Aluno: Gabriel Siqueira

Trabalho Final Automação de Sistemas

1. INTRODUÇÃO

O presente trabalho visa demonstrar as potencialidades criativas que podem ser obtidas a partir do desenvolvimento de sistemas embarcados utilizando a plataforma Arduíno e diversos outros componentes para incrementar o seu potencial. Desta forma, iniciaremos o trabalho com uma breve descrição dos componentes que serão utilizados, com o seu objetivo e o seu funcionamento, pois além de utilizá-los, devemos compreender como se comportam para produzir sistemas cada vez melhores.

Obtendo um breve conhecimento sobre o Arduíno e seus componentes, iniciaremos com um pequeno projeto no qual será desenvolvido um sistema de sensor de Temperatura, no qual compreenderemos como utilizar um display LCD juntamente com o sensor de temperatura. Após isso, criaremos um radar ultrassônico controlado por um servo motor, no qual será possível realizar a detecção de objetos num ângulo de 90 graus, além de adicionar um sinalizador para nós alertamos de obstáculos detectados. Por fim, será desenvolvido um simulador de Jukebox de 8 bits, no qual o estudante aprenderá como trabalhar com interações do usuário e fazer o seu devido tratamento.

2. EQUIPAMENTOS

2.1. Arduino

O Arduino é uma plataforma open-source de prototipagem eletrônica que apresenta hardware e software flexíveis e fáceis de utilizar, destinados a pessoas interessadas em criar objetos ou ambientes interativos. Em outras palavras, é uma plataforma formada por dois componentes, a placa que é o Hardware usado para construir os projetos e o Arduino IDE que é o software utilizado para escrever os códigos para a plataforma (ARDUINOPORTUGAL.PT, 2017)

O principal componente do Arduino é seu microcontrolador, que corresponde a um pequeno processador de computador montado em uma placa com diversos outros componentes que manipulam sua entrada e saída com o propósito de facilitar conectar o mundo físico ao mundo digital.

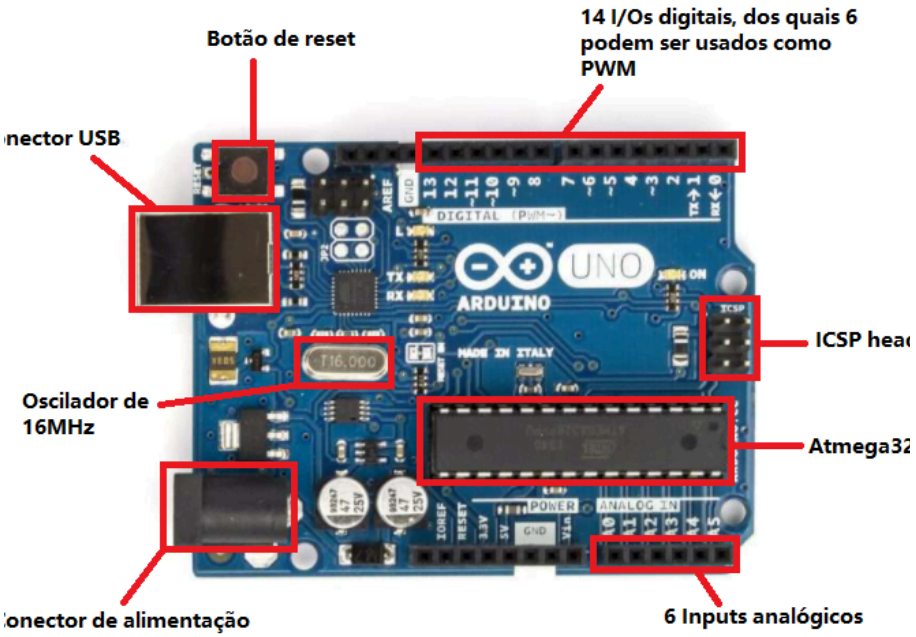


Figura: Arduíno Uno (LOUSADA, 2020)

2.1.1. **A Placa do Arduino Uno**

O Arduino Uno é uma placa que utiliza o microcontrolador ATmega328 que possui 32 kB de memória Flash (local onde o código será salvo) e 2KB de SRAM (memória no qual as variáveis estão armazenadas). Esta placa está dividida em 7 setores diferentes, cada um contendo uma funcionalidade específica, sendo eles segundo Arduinoportugal.Pt (2017)

1. **Microcontrolador:** é o cérebro do Arduíno, ou seja, o dispositivo programável que roda o código de nosso programa
2. **Conector USB:** responsável por conectar o Arduino ao computador ou alimentar a placa
3. **Pinos de Entrada e Saída:** pinos programados para agirem como entrada ou saídas de dados fazendo com que o Arduíno interaja com o meio externo
4. **Pinos de Alimentação:** fornecem diversos valores de tensão utilizados para transmitir energia elétrica aos componentes do projeto
5. **Botão de Reset:** responsável por reiniciar o dispositivo
6. **Conversor Serial-USB e LEDs TX/RX:** o conversor Serial-USB permite que o microcontrolador se comunique com o computador. Enquanto os LEDs TX e

RX acendem quando o Arduino está transmitindo e recebendo dados pela porta serial respectivamente

7. **Conector de Alimentação:** permite que uma fonte alimente a placa
8. **LED de Alimentação:** indica se a placa está transmitindo energia
9. **LED Interno:** LED ligado ao pino digital 13

Vale mencionar que normalmente as entradas analógicas são utilizadas para ler sensores externos e as saídas PWEM e outputs digitais são utilizadas para controlar motores e atuadores e adicionar drivers para cargas externas. (LOUSADA, 2020)

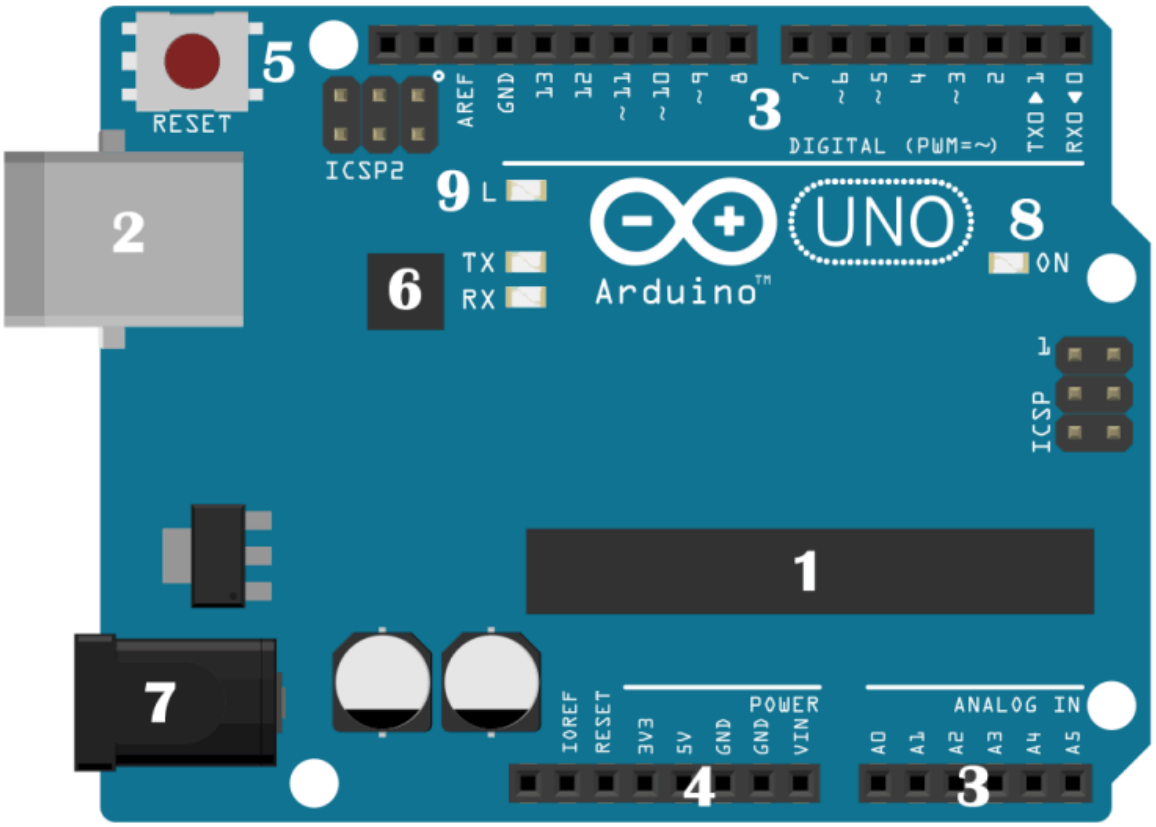


Figura: Placa de Arduíno (LOUSADA, 2020)

2.2. Sensor Ultrassônico

Os sensores de proximidade ultra sônicos são dispositivos usados para detecção sem contato de objetos em muitas áreas da indústria. Ou seja, são

aplicados como detectores de objetos, sendo muito populares na robótica, onde são utilizados para identificar obstáculos e corrigir continuamente o trajeto feito por um robô (VIDAL, 2022).

Em síntese, por meio da emissão de sinais ultrassônicos, é possível especificar à distância do sensor até um determinado objeto, no qual o range de atuação é da ordem de 4 metros, com distância mínima de 2 cm e inclui obstáculos dentre de um ângulo de abertura de 15 graus. Vale salientar que as ondas mecânicas emitidas por este sensor têm frequência acima de 40 kHz, as quais não são perceptíveis pelos humanos (VIDAL, 2022).

Algumas das aplicações que podem ser desenvolvidas utilizando este equipamento segundo Vidal (2022) são as seguintes:

- Detecção de objetos e verificação de presença
- Medição de altura e largura
- Medição de níveis de enchimento
- Detecção de obstáculos
- Correção de rota de robôs e outros mecanismos móveis como carros de controle remoto

Figura: Sensor Ultrassônico HC-SR04 (VIDAL, 2022)

2.2.1. Funcionamento

O seu funcionamento está baseado na emissão de uma onda sonora de alta frequência, e na medição do tempo que leva para a recepção do eco, produzido quando a onda se encontra com um objeto capaz de refletir o som. Basicamente,

este sensor funciona a partir da medição do tempo de propagação do eco, isto é, o intervalo de tempo entre o impulso sonoro emitido e o eco recebido de volta. (Sense, 2024).

Durante sua execução, o sensor emite pulsos de ultrassom ciclicamente. Quando um objeto reflete estes pulsos, o eco resultante é recebido e convertido num sinal elétrico. Vale salientar que a detecção do eco incidente, depende de sua intensidade e da distância do objeto e do sensor ultrassônico. Sua construção faz com que o feixe seja emitido em forma de cone e somente objetos no raio do cone são detectados (INSTRUMENTS, 2024)

Para sua utilização no Arduíno, o mesmo possui a denominação de *sensor HC-SR04*, o qual segundo Vidal (2022) é constituído de 3 partes:

- **Transmissor Ultrassônico:** emite as ondas ultrassônicas que serão refletidas pelos obstáculos
- **Um receptor:** identifica o eco do sinal emitido pelo emissor
- **Circuito de Controle:** controla o conjunto transmissor/receptor, calculando o tempo entre a emissão e recepção do sinal

Além de especificar os componentes, Vidal (2022) específica em três passos como ocorre a comunicação do sensor com o Arduíno, sendo da seguinte forma:

1. O circuito externo (Arduino) envia um pulso de trigger de pelo menos 10us ao nível alto;
2. Ao receber o sinal de trigger, o sensor envía 8 pulsos de 40khz e detecta se há algum sinal de retorno ou não;
3. Se algum sinal de retorno for identificado pelo receptor, o sensor gera um sinal de nível alto no pino de saída cujo tempo de duração é igual ao tempo calculado entre o envio e o retorno do sinal ultrassônico;
4. O cálculo da distância do objeto ao sensor é realizado da seguinte forma: Distância = (Tempo de duração do sinal de saída x velocidade do som) / 2

2.2.2. Servo Motor

O Servo Motor é um dispositivo eletromecânico utilizado para movimentar, com precisão, um objeto, permitindo-o girar em ângulos ou distâncias específicas, com garantia de posicionamento e velocidade. Basicamente, consiste num motor elétrico acoplado a um sensor que passa a condição de seu posicionamento, permitindo o controle preciso de velocidade, aceleração e da posição angular. Além disso, não possui uma rotação livre e de forma contínua como um motor

convencional, ao obedecer comandos estabelecidos. Podendo ser de corrente contínua ou de corrente alternada (CRAVO, 2024).

Segundo Kollmorgen (2024);

> Um servomotor é um dispositivo eletromecânico que produz torque e velocidade com base na corrente e na tensão fornecidas. Um servomotor funciona como parte de um sistema de malha fechada, fornecendo o torque e a velocidade comandados por um servo controlador e utilizando um dispositivo de feedback para fechar a malha de controle. O dispositivo de feedback fornece informações como corrente, velocidade ou posição para o servo-controlador, que ajusta a ação do motor de acordo com os parâmetros comandados.

Em outras palavras, Mattede (2024) menciona que os servomotores são atuadores projetados para aplicações onde é necessário fazer o controle do movimento com posicionamento de alta precisão, reversão rápida e de alto desempenho.

Uma das principais diferenças dos servos motores com outros tipos de motores é a presença de um encoder e um controlador. No qual o primeiro é um sensor de velocidade que possui a função de fornecer a velocidade e posicionamento do motor, enquanto o segundo é usado para controlar a velocidade e a posição final do motor, pois o Servo Motor trabalha com servomecanismos que utilizam feedback (realimentação) de posição. Basicamente, combina internamente um motor com circuito de realimentação, controlador e outros circuitos complementares (MATTEDE, 2024).

Entre as suas mais diversas aplicações, podemos destacar a sua utilização em sistemas de coordenadas e braços robóticos, drones, automação industrial, aeromodelos de helicópteros e aviões, nos ramos aeroespacial, agrícola, defesa, medicina entre outros(CRAVO, 2024)

Figura: Servo Motor (MATTEDE, 2024)

2.2.3. Buzzer

O buzzer é um dispositivo eletroacústico que converte sinais elétricos em ondas sonoras audíveis, ou seja, são utilizados para emitir alertas, notificações ou sinais sonoros numa variedade de dispositivos eletrônicos. Basicamente, produzem um som característico, com um zumbido ou um tom, dependendo do tipo e da aplicação específica (GUSE, 2024)..

Figura: Buzzer (GUSE, 2024)

2.2.3.1. Funcionamento

Em sua essência, o buffer funciona convertendo sinais elétricos em ondas sonoras audíveis. O qual consiste em um **transdutor piezoelétrico**, o qual é um material capaz de gerar vibrações mecânicas quando uma corrente elétrica é aplicada a ele. Ou seja, quando um sinal elétrico é enviado para o transdutor piezoelétrico dentro do buzzer, ele vibra em uma frequência específica, produzindo o som desejado. Estas vibrações são então amplificadas pela caixa do buzzer, aumentando sua intensidade e tornando o som audível. Vale salientar que dependendo do design e da aplicação do buzzer, ele pode produzir diferentes tipos de sons, como zumbidos, tons contínuos ou pulsantes (GUSE, 2024).

2.2.3.2. Características Sonoras do Buzzer

Segundo Guse (2024), estas são as principais características sonoras dos buzzers:

- **Frequência:** número de ciclos por segundo de uma onda sonora, a qual é medida em hertz (Hz).
- **Tom:** é a qualidade do som produzido pelo buzzer e está diretamente relacionado à sua frequência. No qual, buzzers com frequências mais altas

tendem a produzir tons agudos, enquanto os com frequência mais baixa geram tons mais graves;

- **Volume:** indica a intensidade do som emitido, sendo medido em decibeis (db)
- **Duração do Som:** refere-se ao tempo que o som emitido pelo buzzer persiste após sua ativação
- **Qualidade Sonora:** refere-se a fidelidade e clareza do som produzido pelo buzzer;

2.2.4. Display LCD 16×2

O display LCD (Liquid Crystal Display) é um dispositivo que permite criar interfaces gráficas entre homem e máquina, de forma simples e barata. Com ele, é possível enviar textos, números, símbolos e até imagens que podem indicar o comportamento atual do microcontrolador, dos dados a serem transmitidos, coletados, entre outras informações que o desenvolvedor desejar. Para ser possível mostrar as informações em sua tela, o mesmo apresenta leds em sua parte inferior, os quais são os principais responsáveis pelo consumo do componente (MURTA, 2023).

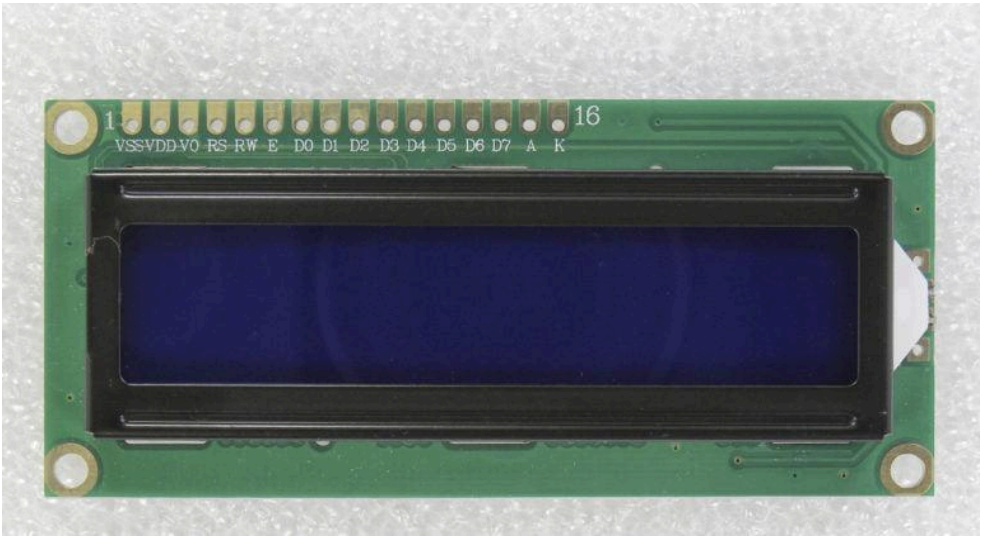


Figura: Display LCD 16x2 (MURTA, 2023)

2.2.4.1. Funcionamento

Nos displays LCD, o seu mostrador é formado por duas placas acrílicas transparentes, contendo entre elas o cristal líquido. Basicamente, o cristal líquido altera o seu comportamento cristalino, dependendo da tensão elétrica aplicada a ele. Desta forma, os displays são formados de vários pontos, onde cada um pode

ficar claro ou escuro, dependendo da polarização elétrica de cada um. Ou seja, sob as placas transparentes existe uma matriz invisível de conexões que controlam estes pontos, sendo os responsáveis por este comportamento os chips controladores (MURTA, 2023).

Vale salientar que a comunicação entre o Controlador de LCD e o microcontrolador pode ser paralela (4 ou 8 bits) ou serial (I2C), no qual se você tiver um número limitado de portas digitais é recomendado o uso da interface I2C. Além disso, o chip do display contém uma ROM interna na qual são armazenados alguns caracteres e símbolos, porém se você desejar gerar seus próprios símbolos, existe uma memória interna para isso (MURTA, 2023).

2.2.4.2. Pinagem do Display LCD

Para ocorrer o funcionamento correto do display e a sua comunicação com o microcontrolador, será necessário compreender o funcionamento de cada pino do dispositivo. Desta forma, a partir de Murta (2023), temos os seguintes pinos disponíveis:

PINO	Nome	Definição
1	VSS	Pino de alimentação (zero volts – GND)
2	VDD	Pino de alimentação de +5V
3	VO	Pino de ajuste do contraste do LCD – depende da tensão aplicada (ajustável)
4	RS	Seleção de Comandos (nível 0) ou Dados (nível 1)
5	R/W	Read(leitura – nível 1) / Write (escrita – nível 0)
6	E	Enable (Ativa o display com nível 1 ou Desativa com nível 0)
7	D0	data bit 0 (usado na interface de 8 bits)
8	D1	data bit 1 (usado na interface de 8 bits)
9	D2	data bit 2 (usado na interface de 8 bits)
10	D3	data bit 3 (usado na interface de 8 bits)
11	D4	data bit 4 (usado na interface de 4 e 8 bits)

12	D5	data bit 5 (usado na interface de 4 e 8 bits)
13	D6	data bit 6 (usado na interface de 4 e 8 bits)
14	D7	data bit 7 (usado na interface de 4 e 8 bits)
15	A	Anodo do LED de iluminação (+5V CC)
16	K	Catodo do LED de iluminação (GND)

-

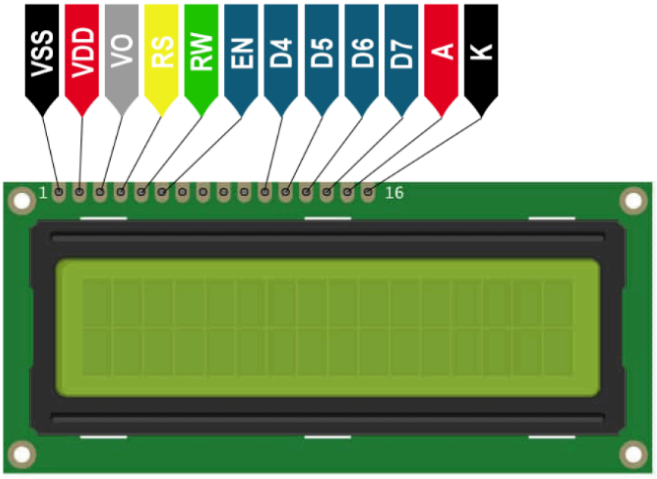


Figura: Pinagem do Display LCD (MURTA, 2023)

Além de saber quais são os pinos e sua função devemos saber com realizar a sua conexão com o Arduino. Desta forma, o pino 15 poderá ser conectado diretamente em +5 V e o pino 16 deverá ser conectado no Terra (GND). Dessa forma, o Led consumirá aproximadamente 22 mA. O consumo total de corrente (LCD + LED) do LCD 16x2 azul é de aproximadamente 23 mA, porém se for considerar o backlight (luz de fundo), o consumo elevará entre 40 e 120 mA. Caso desejar ajustar o contraste do LCD, a tensão do pino 3 deve ser ajustada, para isso poderá utilizar um potenciômetro de 20K ohms. , sendo que nas extremidades do potenciômetro conecte o +5V e o GND, enquanto no pino central conecte o pino 3 do LCD (MURTA, 2023)

2.2.5. Sensor de Temperatura

Os sensores de temperatura são dispositivos responsáveis pela medição da temperatura a partir de uma mudança em outra característica física, como radiação

térmica, resistência elétrica ou campo magnético. Desta forma, existem diversos tipos de sensores de temperatura diferentes, no qual segundo E-Racing (2022) os principais são:

- **Termopares:** são constituídos a partir de um par de fios mecânicos, acoplados em uma extremidade. A partir da diferença de tensão entre as pontas, será possível indicar a diferença de temperatura que existe entre ambas as partes (efeito Seebeck) para obter a temperatura. Este é muito utilizado em aplicações industriais e automotivas.
- **RTDs (Detectores de temperatura de resistência):** são caracterizados por um enrolamento de fios feitos normalmente de cobre, níquel e platina. Para a medição, caso a temperatura mudar, a resistência de qualquer metal mudará, e assim é possível calcular a temperatura desejada por apresentar um formato de mola, quando aquecido, apresenta variação na temperatura ambiente
- **Termistores:** caracterizados como um dispositivo semicondutor, ou seja, equivalência entre os limites da resistência elétrica e da temperatura. São divididos em dois tipos:
    - **NTC (Coeficiente de Temperatura Negativo):** tem sua resistência diminuída conforme o aumento da temperatura. São geralmente feitos de cerÂmica ou polímeros, com média de operação entre −50 °C a 300 °C
    - **PTC (Coeficiente de Temperatura Positivo):** tem sua resistência aumentada conforme o aumento da temperatura. São geralmente feitos de materiais cerâmicos poli cristalizados, com média de operação entre −100 °C a 300 ºC.

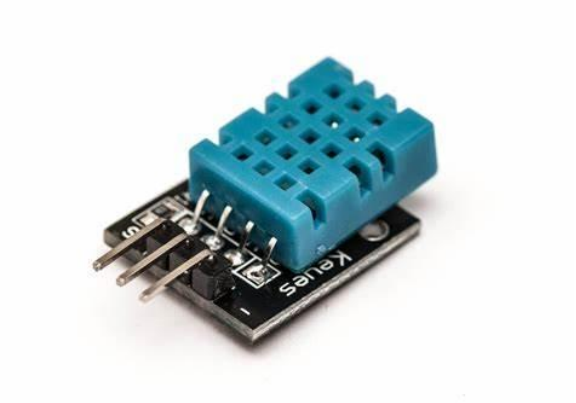


Figura: Sensor de Temperatura (E-RACING, 2022)

2.2.6. Led

O LED (Light Emitting Diodes - Diodos Emissores de Luz) são dispositivos semicondutores que convertem energia elétrica diretamente em luz visível. São construídos de materiais semicondutores dopados, que emitem fótons quando uma corrente elétrica os atravessa. Sua estrutura é composta por um ânodo (terminal positivo) e um cátodo (terminal negativo), contendo uma junção semicondutora entre eles (GUSE, 2024).

Em outras palavras, o LED é um componente eletrônico semicondutor composto de cristal semicondutor de silício ou germânio, que possuem a capacidade de transformar energia em luz. No qual a emissão de luz acontece quando a corrente elétrica percorre o material de junção PN (diodo semicondutor) emitindo radiação infravermelha (MATTEDE, 2024)

Figura: LED (GUSE, 2024)

2.2.6.1. Funcionamento

Para o funcionamento dos LEDs, o fenômeno base é a eletroluminescência, no qual a emissão de luz é gerada quando uma corrente elétrica passa por um material semicondutor. Com base nisso, quando uma tensão é aplicada aos terminais do led, elétrons do cátodo e do anodo são injetados na região de junção, no qual os elétrons se recombinam com as lacunas, liberando energia na forma de fótons de luz. Vale salientar que a cor da luz emitida depende do material utilizado para o semicondutor. (GUSE, 2024)

Além disso, a polarização dos LEDs são cruciais para o seu funcionamento adequado, a fim de determinar a direção do fluxo da corrente elétrica e como consequência da emissão de luz. Desta forma, quando uma tensão elétrica é aplicada nos terminais do LED no sentido correto (ânodo conectado ao potencial positivo e o catodo no negativo), os eletros fluem através do semicondutor, liberando a energia no formato de luz. Porém, se a polarização é invertida, o LED atua como

um dispositivo de bloqueio, impedindo o fluxo da corrente e como consequência, não emitindo luz. (GUSE, 2024)

Vale salientar que para evitar danos ao LED devido ao excesso de corrente, é essencial incluir um resistor limitador de corrente em série com o LED, garantindo uma operação segura e estável do componente. (GUSE, 2024)

2.2.7. Push Button

O botão Push Button é um tipo de interruptor simples utilizado para acionar ou interromper um circuito elétrico. Ao ser pressionado, ele fecha um circuito elétrico, enviando um sinal elétrico para um dispositivo ou sistema, iniciando uma ação específica. Ao liberar, o circuito é aberto novamente, interrompendo a ação. Em outras palavras, o botão apresenta contatos elétricos normalmente separados, no qual ao pressioná-lo, o mecanismo interno move esses contatos elétricos, permitindo que eles se toquem e que a corrente flua. (VIANA, 2020)

Figura: Botão com Push Button (VIANA, 2020)

2.2.7.1. Funcionamento

Seu funcionamento é muito similar a um interruptor elétrico, fechando ou abrindo o circuito elétrico. Porém, sua principal distinção é que a força para acionar o botão é sempre exercida no mesmo sentido, enquanto que a força para acionar um interruptor varia em função do estado atual e do estado pretendido. Além disso, muitas vezes estes botões apresentam um mecanismo que ele volta ao seu estado de repouso quando a força de atuação é removida. (LOUSADA, 2024)

2.2.8. Protoboard

A Protoboard é uma base de plástico repleta de orifícios conectados eletricamente, onde os componentes eletrônicos podem ser inseridos e interligados para formar circuitos. Desta forma, seu principal objetivo é permitir a criação de circuitos de forma rápida e fácil, sem a necessidade de soldagem, facilitando a

modificação e ajustes do circuito durante o processo de desenvolvimento (MAKIYAMA, 2024). Ou seja, é uma placa de teste que permite a montagem de circuitos eletrônicos temporários, fornecendo uma matriz de furos e linhas condutoras para facilitar a inserção e a interconexão de componentes eletrônicos (MATTEDE, 2024)

Em ambientes educacionais, a protoboard é utilizada para ensinar os princípios da eletrônica, por permitir a montagem e teste de circuitos simples, enquanto no ambiente profissional é utilizado para prototipagem rápida (MAKIYAMA, 2024)

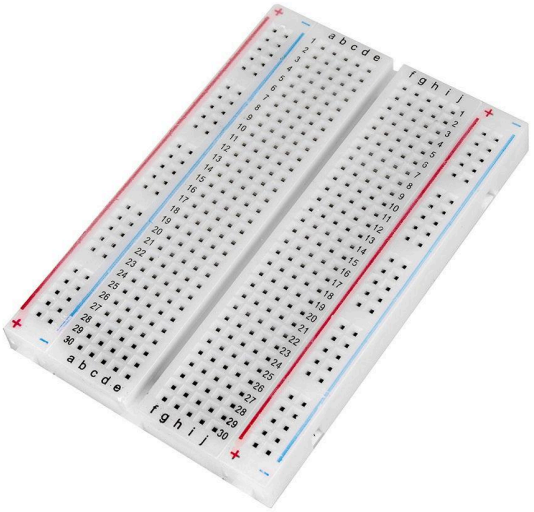


Figura: Protoboard (MAKIYAMA, 2024)

2.2.8.1. Funcionamento

A superfície da protoboard é dividida em várias linhas de orifícios, os quais são interligados eletricamente em filas horizontais ou verticais, dependendo da área da placa. Enquanto na parte central ou área de conexão é inserido os componentes. Vale mencionar que as filas horizontais são conectadas entre si, o que permite que os pinos dos componentes se conectem sem a necessidade de fios externos, no qual as trilhas verticais são independentes, facilitando a alimentação de diferentes partes com diferentes tensões. De outro modo, as áreas laterais, ou trilhas de alimentação, são utilizadas para distribuir as tensões de alimentação ao longo da protoboard (MAKIYAMA, 2024).

2.3. Ferramentas

2.3.1. Arduino IDE

O Arduino Integrated Development Environment (Arduino Software — IDE) é um editor de texto para escrever códigos, uma área de mensagem, um console, uma barra de ferramentas com funções comuns e uma série de menos a fim de facilitar no desenvolvimento de programas para o Arduíno. Corresponde a um software open-Source, com ambiente gráfico desenvolvido em Java e baseado em Processing e outras linguagens Open-Source *(LOUSADA, 2020)*.

A linguagem de programação utilizada para escrever os códigos para o Arduino é baseada nas tradicionais C/C++ com algumas modificações e possui um grau de abstração muito alto, com uma série de bibliotecas que encapsulam a maioria da complexidade do microcontrolador (LOUSADA, 2020).

Para baixar essa linguagem, basta acessar a página oficial e fazer o download.

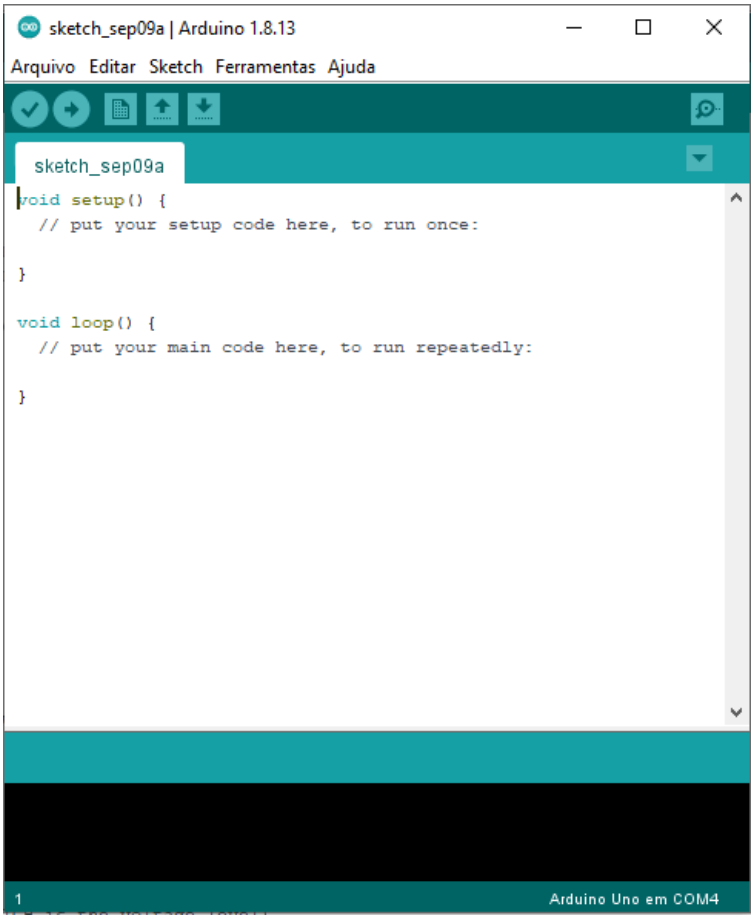


Figura: Tela principal do Arduíno IDE (LOUSADA, 2020)

2.3.2. Processing

O processing é uma linguagem de programação open-source destinada para designers e ilustradores, sendo um subset da Linguagem Java, a fim de facilitar a produção de gráficos, animações, interações, entre outras situações ((FARACO, 2013).

Em relação a seu funcionamento, o código do processing consiste em duas partes principais, blocos de configuração e desenho. O primeiro é executado uma vez quando o código é executado e o segundo é executado continuamente. A sua principal ideia é que ao escrever no bloco de desenho o mesmo será executado 60 vezes por segundo, de cima a baixo, até que o seu programa termine (GELAL, 2024).

Para baixar esta ferramenta, basta acessar a página oficial

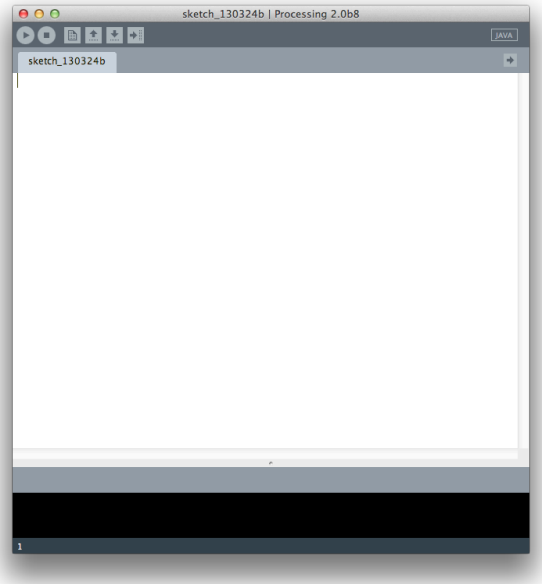


Figura: Tela principal do Processing (GELAL, 2024)

2.3.3. TinkerCad

O TinkerCad é uma plataforma web que combina modelagem 3D e simulação de circuitos eletrônicos, com uma interface amigável e acessível. Com ela é possível

criar modelos 3D facilmente, utilizando formas geométricas básicas que podem ser combinadas para criar designs mais complexos (GUSE, 2024)

Também é utilizada para projetos de impressora 3D e prototipagem rápida, como projetos de Arduino. Ou seja, além de ser uma ferramenta de modelagem, possui um módulo para simulação de circuitos eletrônicos que permite aos usuário projetar e simular circuitos eletrônicos digitais e analógicos intuitivamente. Desta forma, é possível adicionar componentes eletrônicos como resistores, capacitores, LEDs, microcontroladores e conectá-los para criar circuitos funcionais (GUSE, 2024).

Para começar a utilizar a ferramenta, basta acessar o site principal e efetuar o seu cadastro.

3. PREPARAÇÃO DO AMBIENTE

Para realizar o desenvolvimento dos projetos descritos neste trabalho, é necessário seguir as etapas abaixo:

3.1. Aquisição de Hardware (opcional dependendo do projeto)

- Arduino: Você precisará de um microcontrolador Arduino. Existem diferentes modelos, como o Arduino Uno, Arduino Nano e Arduino Mega. A escolha do modelo dependerá das necessidades do seu projeto (por exemplo, número de pinos digitais/analógicos, memória, etc.). O Arduino Uno é uma boa opção inicial devido à sua simplicidade e popularidade.
- Protoboard (Breadboard): Uma protoboard será necessária para realizar as conexões dos componentes eletrônicos de forma prática e sem a necessidade de solda.
- Componentes Eletrônicos: Além da protoboard, será necessário adquirir os componentes eletrônicos básicos como resistores, LEDs, sensores, fios de conexão (jumper wires), potenciômetros, etc., conforme o projeto.

3.2. Alternativa: Uso de Simuladores Online

- Caso você prefira não adquirir o hardware físico, uma excelente alternativa é utilizar o Tinkercad, um simulador online gratuito que permite simular circuitos e programar microcontroladores Arduino.
- Para usar o Tinkercad:
    - Crie uma conta gratuita no site do Tinkercad.

- Após criar a conta, acesse a seção de Circuitos onde você pode montar seus projetos de forma virtual, conectando os componentes e programando o Arduino diretamente no simulador.

3.3. Instalação do Software para Programação

- Arduino IDE: Para programar o Arduino, é necessário instalar o Arduino IDE em seu computador. O software pode ser baixado diretamente do site oficial do Arduino. Ele é utilizado para escrever, compilar e carregar os códigos (sketches) para o seu Arduino.
- Processing (se necessário): Caso o projeto envolve visualização de dados ou controle avançado via computador (como no projeto de Radar, por exemplo), será necessário instalar o Processing. O Processing é uma plataforma de código aberto para programação de gráficos interativos e comunicação com o Arduino.

3.4. Configuração do Ambiente

- Após a instalação do Arduino IDE e do Processing (se aplicável), verifique se a placa Arduino e a porta de comunicação estão corretamente configuradas no IDE. Isso pode ser feito em Ferramentas > Placa e Ferramentas > Porta.

3.5. Testes e Primeiros Projetos

- Ao concluir a configuração, é interessante testar o Arduino com um projeto simples, como o pisca-pisca com LED, para garantir que o ambiente esteja funcionando corretamente. Isso ajuda a familiarizar-se com o processo de escrita, compilação e upload do código.

4. DESENVOLVIMENTO

Quando desejamos dar os primeiros passos no aprendizado de uma nova tecnologia, precisamos primeiro compreender quais são os componentes disponíveis, a estrutura da linguagem e como podemos implementar a lógica de programação para criar nossos projetos.

O Arduino não é diferente, necessitamos primeiro compreender a sua estrutura básica. Como é uma linguagem muito próxima ao C e C++, acaba facilitando para quem já possui o conhecimento na mesma, mas caso você ainda não as conheça, não se preocupe, pois com o tempo irá aprender e criar projetos incríveis.

Um dos pontos mais importantes são as funções **setup** e **loop**, no qual a primeira é o local onde você irá definir os componentes/dispositivos que serão utilizados e as portas tanto digital e analógicas conectadas ao Arduíno fisicamente. Enquanto a segundo corresponde ao código que será executado infinitamente até o dispositivo for desconectado ou parado, muito parecido com a função **main** em outras linguagens.

Desta forma, conhecendo o princípio do funcionamento de um código Arduíno, na divisão em duas funções principais (setup e loop), poderemos dar início ao desenvolvimento de nosso projeto, começando com um mais simples até um avançado.

Nosso aprendizado começará com a criação de um sistema para verificação da temperatura ambiente em graus Celsius, o qual permitirá ao estudante compreender como realizar as primeiras conexões do Arduino com dispositivos externos, como o display LCD e o sensor de temperatura. Esse projeto inicial proporcionará uma base sólida para entender os fundamentos da eletrônica e programação embarcada, abordando conceitos como leitura de sensores, exibição de informações em interfaces visuais e a interação entre hardware e software. Além disso, essa etapa introdutória é essencial para desenvolver habilidades práticas que serão aplicadas em projetos mais avançados no futuro.

Após a conclusão deste projeto, avançaremos para o desenvolvimento de um radar ultrassônico, semelhante aos utilizados em submarinos e aviões para detecção de objetos próximos. Para ampliar o campo de visão e evitar a limitação a uma única direção, integrarem o sensor ultrassônico a um servo motor, permitindo a detecção de objetos em um arco de 180 graus. Adicionalmente, ao identificar a presença de um objeto, o sistema emitirá um som característico, inspirado no estilo do Batman, para alertar sobre a proximidade e a necessidade de atenção. O software Processing será utilizado para apresentar os obstáculos de forma gráfica, em uma interface similar aos sistemas empregados em submarinos para a detecção de objetos em ambientes marinhos.

Por fim, desenvolvemos um simulador de Jukebox, que permitirá ao estudante aprender a integrar músicas em estilo 8 bits com o Arduino, criando um produto funcional e desafiador. Esse projeto exige a compreensão de como armazenar e gerenciar múltiplas músicas em um único dispositivo, enfrentando as limitações de desempenho e armazenamento do Arduino. Para isso, será necessário projetar um software eficiente e bem estruturado, além de implementar

mecanismos para lidar com interrupções do usuário, como a troca de músicas ou o desligamento do dispositivo. Esse desafio proporcionará uma valiosa experiência prática na criação de sistemas embarcados otimizados e interativos.

4.1. Sistema Sensor de Temperatura

4.1.1. Objetivos

Este código foi desenvolvido para monitorar a temperatura do ambiente utilizando um sensor analógico de temperatura conectado a um Arduino e exibi-la em tempo real num display LCD de 16×2.

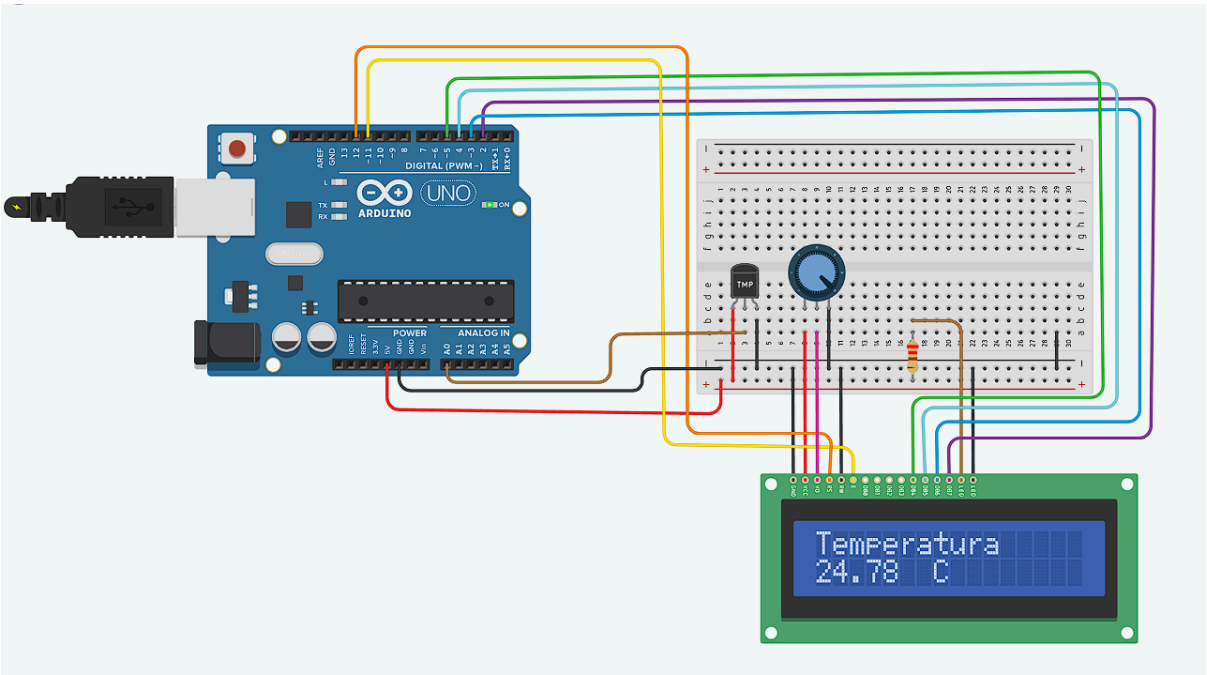
4.1.2. Cenário

Este projeto é ideal para os estudantes compreenderem como utilizar o sensor de temperatura e um display LCD. Além disso, o mesmo pode ser observado em muitos projetos de automação doméstica e industrial que demandam a medição e visualização da temperatura de um ambiente ou objeto de forma simples e acessível. Esse sistema apresenta o seguinte fluxo de execução

1. Inicializa o LCD com mensagem fixa.
2. Mede continuamente a temperatura ambiente usando o sensor.
3. Atualiza o display LCD com o valor da temperatura a cada segundo.

4.1.3. Diagrama

Para que você possa realizar a montagem deste circuito em seu Arduíno, será necessário realizar as conexões dos dispositivos conforme o diagrama abaixo:



4.1.4. Preparação do Ambiente

Para o desenvolvimento deste sistema, será necessário os seguintes componentes:

- 1 Arduino Uno ou Mega
- 1 Display LCD 16x2
- 1 Sensor de Temperatura DHT11
- 1 Protoboard
- 1 resistor de 220 ohm
- 1 Potenciômetro
- 18 jumpers de conexão

4.1.5. Código

Já em relação à implementação deste diagrama em código, o mesmo deverá ser implementado conforme abaixo em seu Arduino para o funcionamento do sistema desenvolvido. A fim de auxiliar no processo, as linhas principais foram comentadas para melhor compreensão

```
// Importar a biblioteca com as funcionalidades do Display LCD
#include <LiquidCrystal.h>

// Definir a porta que esta conectada o sensor de Temperatura
#define pin_temperatura A0
```

```
// Definição do objeto para controlar o Display LCD, com os parêmtros na seguinte ordem: RS, E, D4, D5, D6, D7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  // Informar o número de colunas e linhas do display LCD
  lcd.begin(16, 2);

  // Apresentar na primeira linha a mensagem de Temperatura, valor fixo
  lcd.print("Temperatura ");
}

float get_Temperatura(){

  // Leitura do valor analógico (0-1023) refernte ao sensor de temperatura, obtendo o valor em tensão
  int leitura = analogRead(pin_temperatura);

  // Convertendo a leitura para tensão (0 - 5V)
  float tensao = leitura * (5.0 / 1023.0);

  // Calculando a temperatura em °C
  return (tensao - 0.5) * 100.0;

}

void loop()
{
  // Definir a posição onde será apresentado o valor da temperatura, sendo na coluna 0 da linha 1 (segunda no display)
  lcd.setCursor(0, 1);

  // Apresentar o valor da leitura do sensor
  lcd.print(get_Temperatura());

  // Definir a posição onde será apresentado a unidade de medida da temperatura lida, em nosso caso em Celsius
  lcd.setCursor(7, 1);
  lcd.print("C");

  // Adicionar um delay de 1s entre as leituras
  delay(1000);
}
```

Para uma melhor compreensão das informações do código, na tabela abaixo será exemplificado quais as variáveis, bibliotecas e a função de cada uma para o correto funcionamento do código:

Elemento	Descrição	Função no Código

Bibliotecas		
<LiquidCrystal.h>	Biblioteca que permite o controle de displays LCD baseados no controlador HD44780.	Garante as funcionalidades necessárias para inicializar e manipular o display LCD.
Constantes e Variáveis		
#define pin_temperatura A0	Define o pino analógico conectado ao sensor de temperatura.	Permite a leitura dos valores do sensor na entrada analógica A0.
LiquidCrystal lcd	Configuração do display LCD com os pinos usados no controle (RS, E, D4, D5, D6, D7).	Permite a comunicação entre o microcontrolador e o display LCD para exibição de informações.
Funções		
void setup()	Função de inicialização do sistema.	Configura o LCD com 16 colunas e 2 linhas, além de exibir a mensagem fixa "Temperatura " na 1ª linha.
float get_Temperatura()	Calcula a temperatura em graus Celsius a partir da leitura analógica do sensor.	Lê o pino analógico, converte o valor para tensão e retorna a temperatura calculada.
void loop()	Função principal do programa, executada continuamente.	Lê a temperatura, exibe o valor na segunda linha do LCD com a unidade "C" e aguarda 1 segundo.

4.2. Sistema de Radar Ultrassônico

4.2.1. Objetivos

O código visa implementar um sistema de radar baseado em Arduino que detecta objetos próximos usando um sensor ultrassônico. O radar realiza leituras enquanto o servo motor gira de 0° a 180° e volta, acionando sinais luminosos e sonoros caso um objeto esteja a menos de 30 cm de distância. Além disso, os

dados de ângulo e distância são enviados pela porta serial para análise externa pelo Processing.

4.2.2. Cenário

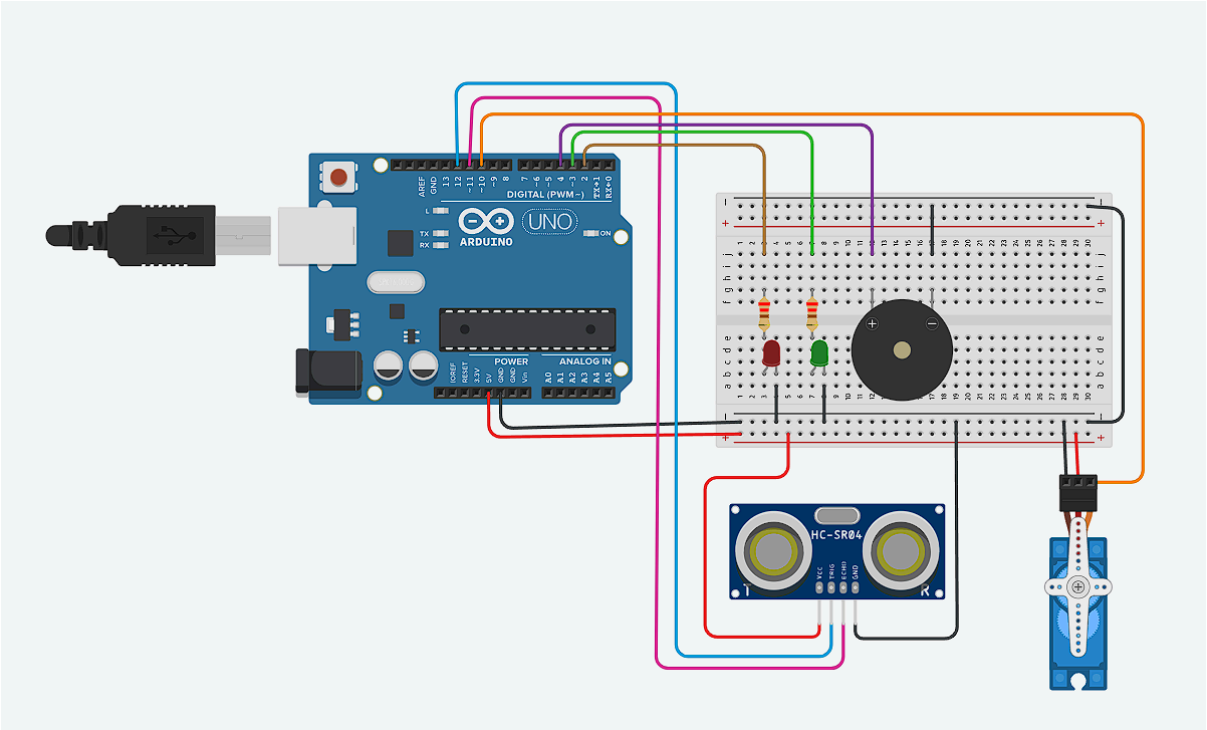
O cenário deste código pode ser imaginado em um ambiente de segurança ou monitoramento. O sistema pode ser usado para detectar intrusos ou objetos a uma certa distância e ativar uma resposta visual (LED) e sonora (buzzer) quando um objeto indesejado é detectado.

O sistema apresenta o seguinte fluxo de execução:

- O servo motor realiza um movimento contínuo de varredura (ida e volta entre 0° e 180°).
- O sensor ultrassônico mede a distância de objetos próximos.
- Se um objeto for detectado a menos de 30 cm:
    - O buzzer emite o som do "BatSinal".
    - O LED vermelho é ativado.
- Caso contrário, o LED verde é ativado.
- Os dados de ângulo e distância são enviados para a porta serial, permitindo monitoramento ou integração com outros sistemas.

4.2.3. Diagrama

Para que você possa realizar a montagem deste circuito em seu Arduíno, será necessário realizar as conexões dos dispositivos conforme o diagrama abaixo:



Lembrando que o sensor ultrassônico deverá ser instalado em cima do Servo Motor para permitir realizar o movimento em 180 graus, no qual a sua modelagem pode ser objetiva neste link para impressão na Impressora 3D:

4.2.4. Preparação do Ambiente

Para o desenvolvimento deste sistema, será necessário os seguintes componentes:

- 1 Arduino Uno ou Mega
- 1 Protoboard
- 1 Buzzer
- 1 Sensor Ultrassônico HC-SR04
- 1 servo motor
- 1 LED vermelho
- 1 LED verde
- 2 resistores de 220 Ohms
- 13 jumpers de conexão
- 1 suporte para Sensor Ultrassônico no Servo Motor, disponível no link Modelo Impressora 3D para impressão na Impressora 3D

4.2.5. Código do Arduino

Já em relação à implementação deste diagrama em código, o mesmo deverá ser implementado conforme abaixo em seu Arduino para o funcionamento do

sistema desenvolvido. A fim de auxiliar no processo, as linhas principais foram comentadas para melhor compreensão

```
#include <Servo.h>  // Importação da biblioteca para utilização do Servo Motor

#define pino_servo 10      // Definir o pino conectado o Servo Motor
#define pino_trigger 12    // Definir o pino trigger do Sensor Ultrassônico
#define pino_echo  11      // Definir o pino echo do Sersor Ultrassônico
#define pino_ledVermelho 2  // Definir o pino do LED Vermelho
#define pino_ledVerde 3    // Definir o pino do LED Verde
#define pino_buzzer 4      // Definir o pino do Buzzer


int DURACION;        // Armazenar a duração da leitura do sensor ultrassonico para realizar a conversão
int DISTANCIA;       // Armazenar a distância em cm do obtido no sensor ultrassônico com base na duração
obtida
int distance_now = 0; // Armazenar a última distância obtida no sernsor ultrassonico
Servo meuServo;      // Criar o objeto para controle do servo motor


void setup() {

  // Definir os pinos a ser utilizados e se serão de entrada ou saída
  pinMode(pino_trigger, OUTPUT);
  pinMode(pino_echo, INPUT);
  pinMode(pino_ledVermelho, OUTPUT);
  pinMode(pino_ledVerde, OUTPUT);
  pinMode(pino_buzzer, OUTPUT);

  // Definir o pino que será utilizado para conexão com o servo motor
  meuServo.attach(pino_servo);

  // Ajustar o servo motor para 0 graus
  meuServo.write(0);

  // Definir o sensor serial que será utilizado para comunicação com o processing
  Serial.begin(9600);
}

/*
* Função responsável por obter a distância do sensor ultrassonico em relação ao objeto detectado
*/
int get_distance(){

  // Realizar a ativação do pino Trigger do Sensor Ultrassônico
        digitalWrite(pino_trigger,HIGH);
        delay(1);

  // Realizar a desativação do pino Trigger do Sensor Ultrassonico
        digitalWrite(pino_trigger,LOW);

  // Calcular a duração da transmissão do pulso encaminhado, em tensão
        DURACION=pulseIn(pino_echo,HIGH);
```

```
 // Realizar a conversão da tensão em centimetros
        DISTANCIA=DURACION/58.2;

 // Retornar a distância em cm
 return DISTANCIA;
}


/*
* Função responsável por simular o BatSinal quando um objeto é detectado
*/
void tocarBatsinal() {

 // Alterna tons graves e agudos para simular o som do Batsinal
 for (int i = 0; i < 3; i++) { // Repete o som 3 vezes
  tone(pino_buzzer, 800); // Frequência grave
  delay(300);
  tone(pino_buzzer, 1200); // Frequência aguda
  delay(300);
  noTone(pino_buzzer);
  delay(100); // Pausa entre repetições
 }
}

/*
* Função responsável por desligar o LED verde e ligar o LED vermelho
*/
void liga_led_vermelho(){
 digitalWrite(pino_ledVermelho, HIGH);
 digitalWrite(pino_ledVerde, LOW);
}

/*
* Função responsável por desligar o LED vermelho e ligar o LED verde
*/
void liga_led_verde(){
        digitalWrite(pino_ledVermelho, LOW);
  digitalWrite(pino_ledVerde, HIGH);
}

/*
* Função responsável por movimentar o servo motor num determinado ângulo, efetuar a leitura da
distância do objeto mais próximo e se estiver numa distância
* inferior a 30 cm ligar o LED vermelho e soar o sinalizador do batman.
*/
void verificador_distancia(int angulo){

 // Movimentar o servo motor para o ângulo atual
        meuServo.write(angulo);

 // Realizar a leitura da distância do objeto mais próximo
 distance_now = get_distance();

 // Verificar se a distância é inferior a 30 cm ou não
 if(distance_now < 30){
  tocarBatsinal();
```

```
   liga_led_vermelho();
  }else{
   noTone(pino_buzzer);
   liga_led_verde();
  }

  // Definir a estrutura que será encaminhado para o processing
  Serial.print(angulo);
  Serial.print(",");
  Serial.println(distance_now);

  // Dar uma pausa de 30 milisegundos para a próxima leitura
  delay(30);
}


void loop() {

  // Realizar a leitura nos angulo de 0 a 180 graus
  for(int angulo = 0; angulo < 180; angulo++)
   verificador_distancia(angulo);

  // Realizar a leitura nos angulo de 180 a 0 graus
  for(int angulo = 180; angulo > 1; angulo--)
   verificador_distancia(angulo);
}
```

Para uma melhor compreensão das informações do código, na tabela abaixo será exemplificado quais as variáveis, bibliotecas e a função de cada uma para o correto funcionamento do código:

Elemento	Descrição	Função no Código
Biblioteca		
Servo.h	Biblioteca para controle de servo motor.	Permite a movimentação do servo motor para realizar a varredura do sensor ultrassônico.
Constantes e Variáveis		
pino_servo	Pino 10 conectado ao servo motor.	Controla o servo motor, ajustando sua posição de acordo com os ângulos definidos no código.
pino_trigger	Pino 12 conectado ao	Aciona o sensor

	pino trigger do sensor ultrassônico.	ultrassônico para enviar o pulso de som e realizar a medição da distância.
pino_echo	Pino 11 conectado ao pino echo do sensor ultrassônico.	Recebe o pulso de som refletido pelo objeto e calcula a distância entre o sensor e o objeto.
pino_ledVermelho	Pino 2 conectado ao LED vermelho.	Controla o LED vermelho, indicando alerta (objeto detectado a menos de 30 cm).
pino_ledVerde	Pino 3 conectado ao LED verde.	Controla o LED verde, indicando que não há objetos próximos (distância superior a 30 cm).
pino_buzzer	Pino 4 conectado ao buzzer.	Emite um som para alertar sobre a presença de um objeto próximo (simula o BatSinal).
DURACION	Variável que armazena a duração do pulso do sensor ultrassônico.	Usada para calcular a distância entre o sensor e o objeto, com base no tempo de resposta do pulso.
DISTANCIA	Variável que armazena a distância medida pelo sensor ultrassônico.	Armazena a distância em centímetros do objeto detectado pelo sensor ultrassônico.
distance_now	Variável que armazena a última distância medida pelo sensor ultrassônico.	Armazena o valor da distância mais recente para comparações subsequentes e acionar respostas adequadas.
meuServo	Objeto da classe Servo.	Controla o servo motor, permitindo movê-lo para diferentes ângulos durante a varredura.
Funções		
setup()	Função de configuração executada uma vez ao	Inicializa os pinos, configura o servo motor e

	iniciar o código.	a comunicação serial, e ajusta o servo para 0 graus.
get_distance()	Função que mede a distância usando o sensor ultrassônico.	Ativa o trigger, mede a duração do pulso no echo, calcula e retorna a distância em centímetros.
tocarBatsinal()	Função que emite o som do BatSinal.	Emite um som alternado (grave e agudo) por três vezes utilizando o buzzer.
liga_led_vermelho()	Função que acende o LED vermelho e apaga o LED verde.	Indica que um objeto foi detectado a menos de 30 cm, acionando o alerta visual vermelho.
liga_led_verde()	Função que apaga o LED vermelho e acende o LED verde.	Indica que não há objetos próximos (distância superior a 30 cm), acionando o sinal verde.
verificador_distancia(angulo)	Função que move o servo motor, mede a distância e controla LEDs e buzzer conforme a proximidade do objeto.	Controla a movimentação do servo motor, mede a distância e ativa o LED e buzzer de acordo com a proximidade do objeto.
loop()	Função principal executada repetidamente.	Realiza a varredura do sensor ultrassônico de 0 a 180 graus e de 180 a 0 graus, chamando verificador_distancia() em cada ângulo.

4.2.6. Código no Processing

Para ser possível visualizar as informações de ângulo e distância obtidas pelo Arduino, será necessário no Processing adicionar o seguinte código:

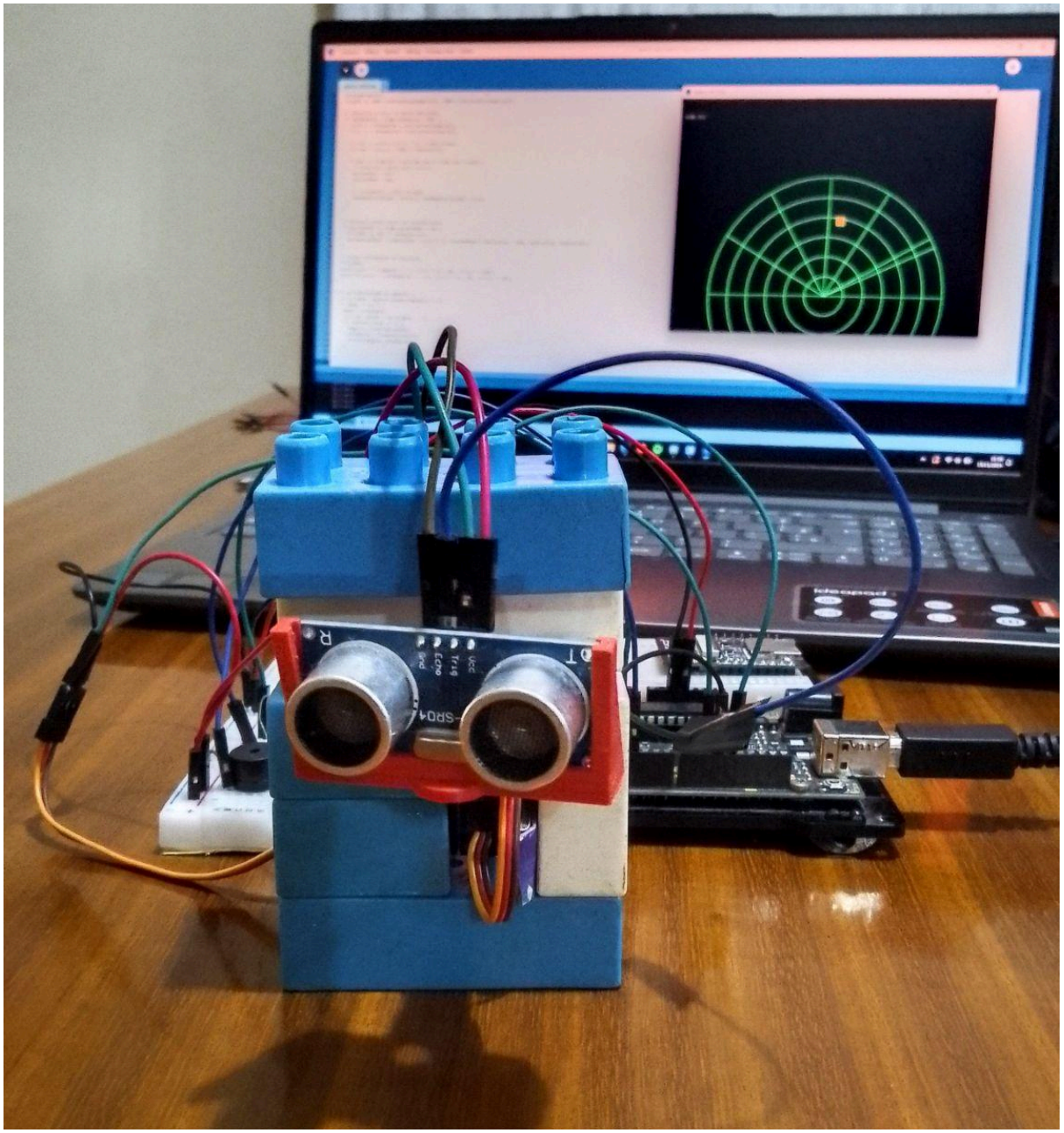
```
// Importação da biblioteca Serial para comunicação com o Arduino
```

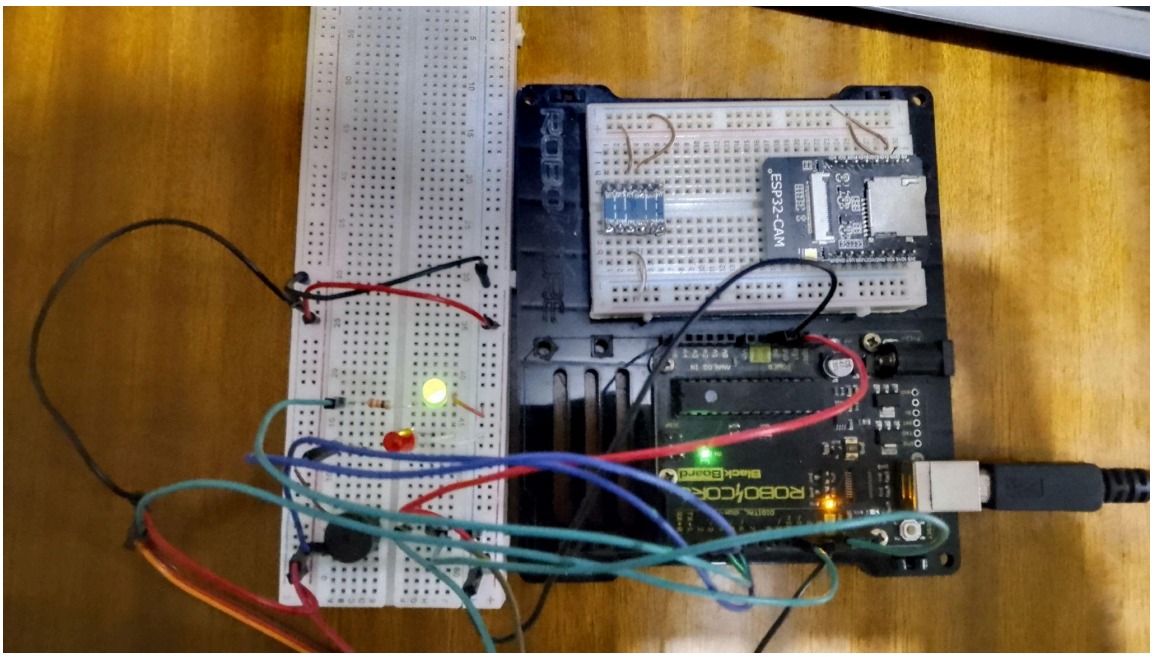
```
import processing.serial.*;

// Declaração do objeto serial para gerenciar a comunicação
Serial myPort;

// Variáveis globais para armazenar a distância e o ângulo recebidos do Arduino
float distancia = 0, angulo = 0;

void setup() {
  // Define o tamanho da janela (largura: 800px, altura: 600px)
  size(800, 600);

  // Inicializa a comunicação serial. Substitua "/dev/ttyACM0" pelo nome da porta correta no seu sistema.
  myPort = new Serial(this, "/dev/ttyACM0", 9600);

  // Configura o buffer para processar os dados apenas quando a linha completa ('\n') for recebida
  myPort.bufferUntil('\n');
}

void draw() {
  // Define o fundo preto para o radar
  background(0);

  // Move a origem (0, 0) para o centro inferior da tela
  translate(width / 2, height - 100);

  // Define a cor verde para as linhas do radar
  stroke(0, 255, 0);
  noFill();

  // Desenha as linhas do radar em ângulos de 30° (0° a 180°)
  for (int i = 0; i <= 180; i += 30) {
    // Calcula as coordenadas finais para cada linha e a desenha
    line(0, 0, 300 * cos(radians(i)), -300 * sin(radians(i)));
  }

  // Desenha os círculos concêntricos que representam as distâncias no radar
  for (int r = 50; r <= 300; r += 50) {
    ellipse(0, 0, r * 2, r * 2);
  }

  // Desenha a linha verde representando o movimento do radar (ângulo atual)
  strokeWeight(2);
  stroke(0, 255, 0);
  line(0, 0, 300 * cos(radians(angulo)), -300 * sin(radians(angulo)));

  // Verifica se há um objeto dentro do intervalo de leitura (0-300 cm)
  if (distancia > 0 && distancia < 300) {
    // Calcula a posição (x, y) do ponto baseado na distância e no ângulo
    float x = distancia * cos(radians(angulo));
    float y = distancia * sin(radians(angulo));

    // Desenha um ponto vermelho para marcar o objeto detectado
    fill(250, 0, 0);
    noStroke();
    ellipse(x, y, 8, 8); // Ponto de 8px de diâmetro
```

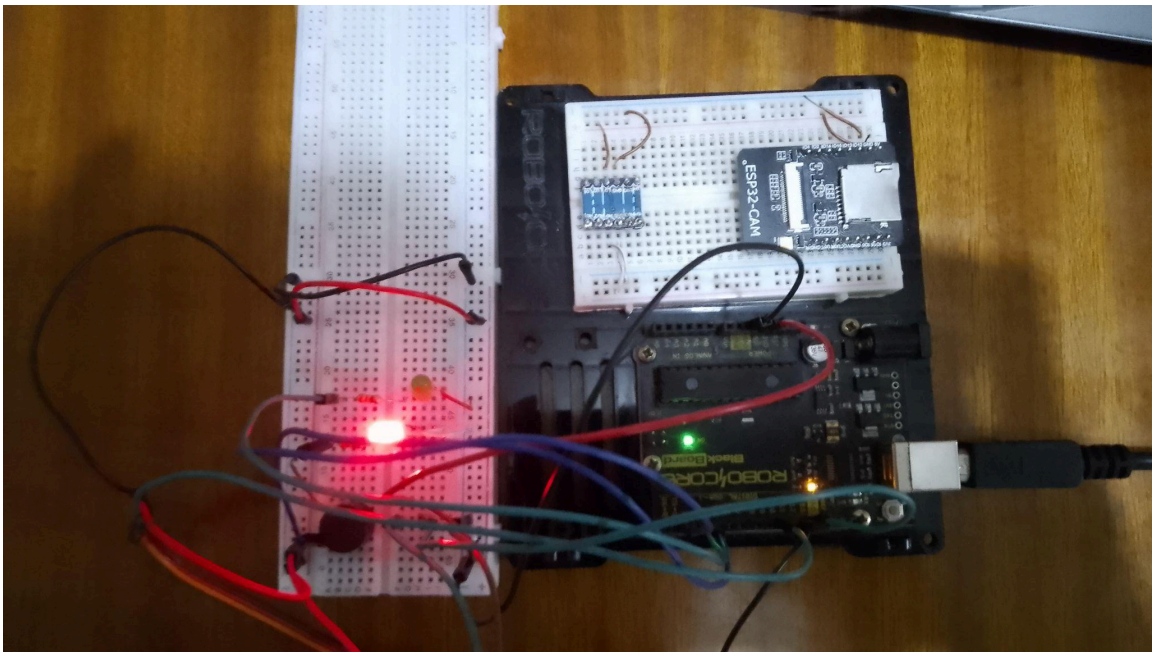
```
  }

  // Exibe na tela as informações de ângulo e distância recebidas
  fill(255); // Define a cor branca para o texto
  text("Angle: " + angulo + "°", -width / 2 + 10, -height + 150); // Mostra o ângulo atual
  text("Distance: " + distancia + " cm", -width / 2 + 10, -height + 10); // Mostra a distância atual
}

void serialEvent(Serial myPort) {
  // Lê os dados recebidos da porta serial até encontrar o caractere '\n'
  String data = myPort.readStringUntil('\n');

  // Verifica se há dados recebidos
  if (data != null) {
    data = trim(data); // Remove espaços extras no início ou no fim da string

    // Divide os dados em duas partes, separadas por uma vírgula (formato esperado: "angulo,distancia")
    String[] values = split(data, ',');

    // Se os dados contiverem exatamente dois valores (ângulo e distância)
    if (values.length == 2) {
      angulo = float(values[0]); // Converte o primeiro valor (ângulo) para float
      distancia = float(values[1]); // Converte o segundo valor (distância) para float

      // Imprime os valores no console para depuração
      println(angulo, distancia);
    }
  }
}
```

4.2.7. Visualização dos resultados

Segue abaixo algumas imagens do projeto em execução, neste caso do Radar Ultrassônico:


R
T
Gnd
Echo
Trig
Vcc





4.3. Simulador de Jukebox de 8 bits

4.3.1. Objetivos

Este código é projetado para controlar a reprodução de músicas em um dispositivo que utiliza um buzzer, LEDs e um botão para interagir com o usuário. O código tem como principais objetivos:

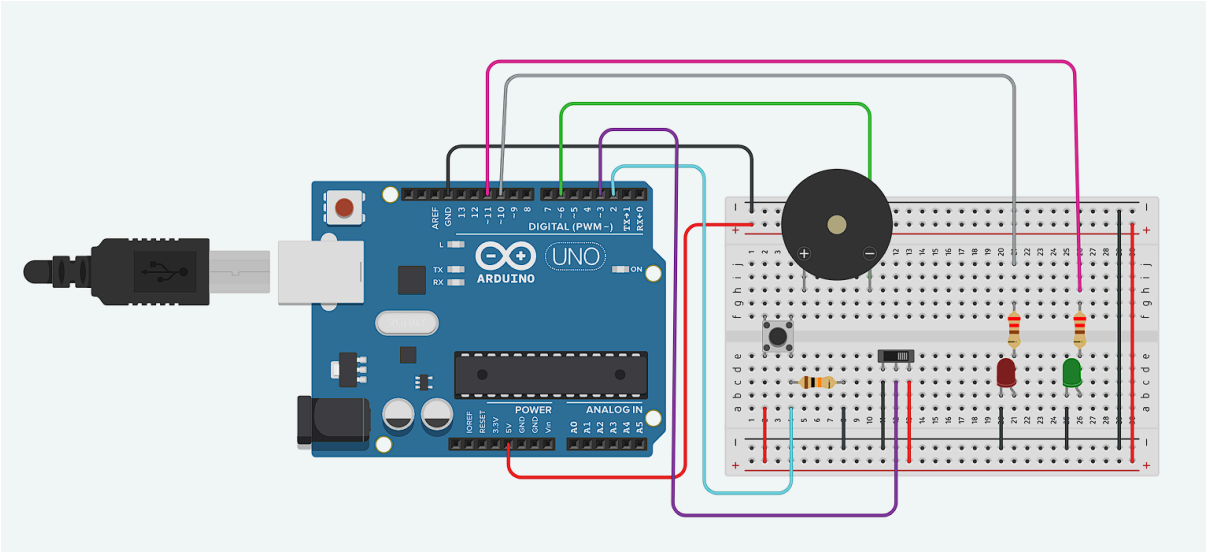
- **Reprodução de Música:** O dispositivo é capaz de tocar diferentes músicas armazenadas no código, com a capacidade de alternar entre elas.
- **Interação com o Usuário:** O usuário pode interagir com o sistema por meio de um botão, que alterna entre as músicas disponíveis.
- **Controle Visual:** LEDs são usados para indicar o estado do dispositivo (ligado/desligado) e para sinalizar a troca de músicas.
- **Modo de Desligamento:** Um interruptor é usado para ligar ou desligar o dispositivo, controlando também os LEDs e a reprodução das músicas.

4.3.2. Cenário

O cenário de aplicação deste código é em um dispositivo musical interativo que toca músicas e dá feedback visual e auditivo ao usuário. Segue o fluxo de execução do mesmo:

- **Música Tocando:**
    - Quando o dispositivo está ligado (interruptor em HIGH), a música começa a tocar no buzzer.
    - O botão permite alternar entre as músicas armazenadas.
    - A troca de música é indicada por um LED vermelho piscando, seguido do LED verde aceso quando a nova música começa a tocar.
- **Desligamento do Dispositivo:**
    - Quando o interruptor está em LOW (desliga), os LEDs são apagados e a reprodução da música é interrompida.
- **Troca de Música:**
    - Ao pressionar o botão, a música atual é interrompida e a próxima música começa a ser tocada.
    - A troca é sinalizada pelo LED vermelho piscando brevemente antes de retornar ao verde.

4.3.3. Diagrama



4.3.4. Preparativos do Ambiente

Para o desenvolvimento deste sistema, será necessário os seguintes componentes:

- 1 Arduino Uno ou Mega
- 1 Protoboard
- 1 Buzzer
- 1 Push Button
- 1 Interruptor
- 1 LED Vermelho
- 1 LED Verde
- 2 resistores de 220 ohm
- 1 resistor de 10 K ohm
- 16 jumpers

4.3.5. Código

```
#include "pitches.h"

#define BUZZER_PIN 6
#define BUTTON_PIN 2
#define INTERRUPTOR_PIN 3
#define VERMELHO_PIN 10
#define VERDE_PIN 11
```

```
int *melody[] = {

  (int[]){
    NOTE_AS4, NOTE_AS4, NOTE_AS4, NOTE_F5, NOTE_C6, NOTE_AS5, NOTE_A5, NOTE_G5, NOTE_F6,
NOTE_C6, NOTE_AS5, NOTE_A5, NOTE_G5, NOTE_F6, NOTE_C6, NOTE_AS5, NOTE_A5, NOTE_AS5,
NOTE_G5, NOTE_C5, NOTE_C5, NOTE_C5, NOTE_F5, NOTE_C6, NOTE_AS5, NOTE_A5, NOTE_G5, NOTE_F6,
NOTE_C6, NOTE_AS5, NOTE_A5, NOTE_G5, NOTE_F6, NOTE_C6, NOTE_AS5, NOTE_A5, NOTE_AS5,
NOTE_G5, NOTE_C5, NOTE_C5, NOTE_D5, NOTE_D5, NOTE_AS5, NOTE_A5, NOTE_G5, NOTE_F5, NOTE_F5,
NOTE_G5, NOTE_A5, NOTE_G5, NOTE_D5, NOTE_E5, NOTE_C5, NOTE_C5, NOTE_D5, NOTE_D5, NOTE_AS5,
NOTE_A5, NOTE_G5, NOTE_F5, NOTE_C6, NOTE_G5, NOTE_G5, REST, NOTE_C5, NOTE_D5, NOTE_D5,
NOTE_AS5, NOTE_A5, NOTE_G5, NOTE_F5, NOTE_F5, NOTE_G5, NOTE_A5, NOTE_G5, NOTE_D5, NOTE_E5,
NOTE_C6, NOTE_C6, NOTE_F6, NOTE_DS6, NOTE_CS6, NOTE_C6, NOTE_AS5, NOTE_GS5, NOTE_G5,
NOTE_F5, NOTE_C6
  },

   // Musica Harry Potter
  (int[]){
    REST, NOTE_D4, NOTE_G4, NOTE_AS4, NOTE_A4, NOTE_G4, NOTE_D5, NOTE_C5, NOTE_A4, NOTE_G4,
NOTE_AS4, NOTE_A4, NOTE_F4, NOTE_GS4, NOTE_D4, NOTE_D4, NOTE_G4, NOTE_AS4, NOTE_A4,
NOTE_G4, NOTE_D5, NOTE_F5, NOTE_E5, NOTE_DS5, NOTE_B4, NOTE_DS5, NOTE_D5, NOTE_CS5,
NOTE_CS4, NOTE_B4, NOTE_G4, NOTE_AS4, NOTE_D5, NOTE_AS4, NOTE_D5, NOTE_AS4, NOTE_DS5,
NOTE_D5, NOTE_CS5, NOTE_A4, NOTE_AS4, NOTE_D5, NOTE_CS5, NOTE_CS4, NOTE_D4, NOTE_D5, REST,
NOTE_AS4, NOTE_D5, NOTE_AS4, NOTE_D5, NOTE_AS4, NOTE_F5, NOTE_E5, NOTE_DS5, NOTE_B4,
NOTE_DS5, NOTE_D5, NOTE_CS5, NOTE_CS4, NOTE_AS4, NOTE_G4
  }

};

int *durations[] = {
// Musica do Star Wars
  (int[]){
    8, 8, 8, 2, 2, 8, 8, 8, 2, 4, 8, 8, 8, 2, 4, 8, 8, 8, 2, 8, 8, 8, 2, 2, 8, 8, 8, 2, 4, 8, 8, 8, 2, 4, 8, 8, 8, 2, 8, 16, 4, 8, 8,
8, 8, 8, 8, 8, 8, 4, 8, 4, 8, 16, 4, 8, 8, 8, 8, 8, 8, 16, 2, 8, 8, 4, 8, 8, 8, 8, 8, 8, 8, 8, 4, 8, 4, 8, 16, 4, 8, 4, 8, 4, 8, 4,
8, 1
  },

   // Musica do Harry Potter
  (int[]){
    2, 4, 4, 8, 4, 2, 4, 2, 2, 4, 8, 4, 2, 4, 1, 4, 4, 8, 4, 2, 4, 2, 4, 2, 4, 4, 8, 4, 2, 4, 1, 4, 2, 4, 2, 4, 2, 4, 2, 4, 4, 8, 4,
2, 4, 1, 4, 4, 2, 4, 2, 4, 2, 4, 2, 4, 4, 8, 4, 2, 4, 1
  }
};

// Vetor indicando a duração de cada música, visto que é utilizado para armazenamento das mesmas
vetores dinâmicos
int size_duration[] = {72, 60};

// Informar a quantidade de músicas disponíveis
int total_musics = 2;

// Estado da música e do botão

// Inicialmente, a música está parada
bool tocando = false;
```

```
// Para detectar mudanças de estado do botão
bool botao_anterior = false;

// Indice da música que esta tocando
int actual_music = 0;

// Índice da nota atual
int nota_atual = 0;

// Indicar se o interruptor esta ativado ou não
int vb = 0;


void setup() {
  Serial.begin(9600);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP); // Usar pull-up interno para o botão
  pinMode(INTERRUPTOR_PIN, INPUT);
  pinMode(VERMELHO_PIN, OUTPUT);
  pinMode(VERDE_PIN, OUTPUT);
}


// Função responsável por desligar o LED vermelho e ligar o verde
void liga_led(){
            Serial.println("Ligado");
    digitalWrite(VERDE_PIN, HIGH);
    digitalWrite(VERMELHO_PIN, LOW);
}

void desliga_led(){
  digitalWrite(VERDE_PIN, LOW);
  digitalWrite(VERMELHO_PIN, HIGH);
  Serial.println("Desligado");
}

// Função responsável por indicar a troca da música, no qual vai desligar a luz verde, ativar a vermelha por
um breve período e depois vai ativar a verde novamente
void troca_musica_led(){
  digitalWrite(VERDE_PIN, LOW);
  delay(100);
  digitalWrite(VERMELHO_PIN, HIGH);
  delay(100);
  digitalWrite(VERMELHO_PIN, LOW);
  digitalWrite(VERMELHO_PIN, HIGH);
}


/*
Função responsável por verificar se o botão foi presionado e caso afirmativo vai realizar a troca da música
a ser tocada
*/
void verifica_botao_pressionado(){

  // Verifica o estado do botão (pressionado = LOW)
            bool botao_atual = digitalRead(BUTTON_PIN) == LOW;
```

```
        if (botao_atual && !botao_anterior) {

    // Alternar a musica a ser tocada
    tocando = !tocando;

    // Casp a musica já estiver tocando, somente apresenta no log
    if (tocando) {

      // Log: início da reprodução
      Serial.println("Tocando música...");
    }
    else {

      // Acender os leds que indica troca da música
      troca_musica_led();

      // Altera a música atual para a próxima música disponível
      actual_music = (actual_music + 1) % total_musics;

      // Reinicia a contagem da nota musical, ou seja, define para que a mesma inicie na posição 0
      nota_atual = 0;

      // Informa que o dispositivo está pronto para tocar a música e da um delay de 1 segundo antes de
começar a nova música
      tocando = true;
      delay(1000);

      // Log de Troca de Música
      Serial.println("Houve a troca de música");
    }
   }

        // Atualiza o estado anterior do botão para detectar mudanças futuras
        botao_anterior = botao_atual;

}

void tocar_nota_musical(){

        // Calcula a duração da nota atual
        int duration = 1000 / durations[actual_music][nota_atual];

        // Emite o som correspondente à nota, caso não seja um descanso
        if (melody[actual_music][nota_atual] != REST) {
    tone(BUZZER_PIN, melody[actual_music][nota_atual], duration);
   }

        // Aguarda a duração da nota antes de avançar
        delay(duration * 1.30);

        // Para o som da nota atual
   noTone(BUZZER_PIN);

        // Avança para a próxima nota
        nota_atual++;
```

```
}

/*
* Função responsável por trocar a música em execução ao finalizar a anterior
*/
void trocar_musica_ao_finalizar_anterior(){

  // Log: fim da música
  Serial.println("Música terminou.");

  // Parar a reprodução
    tocando = false;

  // Alterar para a próxima música disponível
  actual_music = (actual_music + 1) % total_musics; // Incrementa e reinicia no total

  // Reinicia o índice para o início da música
    nota_atual = 0;

  // Iniciar a reprodução da nova música
    tocando = true;

}


void loop() {


  // Ler o Interruptor, que indica se o dispositivo está no modo ligado ou desligado
  vb = digitalRead(INTERRUPTOR_PIN);

  // Caso o dispositivo estiver ligado -> Interruptor como HIGH
  if (vb==HIGH) {

    // Acender os leds indicando que o dispositivo esta ligado
    liga_led();

    // Verificar se algum botão foi pressionado ou se necessita trocar de música
   verifica_botao_pressionado();

/**
   * Reproduz as notas da música se `tocando` for verdadeiro.
   * - Toca cada nota em sequência, respeitando sua duração.
   * - Interrompe automaticamente quando a música termina, reiniciando os índices.
 */

    if (tocando) {
      if (nota_atual < size_duration[actual_music])
         tocar_nota_musical();
      else
        trocar_musica_ao_finalizar_anterior();
    }
  }

  // Caso o dispositivo estiver desligado
  else {
```

```
    desliga_led();

  }
}
```

Para uma melhor compreensão das informações do código, na tabela abaixo será exemplificado quais as variáveis, bibliotecas e a função de cada uma para o correto funcionamento do código:

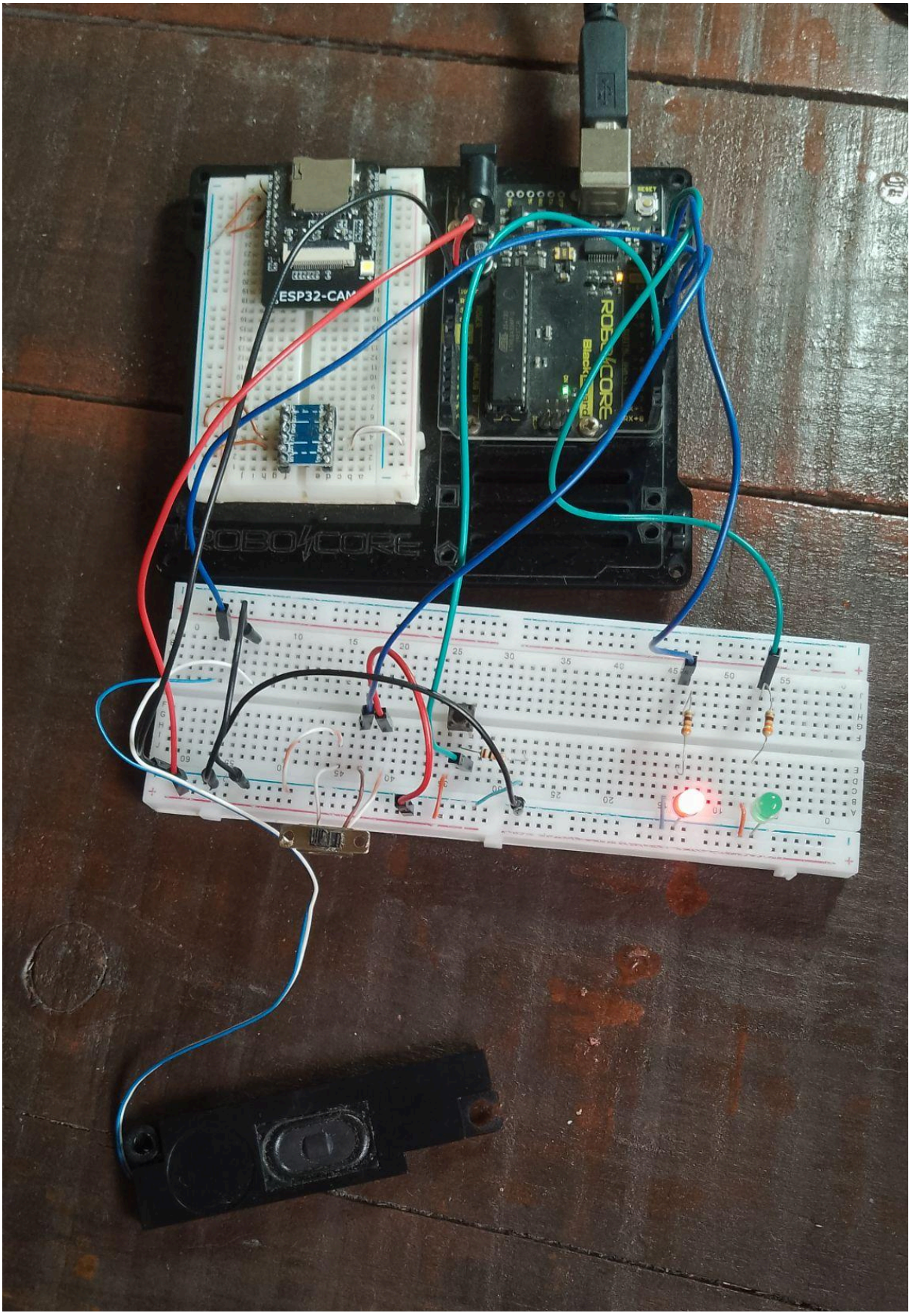
Elemento	Descrição	Função no Código
Constantes e Variáveis		
BUZZER_PIN	Pino digital 6 conectado ao buzzer.	Controla a emissão de sons pelas notas musicais quando ativado.
BUTTON_PIN	Pino digital 2, configurado como entrada com pull-up interno.	Detecta a pressão do botão para alternar entre as músicas.
INTERRUPTOR_PIN	Pino digital 3, configurado como entrada.	Detecta o estado do interruptor (ligado/desligado) para controlar a ativação do dispositivo.
VERMELHO_PIN	Pino digital 10 conectado ao LED vermelho.	Acende o LED vermelho para indicar que o dispositivo está trocando de música ou desligado.
VERDE_PIN	Pino digital 11 conectado ao LED verde.	Acende o LED verde para indicar que o dispositivo está ligado e em funcionamento.
melody[]	Vetor de ponteiros de arrays de notas musicais (valores definidos em pitches.h)	Armazena as notas musicais para as músicas a serem tocadas. Cada música tem seu próprio array de notas.
durations[]	Vetor de ponteiros de arrays de durações das notas musicais.	Armazena a duração das notas para cada música, que determina quanto tempo cada nota será tocada.

size_duration[]	Vetor de inteiros que indica a quantidade de notas em cada música.	Determina o número de notas de cada música para que o código saiba quantas notas deve processar antes de finalizar a execução da música.
total_musics	Variável inteira que armazena o número total de músicas.	Controla o número total de músicas disponíveis para alternância.
tocando	Variável booleana que indica se a música está tocando ou não.	Controla o estado da música (se está tocando ou não). Quando a música termina, ela é reiniciada ou trocada.
botao_anterior	Variável booleana que armazena o estado anterior do botão (pressionado ou não).	Usada para detectar mudanças de estado no botão e evitar múltiplas leituras do mesmo pressionamento.
actual_music	Variável inteira que armazena o índice da música atual em execução.	Controla qual música está sendo tocada no momento. Altera-se cada vez que o botão é pressionado para alternar a música.
nota_atual	Variável inteira que armazena o índice da nota atual sendo tocada.	Controla a posição da música, determinando qual nota deve ser tocada em cada momento.
vb	Variável inteira que armazena o estado do interruptor (ligado ou desligado).	Lê o estado do interruptor para determinar se o dispositivo está ligado ou desligado.
Funções		
setup()	Função de inicialização.	Configura os pinos e inicializa a comunicação serial para monitoramento do estado do dispositivo.
liga_led()	Função que acende o LED verde e apaga o LED vermelho.	Indica que o dispositivo foi ligado e está em funcionamento.

desliga_led()	Função que acende o LED vermelho e apaga o LED verde.	Indica que o dispositivo foi desligado.
troca_musica_led()	Função que faz a troca dos LEDs (vermelho e verde) ao alternar de música.	Inicia a troca de música, acendendo o LED vermelho brevemente antes de voltar ao verde.
verifica_botao_pressionado()	Função que verifica o estado do botão e alterna a música se pressionado.	Detecta a pressão do botão e alterna a música. Também reinicia a música e atualiza os LEDs ao trocar a música.
tocar_nota_musical()	Função que toca a nota musical com base no índice atual de nota e duração.	Emite a nota musical no buzzer e aguarda pela duração antes de parar o som e avançar para a próxima nota.
trocar_musica_ao_finalizar_anterior()	Função que troca a música automaticamente ao finalizar a atual.	Quando a música atual termina, altera para a próxima música na lista, reinicia o índice de notas e começa a tocar a nova música.
loop()	Função principal que executa repetidamente.	Contém o ciclo de leitura do estado do interruptor e do botão, e controla a execução da música e a troca de LEDs conforme o estado do dispositivo.

4.3.6. Visualização dos resultado

Segue abaixo uma imagem do circuito configurado:


ESP32-CAM
ROBOCORE

5. DICAS E ERROS COMUNS

Abaixo apresentaremos os principais erros obtidos durante o desenvolvimento do projeto e possíveis dicas para novos usuários

5.1. Erros

Segue a lista dos principais erros encontrados:

- Conectar os LED na polaridade incorreta
- Deixar o monitor Serial quando for realizar o upload do código no Arduino
- Conectar os jumpers nas portas incorretas do componente
- Realizar a criação do circuito direto na placa sem realizar um planejamento com um diagrama
- Efetuar o download da biblioteca incorretamente
- Não cuidar com o tamanho do código e do uso de variáveis desnecessários, causando erro de falta de memória

5.2. Dicas

A fim de auxiliar os novos usuários, seguem algumas dicas para o melhor aproveitamento dos projetos

- Antes de começar um circuito, desenvolver um diagrama e simulação no Tinkercad
- Realizar comentários no código explicando o funcionamento de cada trecho, facilitando na manutenção e documentação do projeto
- Buscar em fóruns online, AI ou blogs o funcionamento do componente, para compreender como aplicar na prática
- Realizar estudos em estrutura de dados
- Realizar a nomeação das variáveis claramente e com base em sua funcionalidade.

6. CONSIDERAÇÕES FINAIS

Conclui-se que o desenvolvimento de sistemas embarcados utilizando a plataforma Arduino oferece um vasto potencial criativo, permitindo a construção de soluções práticas e interativas. Por meio da compreensão dos componentes e de

seu funcionamento, foi possível implementar projetos como o sensor de temperatura com display LCD, o radar ultrassônico com sinalização de obstáculos e o simulador de jukebox em 8 bits. Cada etapa reforçou a importância da integração entre hardware e software, além de consolidar conhecimentos fundamentais em eletrônica e programação embarcada, abrindo caminho para o desenvolvimento de aplicações mais avançadas e inovadoras.

7. REFERÊNCIAS

ARDUINOPORTUGAL.PT. O que é o Arduino UNO? 2017. Disponível em: https://www.arduinoportugal.pt/o-que-e-o-arduino/. Acesso em: 23 nov. 2024.

CHEETAH E-RACING. Como utilizar um sensor de temperatura com Arduino? 2022. Disponível em: https://www.makerhero.com/blog/como-utilizar-um-sensor-de-temperatura/. Acesso em: 24 nov. 2024.

CRAVO, Edilson. Servo Motor: o que é um, como funciona e quais as vantagens. O que é um, como funciona e quais as vantagens. 2024. Disponível em: https://blog.kalatec.com.br/o-que-e-servo-motor/. Acesso em: 24 nov. 2024.

FARACO, João. Introdução ao Processing. 2013. Disponível em: https://joaofaraco.com.br/blog/2013/03/25/introducao-ao-processing/. Acesso em: 24 nov. 2024.

GELAL, Oguz. Ultimate Guide to the Processing Language Part I: the fundamentals. The Fundamentals. Disponível em: https://www.toptal.com/game/ultimate-guide-to-processing-the-fundamentals. Acesso em: 24 nov. 2024.

GUSE, Rosana. Buzzer: o que é, tipos, função e aplicações. O que é, tipos, função e aplicações. 2024. Disponível em: https://www.makerhero.com/guia/componentes-eletronicos/buzzer/. Acesso em: 24 nov. 2024.

GUSE, Rosana. O que é LED: funcionamento, tipos e exemplos. Funcionamento, tipos e exemplos. 2024. Disponível em: https://www.makerhero.com/guia/componentes-eletronicos/led/. Acesso em: 24 nov. 2024.

INSTRUMENTS, Sense Sensors e. Sensores Ultrassônicos. Disponível em: https://www.sense.com.br/arquivos/produtos/arq1/Sensores_Ultrass%c3%b4nicos_Sense_Folheto_Rev_%20J.pdf. Acesso em: 24 nov. 2024.

KOLLMORGEN. O que é um servomotor? 2024. Disponível em: https://www.kollmorgen.com/pt-br/resources/technologies-explained/motors/what-is-a-servo-motor. Acesso em: 24 nov. 2024.

LOUSADA, Ricardo. Componentes Básicos do Arduino: o que é resistor, led, potenciômetro, push button. O que é Resistor, LED, Potenciômetro, Push Button. 2024. Disponível em: https://blog.eletrogate.com/componentes-basicos-do-arduino-o-que-e-resistor-led-potenciometro-push-button/. Acesso em: 24 nov. 2024.

LOUSADA, Ricardo. O que é Arduino: para que serve, vantagens e como utilizar. Para que Serve, Vantagens e como Utilizar. 2020. Disponível em: https://blog.eletrogate.com/o-que-e-arduino-para-que-serve-vantagens-e-como-utilizar/. Acesso em: 24 nov. 2024.

MAKIYAMA, Marcio. O que é protoboard: o que é, como funciona e para que serve. O que é, como funciona e para que serve. 2024. Disponível em: https://victorvision.com.br/blog/o-que-e-protoboard/. Acesso em: 24 nov. 2024.

MATTEDE, Henrique. O que é Servo motor e como funciona? Disponível em: https://www.mundodaeletrica.com.br/o-que-e-servo-motor-e-como-funciona/. Acesso em: 24 nov. 2024.

MATTEDE, Henrique. O que é um LED? 2024. Disponível em: https://www.mundodaeletrica.com.br/o-que-e-um-led/. Acesso em: 24 nov. 2024.

MATTEDE, Henrique. Protoboard: o que é? Tipos e como usar!. O que é? Tipos e como usar!. 2024. Disponível em: https://www.blogdaeletronica.com.br/protoboard-que-e-tipos-como-usar/. Acesso em: 24 nov. 2024.

MURTA, José Gustavo Abreu. Guia Completo do Display LCD: Arduino. Arduino. 2023. Disponível em: https://blog.eletrogate.com/guia-completo-do-display-lcd-arduino/. Acesso em: 24 nov. 2024.

VIANA, Carol Correia. Ligar e Desligar LED com Botão (Push Button – Chave Táctil) com Arduino. 2020. Disponível em: https://www.blogdarobotica.com/2020/09/28/ligar-e-desligar-led-com-botao-push-button-chave-tactil-com-arduino/. Acesso em: 24 nov. 2024.

VIDAL, Vitor. Sensor Ultrassônico HC-SR04 com Arduino. 2022. Disponível em: https://blog.eletrogate.com/sensor-ultrassonico-hc-sr04-com-arduino/. Acesso em: 24 nov. 2024.