOW);
47    digitalWrite(amarelo1, LOW);
48    digitalWrite(vermelho1, HIGH);
49
50    digitalWrite(verde2, HIGH);
51    digitalWrite(amarelo2, LOW);
52    digitalWrite(vermelho2, LOW);
53
54    delay(3000);//tempo em sinal vermelho no semaforo 1
55
56    //Estado 4 - Amarelo do semaforo 2 ligado
57    digitalWrite(verde1,LOW);
58    digitalWrite(amarelo1, LOW);
59    digitalWrite(vermelho1, HIGH);
60
61    digitalWrite(verde2, LOW);
62    digitalWrite(amarelo2, HIGH);
63    digitalWrite(vermelho2, LOW);
64
65    delay(1500);//tempo em sinal vermelho no semaforo 1
66
67 }
```

APÊNDICE D - CANCELA COM POTENCIÔMETRO

```
#include <Servo.h>

Servo meuServoMotor; // Cria um objeto da classe Servo

#define potenciometro  A0 // Pino analogico usado pelo potenciometro
#define pinServo 8 // Pino digital usado pelo servo motor

void setup() {
  meuServoMotor.attach(pinServo); // Inicializa o servo motor
}

void loop() {
  int valorDoPotenciometro = analogRead(potenciometro);  // Le o valor
   do potenciometro
  int angulacao = map(valorDoPotenciometro, 0, 1023, 0, 180);  // Mapeia
    o valor lido para o angulo do servo (0 a 180)

  meuServoMotor.write(angulacao); // Move o servo para a posicao
   desejada

  delay(15); // Pequeno atraso para evitar oscilacoes rapidas
}
```

Listing 3 – Cancela com potenciômetro

APÊNDICE E - BRAÇO ROBÓTICO COM POTENCIÔMETROS

```
#include <Servo.h>
// Cria um objeto da classe Servo para cada parte do braco
Servo meuServoMotor1;
Servo meuServoMotor2;
Servo meuServoMotor3;
Servo meuServoMotor4;

//Definindo os pinos analogicos dos potenciometros
#define potenciometro1  A0
#define potenciometro2  A1
#define potenciometro3  A2
#define potenciometro4  A3
//Definindo os pinos de sinais dos servos
#define pinServo1 7
#define pinServo2 6
#define pinServo3 5
#define pinServo4 4

void setup() {
  //Inicializando todos os servos
  meuServoMotor1.attach(pinServo1);
  meuServoMotor2.attach(pinServo2);
  meuServoMotor3.attach(pinServo3);
  meuServoMotor4.attach(pinServo4);
}

void loop() {
  //Repete o mesmo processo de apenas um potenciometro: leitura,
   mapeamento de valor e escrita no servo
  int valorDoPotenciometro1 = analogRead(potenciometro1);  // Le o valor
    do potenciometro
  int angulacao1 = map(valorDoPotenciometro1, 0, 1023, 0, 180);  //
   Mapeia o valor lido para o angulo do servo (0 a 180)
  meuServoMotor1.write(angulacao1);  // Move o servo para a posicao
   desejada
  delay(15);

  int valorDoPotenciometro2 = analogRead(potenciometro2);  // Le o valor
    do potenciometro
  int angulacao2 = map(valorDoPotenciometro2, 0, 1023, 0, 180);  //
   Mapeia o valor lido para o angulo do servo (0 a 180)
  meuServoMotor2.write(angulacao2);
```

```
37   delay(15);
38
39   int valorDoPotenciometro3 = analogRead(potenciometro3);  // Le o valor
       do potenciometro
40   int angulacao3 = map(valorDoPotenciometro3, 0, 1023, 0, 180);  //
      Mapeia o valor lido para o angulo do servo (0 a 180)
41   meuServoMotor3.write(angulacao3);
42   delay(15);
43
44   int valorDoPotenciometro4 = analogRead(potenciometro4);  // Le o valor
       do potenciometro
45   int angulacao4 = map(valorDoPotenciometro4, 0, 1023, 0, 180);  //
      Mapeia o valor lido para o angulo do servo (0 a 180)
46   meuServoMotor4.write(angulacao4);
47   delay(15);
48 }
```

Listing 4 – Braço robótico controlado por potenciômetros

APÊNDICE F - ALARME

```
// Definicao dos pinos
#define pinBotao  2     // Pino do botao
#define pinBuzzer 6     // Pino do buzzer
#define pinPIR 5        // Pino do sensor PIR
#define pinLed 4

// Variaveis
int botaoApertado = 0;// Estado atual do botao
int alarme = 0;// Estado do alarme (ligado/desligado)

void setup() {
  // Configuracao dos pinos
  pinMode(pinBotao, INPUT);
  pinMode(pinPIR, INPUT);
  pinMode(pinBuzzer, OUTPUT);
  pinMode(pinLed, OUTPUT);
  Serial.begin(9600); // Inicia a comunicacao serial
}

void loop() {
  // Leitura do estado do botao
  botaoApertado = digitalRead(pinBotao);
  if(botaoApertado == 1){// Se o retorno da funcao for HIGH
    alarme = 1;// Atribui 1 a alarme para indicar que esta ligado
  }
  delay(400);
  if(alarme == 1){ //Se o alarme estiver ligado entra em um loop while
    while(1){//Colocar 1 no argumento torna o while infinito, ate que a estrutura break seja chamada para quebra-lo.
      Serial.println("ligado");//Imprime a saida serial no monitor serial apenas para fins de indicacao
      digitalWrite(pinLed,HIGH);// led que mostra fisicamente se o botao foi pressionado
      int detectou = digitalRead(pinPIR);
        if(detectou == 1){//verifica se o sensor detectou algo, ou seja, recebeu um sinal HIGH
          Serial.println("Detectou");//Texto indicativo no monitor serial
          digitalWrite(pinBuzzer, HIGH); // Liga o buzzer, faz soar o alarme
        }
      botaoApertado = digitalRead(pinBotao);
```

```
37        if(botaoApertado == 1){// Caso o botao tenha sido pressionado o
      while sera quebrado, e o fluxo do codigo voltara para o fluxo normal
      de loop()
38            break;
39        }
40        digitalWrite(pinBuzzer, LOW);  // Desliga o buzzer, para o alarme
41       }
42    }
43    delay(400);//Faz uma pequena espera
44    alarme = 0;//torna alarme zero para evitar a entrada no while sem que
       o botao tenha sido pressionado novamente
45    digitalWrite(pinLed,LOW);//Apaga o LED
46    Serial.println("desligado");//Indicador no monitor serial de que o
       alarme esta desligado
47 }
```

Listing 5 – Alarme