UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELÉTRICA E ELETRÔNICA
BACHARELADO EM ENGENHARIA ELÉTRICA

LORENA SMIGUEL TEIXEIRA

ESTUDO SOBRE SENSORES NA PLATAFORMA ARDUÍNO PARA APLICAÇÕES EM MÚSICA ELETRÔNICA

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2019

LORENA SMIGUEL TEIXEIRA

UM ESTUDO SOBRE SENSORES NA PLATAFORMA ARDUÍNO PARA APLICAÇÕES EM MÚSICA ELETRÔNICA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica do Departamento Acadêmico de Elétrica da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. André Koscianski

PONTA GROSSA

2019

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEP. ACADEMICO DE ELETRONICA-PG

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO - TCC

ESTUDO SOBRE SENSORES NA PLATAFORMA ARDUÍNO PARA APLICAÇÕES EM MÚSICA ELETRÔNICA

Por

Lorena Smiguel Teixeira

Monografia apresentada às 10 horas 00 min. do dia 21 de agosto de 2020 como requisito parcial, para conclusão do Curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná, Câmpus Ponta Grossa. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação e conferidas, bem como achadas conforme, as alterações indicadas pela Banca Examinadora, o trabalho de conclusão de curso foi considerado APROVADO.

Banca examinadora:

Prof. Dr. Hugo Valadares Siqueira	Membro
Prof. Dr. Max Mauro Dias Santos	Membro
Prof. Dr. André Koscianski	Orientador
Prof. Dr. Josmar Ivanqui	Professor(a) responsável TCCII

Documento assinado eletronicamente por **MAX MAURO DIAS SANTOS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 24/08/2020, às 16:05, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **HUGO VALADARES SIQUEIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 24/08/2020, às 16:42, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **ANDRE KOSCIANSKI**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 26/08/2020, às 16:10, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **JOSMAR IVANQUI**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 27/08/2020, às 18:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.utfpr.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1564991** e o código CRC **5136077F**.

Referência: Processo nº 23064.021957/2020-01 SEI nº 1564991

Dedico este trabalho à minha avó, que faleceu esse ano e levou uma parte de mim...

“A música exprime a mais alta filosofia numa linguagem que a razão não compreende.” (SCHOPENHAUER, Arthur, 1819)

RESUMO

TEIXEIRA, Lorena Smiguel. **Estudo sobre sensores na plataforma Arduino para aplicações em música eletrônica.** 2019. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

Os instrumentos musicais estão em desenvolvimento constante assim como a maneira de utilizá-los. Este trabalho apresenta um estudo da utilização de sensores para o controle dos sons, verificando também sua confiabilidade e acessibilidade para tal aplicação. O microcontrolador utilizado neste trabalho foi o Arduino, testado com os sensores HC-SR04 e MPU6050. Testes de confiabilidade mostraram que ambos foram suficientemente sensíveis e precisos para gestos como o afastar e aproximar das mãos, assim como a rotação do pulso. Mais estudo seria requerido para implementar mais sensores, assim como usar uma comunicação mais avançada entre computador e sensor, a fim de possibilitar uma gama maior de criação sonora. Por fim, o trabalho propõe algumas modificações e futuras melhorias para tornar a criação de músicas mais intuitiva.

Palavras-chave: Música. Sensor. Microcontrolador. Arduino.

ABSTRACT

TEIXEIRA, Lorena Smiguel. **Study of Sensors in the Arduino Platform for Electronic Music Applications.** 2019. 56 p. Work of Conclusion Course (Graduation in Electronic Engineering) - Federal Technology University - Paraná. Ponta Grossa, 2019.

Musical instruments are in constant development and so is the way to use them. This paper presents a study of the use of sensors to control sounds through movements, analyzing also its reliability and accessibility for such application. The microcontroller used in this work was Arduino, tested with the HC-SR04 and MPU6050 sensors. Reliability tests showed that both were sufficiently sensitive and accurate for movements such as drawing the hands closer and farther, so as rotating the wrist. More study would be required to implement more sensors as well as the use of more advanced communication between computer and sensor to enable a wider range of sound creation. Lastly, the work proposes some modifications and future improvements to make music creation more intuitive.

Keywords: Music. Sensor. Microcontroller. Arduino.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ADSR	*Attack, Decay, Sustain, Release*
CPU	*Central Processing Unit*
FM	*Frequency Modulator*
GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
PWM	*Pulse Width Modulation*
RFID	*Radio Frequency Identification*
USB	*Universal Serial Bus*

LISTA DE ACRÔNIMOS

MIDI	*Musical Instrument Digital Interface*

LISTA DE UNIDADES

Hz	Hertz

SUMÁRIO

1 INTRODUÇÃO

A música faz parte da história da humanidade e se mostra grande influenciadora principalmente do público jovem. Além de servir como entretenimento, também é importante contribuinte para movimentos culturais, estudos matemáticos e até mesmo terapias.

À primeira vista, música e engenharia não possuem relação alguma, mas na realidade temos “uma grande quantidade de evidências de conexões entre matemática e música: vibrações, frequências, técnicas composicionais e assim vai” (KUNG, 2013, p.99). Além disso, Kung (2013) também apresenta outras similaridades entre esses campos como o desenvolvimento benéfico para crianças, o uso da criatividade e o emprego da abstratividade.

Os instrumentos musicais são em sua maioria compostos por um objeto o qual, em contato com certos gestos, produz vibrações gerando ondas sonoras. Esses gestos possuem inúmeras variações, por exemplo, no piano temos o uso dos dedos para tocar cada tecla, mas não só isso, a força usada modifica a intensidade do som reproduzido. Já em instrumentos como o violoncelo temos ainda mais gestos aplicáveis como o vibrato, que é uma técnica que consiste em pressionar uma corda e vibrar a mão levemente, tornando o som mais rico, dando um toque mais sentimental.

Curiosamente, não só na música, mas em nosso cotidiano, gestos se tornaram essenciais. Telas de celulares *touchscreen* dependem de inúmeros gestos para seu uso. Alguns até possuem sensores para tirar fotos com o gesto da mão, sem ao menos tocar no aparelho.

Já em um ambiente mais específico como fábricas e indústrias, sensores de proximidade também são parte crucial da segurança no ambiente de trabalho.

A interface digital também contribui para a facilidade do uso de certos equipamentos, tornando a ação mais visual e intuitiva. Interfaces de toque usadas em hospitais para controlar equipamentos e interfaces de *softwares* para criação de música são exemplos clássicos que podemos mencionar aqui.

É interessante ressaltar que “o uso da tecnologia de sensores é parte fundamental na criação de interfaces para música feitas em computador. No entanto, pouca investigação ocorreu sobre a adequação de sensores para tarefas específicas nessas interfaces.” (MARSHALL; WANDERLEY, 2013, p.99)

Há ainda muitos campos de investigação em relação a telas *touchscreen* e qualquer outro aparelho que use gestos para sua manipulação básica. Por exemplo, muitas pessoas ainda apresentam certa dificuldade para com seu uso, como deficientes físicos.

Com a criação de um instrumento onde não há supostamente a necessidade de teclas, botões ou cordas, estendemos a possibilidade da interação e acessibilidade para uma fatia muito mais ampla de pessoas.

Este trabalho procura estudar esse assunto, isto é, a possibilidade de utilizar recursos eletrônicos para melhorar a experiência e acessibilidade na criação de música utilizando sensores, *softwares* e posteriormente sintetizadores eletrônicos.

2 REVISÃO DE LITERATURA

2.1 SONS E SINTETIZADORES

Levando tudo citado na introdução deste trabalho em consideração, é conveniente dar uma olhada mais profunda em um gênero específico da música: a eletrônica.

Com o descobrimento do elétron por J. J. Thomson, e as equações de eletromagnetismo de J. C. Maxwell, certamente houve uma revolução na área da eletrônica no final do século XIX que continuou fortemente durante as décadas seguintes. E não foi diferente em relação à música.

Como já apontou Couturier em sua tese,

> Os avanços em eletrônica e informática tornaram possível desenvolver novas formas de gerar som livres de tensões mecânicas em instrumentos musicais acústicos, bem como novas formas de controle do som através do desenvolvimento de sistemas de captura de gestos.” (COUTURIER, 2004, p. 9)

Um dos primeiros instrumentos musicais eletrônicos criados foi o theremin (Figura 1), inventado por Léon Theremin no começo do século XX. Esse instrumento consistia em duas hastes as quais controlam a amplitude e frequência do som sem ao menos tocar no instrumento. Ali, a mão do músico funciona como uma espécie de capacitor variável e seu movimento permite a criação de sons na faixa de frequências audíveis, os mesmos resultam em um sinal enviado para um alto-falante. (COUTURIER, 2004, p. 15)

Figura 1 - Clara Rockmore tocando o theremin

Fonte: Couturier (2004)

Algumas décadas depois, mais precisamente em 1948, Hugh Le Craine cria um protótipo, por muitos considerado o primeiro sintetizador, o qual finalmente “dava ao músico um controle sensível por toque de elementos como frequência, timbre e volume” (HOLMES, 2002, p.147). Esse instrumento foi chamado pelo criador de *Electronic Sackbut* (Figura 2) e possuía um espectro muito maior de possibilidades de tons do que as teclas de um teclado, pois se conseguia ajustar o timbre pelo controle da tensão e sua forma de onda. Um bom exemplo seria a possível transição de uma nota do teclado para outra de um jeito suave, pressionando a tecla inclinadamente para o lado da próxima tecla desejada.

No decorrer dos anos teremos mais sintetizadores, ainda mais precisos e com mais funções. Dentre os que se destacaram tem-se o sintetizador Moog, criado em 1964 por Robert Moog, que contou com a utilização de transistores para se ter um controle modular analógico das ondas produzidas. Posteriormente houve o surgimento do GROOVE, criado por Max Mathews e Richard Moore em 1970, sendo o primeiro sintetizador que fazia conexões com um computador, aumentando as possibilidades de controle e até mesmo de armazenamento do som.

Figura 2 - Le Craine tocando o *Electronic Sackbut*.

Fonte: Couturier (2004)

Daí em diante, com o avanço das pesquisas em informática, “viu-se uma necessidade em poder armazenar todo conteúdo musical no meio computacional por meio de alguma linguagem ou protocolo. Isso permitiria que o som pudesse ser

manipulado de outras formas, aumentando ainda mais as possibilidades de criação" (MENEZES, 2007, p. 2). E assim surge o protocolo MIDI (*Musical Instrument Digital Interface*).

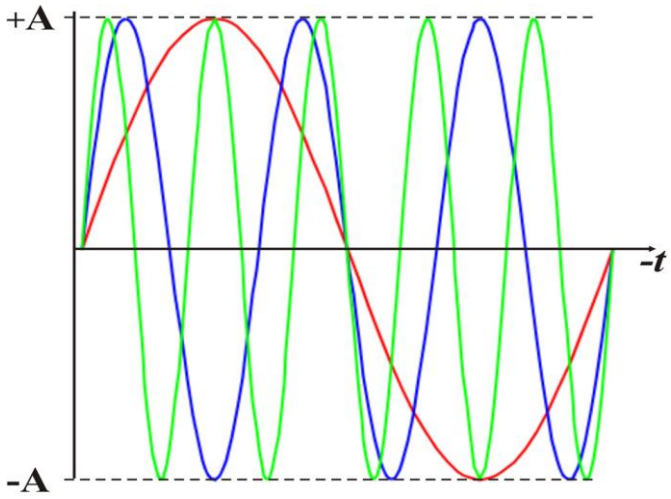
2.1.1 Frequências e Notas Musicais

O som possui inúmeras características como intensidade, timbre, e uma das mais notáveis é a frequência.

A frequência é a propriedade responsável pelo agudo ou grave, sendo que quanto mais alta a frequência, mais agudo é o som, e quanto mais baixa a frequência, mais grave. Na Figura 3 podemos notar três ondas, todas com frequências distintas.

Levando em consideração o que foi dito acima, temos conhecimento que a onda na cor vermelha seria o tom mais grave, a verde seria a mais aguda, e a azul seria um tom intermediário.

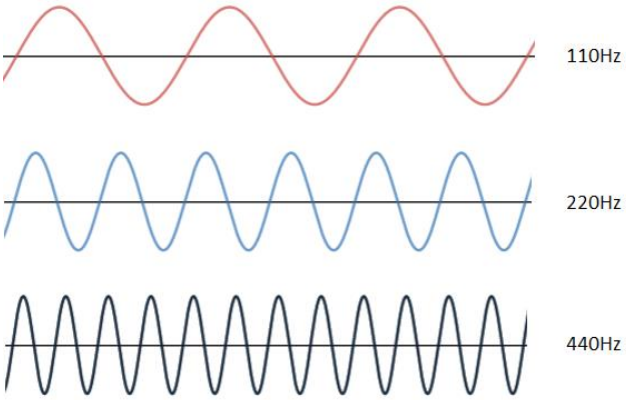
Figura 3 - Três ondas senoidais com frequências diferentes



Fonte: Dantas; Cruz (2018)

Para se reproduzir notas musicais precisamos de frequências específicas e essa relação já foi notada há muito tempo pelo famoso matemático Pitágoras, por volta de 540 A.C., que notou certas características entre os sons harmônicos. Ele percebeu que ao tocar uma corda esticada pelas extremidades, essa corda produzia um som específico. Ao dividir a corda pela metade, a mesma produzia o mesmo som, porém mais agudo (PEREIRA, 2013, p. 19). Na Figura 4 podemos visualizar isso com a nota lá, representada em diferentes frequências.

Figura 4- Nota lá representada em frequências diferentes



Fonte: Autoria própria

Com o passar do tempo viu-se a necessidade de separar e organizar as notas de maneira equivalente. Para isso, foi usado de referência as notas Dó (261,6 Hz) e Si (246,9 Hz), que possuíam um intervalo mais próximo (semitom) e foi obtida uma relação entre suas frequências. (DANTAS; CRUZ, 2018, p. 3)

$$\frac{261{,}6}{246{,}9} = 1{,}0595$$

Com essa relação foi possível achar todas as outras notas e chegar ao que chamamos de “escala cromática”, a qual possui doze notas no total (7 tons e 5 semitons).

Curiosamente, esse mesmo número é igual à raiz décima segunda de dois, de modo que se o número for multiplicado por ele mesmo doze vezes, voltará ao dois inicial, como mostrado abaixo:

$$\sqrt[12]{2} = 1{,}0595$$
$$(\sqrt[12]{2})^2 = 2$$

Isso apenas confirma o que Pitágoras notou ao dividir o comprimento da corda ao meio, onde a nota voltará a ser a mesma, porém com a frequência duplicada.

O código usado neste trabalho utilizou valores tabelados da escala cromática para gerar tons usando os sensores citados anteriormente.

2.1.2 Geração eletrônica de sons

2.1.2.1 Utilização de PWM

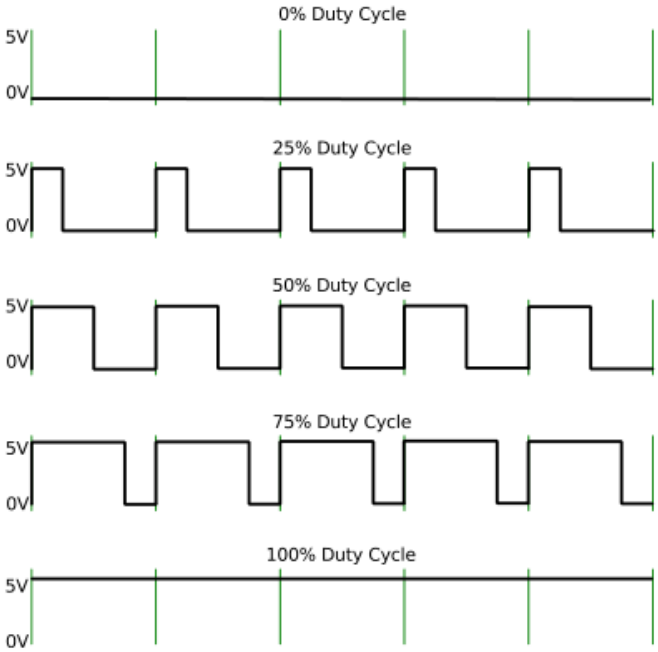
O *Pulse Width Modulation*, ou Modulação de Largura de Pulso, se trata do conceito de controlar um sinal digital, gerindo a energia que alimenta um equipamento elétrico. Essa técnica tem grande aplicação em motores elétricos, LEDs e qualquer outro aparelho que precise de um controle em sua função principal.

No caso dos microcontroladores, muitos vem com entradas e saídas digitais com apenas dois estados, 1 ou 0. Nesse caso quando se quer usá-lo para acender e apagar um LED, por exemplo, o aparelho funcionará perfeitamente. Porém, se o microcontrolador possuir uma função PWM, ter-se-á a possibilidade de controlar a luminosidade do mesmo. Em motores, pode-se facilmente controlar a sua velocidade de rotação.

Esse controle só é possível com a modulação dos níveis de tensão e corrente fornecidas a um equipamento, a qual se resume em fornecer e retirar a energia da carga em uma taxa muito rápida. Logo, quanto maior o tempo que a carga fica energizada, maior será a potência fornecida a ela. Deste modo, se o objetivo é que um motor gire com apenas 50% de sua velocidade máxima, basta programar o PWM para manter a mesma proporção na duração de liga e desliga.

No PWM temos o que chamamos de ciclo ativo (*duty cycle*), que nada mais é a duração do nível alto em uma onda quadrada. Na Figura 5, podemos ver cinco diferentes ciclos ativos. O primeiro está em 0%, o que significa que não há nenhuma energia sendo fornecida ao equipamento. Logo abaixo temos um ciclo ativo de 25%, ou seja, o aparelho fica em nível lógico alto em apenas um quarto do tempo de um ciclo. O mesmo é mostrado nos gráficos seguintes, apenas com ciclos ativos maiores até atingir a carga total. (BIANCHI; QUEIROZ, 2012, p. 540)

Figura 5 – Exemplos de diferentes *Duty Cycles*



Fonte: Bianchi; Queiroz. (2012)

Sintetizadores utilizam do PWM para controlar inúmeras características do som, por exemplo, controlar o volume produzido pelo autofalante de forma amostrada e até mesmo mudar sua frequência de forma precisa.

2.1.2.2 Síntese FM

Antes de entrar na metodologia do trabalho em si, é interessante citar o conceito de síntese FM (*Frequency Modulator*).

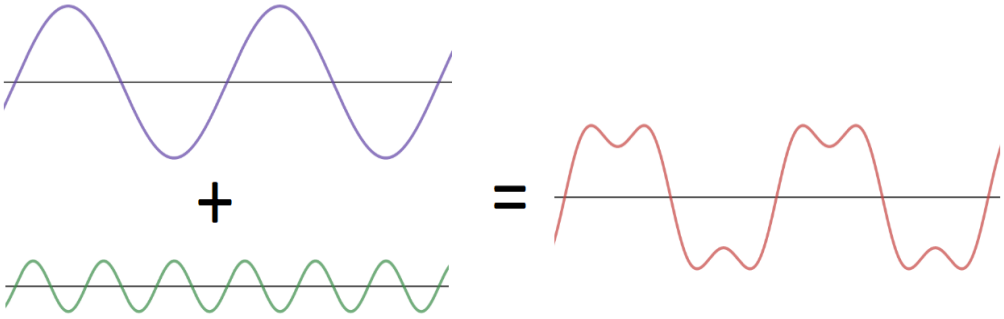
A síntese FM se resume no uso de dois ou mais osciladores em série para modificar uma onda básica, geralmente um seno, mas também podendo ser uma onda quadrada ou triangular.

Todos os osciladores, também conhecidos como operadores, possuem uma entrada e uma saída onde podem ser conectados a outros operadores a fim de modular a onda consecutivamente. O operador final reproduz o som desejado para o usuário contendo todas as modificações anteriores. Essas modificações são responsáveis pela criação de inúmeros timbres, ou seja, pela criação de sons com diferentes aspectos sonoros. (FOLLE, 2015, p.10)

> In FM, the instantaneous frequency of a carrier wave is varied according to a modulating wave, such that the rate at which the carrier varies is the frequency of the modulating wave, or modulating frequency. The amount the carrier varies around its average, or peak frequency deviation, is proportional to the amplitude of the modulating wave. (CHOWNING, 1973, p.1)

Um exemplo simples de como os operadores “carregam” a onda se encontra na Figura 6, abaixo: a soma de duas ondas senoidais distintas formando uma terceira e nova onda. A imagem ilustra uma maneira que sintetizadores, tanto em hardware quanto em software, podem usar para combinar duas ondas para obter uma terceira.

Figura 6 - Soma de duas ondas senoidais formando uma nova onda



Fonte: Autoria própria

Essa técnica de modulação do som foi citada pela primeira vez em 1973 por John Chowning, e rapidamente patenteada e licenciada pela corporação japonesa Yamaha (CHOWNING, 1973, p.1). Aproveitando seus direitos sobre a nova técnica, a empresa lançou o primeiro instrumento comercial capaz de realizar essa síntese: o teclado Yamaha DX7 (Figura 7), lançado em 1983. Esse sim possuía um leque imenso para criação de sons e pela primeira vez havia no mercado um sintetizador de preço acessível, tornando-o um dos instrumentos mais adquiridos da época.

Figura 7 - Teclado Yamaha DX7

Fonte: Yamaha Corporation. Acesso em: 11/10/2019.

Uma alternativa interessante para a síntese FM seria o uso da envoltória de amplitude chamada ADSR. Essa envoltória, ou “envelope”, contém quatro valores de tempo para uma certa amplitude sonora: ataque, decaimento, sustentação e repouso. Cada instrumento possui um valor característico para esses parâmetros e isso faz com que seja uma técnica simples muito usada em sintetizadores. (CANN, 2007, p.15)

Basicamente, com essas técnicas e o protocolo MIDI, a criação de sons se tornou muito mais fácil e intuitiva, fazendo com que os músicos tivessem mais possibilidades de produzir uma melodia complexa, mesmo usando um princípio de funcionamento relativamente simples.

2.1.3 Computadores e Música

2.1.3.1 O protocolo MIDI

Inúmeras empresas notaram a necessidade de criar uma interface padrão para que músicos pudessem ter mais recursos musicais de uma maneira mais ampla e prática. E assim, graças à colaboração entre as corporações Sequencial Circuits, Yamaha, Roland, Korg e Kawai, essa interface foi consolidada em 1983 e foi nomeada de conexão MIDI, a qual finalmente permitia uma comunicação padrão entre o instrumento e o computador.

> MIDI (Musical Instrument Digital Interface) é um protocolo padrão que apresenta um conjunto de mensagens capaz de levar toda a informação necessária a um equipamento musical eletrônico digital para torná-lo capaz de gerar ou reproduzir músicas. (MENEZES JUNIOR, 2007, p. 11)

O protocolo MIDI não armazena a onda sonora digitalizada, mas sim transcreve os dados de forma que possam ser executados pelo dispositivo MIDI, podendo conter informações como número de instrumentos, tonalidade e intensidade. Mas como isso é possível?

Tudo funciona com uma sequência de instruções. Por exemplo, supondo que se tem um teclado conectado a um computador ou caixa sintetizadora e quer-se tocar a nota “mi”. Quando pressionada a tecla “mi” no teclado, este mandará um comando MIDI “essa tecla está pressionada” para o computador. Lá, essa mensagem será

traduzida e a nota mi será reproduzida pelo dispositivo. Consequentemente, a mesma comunicação acontece quando se deixa de pressionar a tecla. (MENEZES JUNIOR, 2007, p. 11)

Além disso, o padrão MIDI torna possível também a criação de músicas mais ricas usando um sistema de camadas, onde pode-se reproduzir sons de inúmeros instrumentos e tocá-los em diferentes tons. O protocolo permite inclusive que o músico armazene dados e consiga alterá-los depois, podendo consertar erros, mudar a tonalidade, organizar o compasso e alterar diretamente a forma de onda. Tudo isso podendo ser alterado ou reproduzido por qualquer outro instrumento que possua a mesma arquitetura.

Uma de suas aplicações mais notáveis é sua utilização em *softwares* de criação de música. “With appropriate software, the computer can simulate a multitrack tape recorder using not just 4, 16, or 48 tracks but hundreds of tracks, and in some cases, unlimited number of tracks.” (ROTHSTEIN, 1995, p. 2)

Muitos anos se passaram desde sua invenção, e consequentemente, os computadores e suas entradas também foram modificados. As antigas CPUs possuíam inúmeras entradas e todas costumavam ter padrões diferentes, porém isso mudou quando surgiu a entrada USB, praticamente com o mesmo objetivo que o protocolo MIDI: a padronização. Não demorou muito para que a entrada MIDI também se atualizasse para o novo padrão, facilitando a conexão e até mesmo a comunicação com a CPU.

2.1.3.2 *Softwares* de geração de sons

Existem vários *softwares* relacionados à criação de música atualmente e todos possuem características distintas e que ajudam na criatividade do usuário.

Alguns exemplos *SuperCollider*, *ZynAddSubFX*, *Chuck* e *FL Studio*. Todos oferecem uma flexibilidade de elaboração ampla e não existente em meios tradicionais, porém muitas vezes falta nessa atividade uma interface mais intuitiva, mais próxima daquilo que um músico está acostumado a empregar.

Começando por *softwares* como *SuperCollider* e *ChucK*, pode-se notar uma interessante forma de reproduzir sons por meio de uma linguagem de programação.

Figura 8 - Exemplo de código utilizado no *SuperCollider*

```
// modulate a sine frequency and a noise amplitude with another sine
// whose frequency depends on the horizontal mouse pointer position
{
        var x = SinOsc.ar(MouseX.kr(1, 100));
        SinOsc.ar(300 * x + 800, 0, 0.1)
        +
        PinkNoise.ar(0.1 * x + 0.1)
}.play;
```

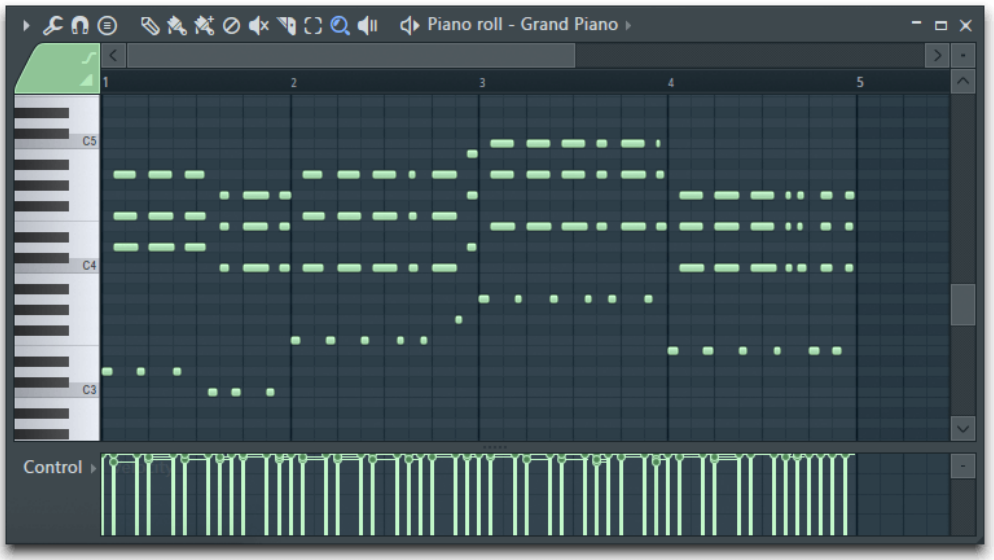
Fonte: SuperCollider (2019)

O código acima (Figura 8) produz um ruído cuja frequência depende da posição horizontal do cursor do mouse. É uma interação interessante e até mesmo dinâmica para a criação de sons, porém sua interface é pouco atrativa e também pouco intuitiva.

O *software ChucK* também usufrui de linguagem de programação para compor melodias, podendo até mesmo alterar o programa sem precisar parar a simulação, modificando o som em tempo real.

Já *softwares* como o *FL Studio* são mais complexos e possuem uma interface bem mais elaborada. Como pode-se observar na Figura 9, há elementos muito mais intuitivos na tela. O teclado permite uma fácil variação de tons e as linhas verticais permitem uma visualização ampla do tempo e compasso.

Figura 9 – Interface do *software FL Studio*



Fonte: FL Studio (2019)

Apesar disso, ainda é possível observar uma barreira entre o *software* em si e o instrumento musical físico.

2.2 PLATAFORMAS DE PROTOTIPAÇÃO PARA O PROJETO

2.2.1 Microcontroladores

Antigamente era extremamente raro algum contato com computadores e afins. Eles eram caros, pesados e continham pouca capacidade de processamento. Hoje, porém, tem-se computadores de todos os tamanhos e em todos os lugares, como *smart TVs*, consoles de videogame, *smartwatches* e *smartphones.* É como se a tecnologia tivesse crescido exponencialmente, o que de fato aconteceu, transformando computadores que tinham o tamanho de quartos inteiros em aparelhos que cabem na palma da mão.

No final da década de 60 e começo da década de 70, houve uma necessidade de criar equipamentos eletrônicos menores e mais acessíveis. Pensando justamente nisso, engenheiros da Intel foram designados a tal tarefa, voltando diminuir o número de circuitos integrados em vários tipos de aparelhos e assim “construir uma máquina capaz de mudar sua funcionalidade com base em um programa armazenado em uma memória.” (OLIVEIRA NETO; MONTEIRO; QUEIROGA, 2012, p.3)

Nisso houve surgimento do primeiro microprocessador e desde então esses pequenos eletrônicos não pararam de evoluir.

O microcontrolador é uma plataforma de prototipagem eletrônica capaz de executar inúmeras tarefas diferentes e ainda assim ser acoplado a outros tipos de aparelhos como sensores, circuitos em protoboards e computadores. Sua composição consiste em conjuntos de portas de entrada e saída, *timers*, contadores, memórias de dados, conversores analógico-digitais, modulação PWM, entre outras funções.

Todos esses componentes e funções tornaram possível a aplicação dos microcontroladores em robótica, engenharia mecânica, indústrias, automação e área musical.

Com a crescente popularidade do microcontrolador, outras empresas viram interesse e resolveram criar suas próprias plataformas de prototipagem. Assim surgiram nomes como *Raspberry Pi,* Arduino*, cloudBit, Photon* e ESP8266*.*

O *Raspberry Pi* (Figura 10) foi desenvolvido por Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft, que tinham como principal objetivo incentivar jovens a

programar desde cedo e promover o ensino de ciência da computação em instituições de ensino.

> “The Raspberry Pi computer has been immensely successful for countless electronic projects: Its low cost and accessibility have made it a favorite for do-it-yourself tinkering experiments and social purpose projects, from home control to data gathering on balloons to being shot into the upper atmosphere, and everything in-between. Despite being a general purpose computer, its low cost has also made it feasible to be used as a componente in single-purpose devices.” (JOHNSON; COX, 2017, p. 8)

Figura 10 - *Raspberry Pi* 3

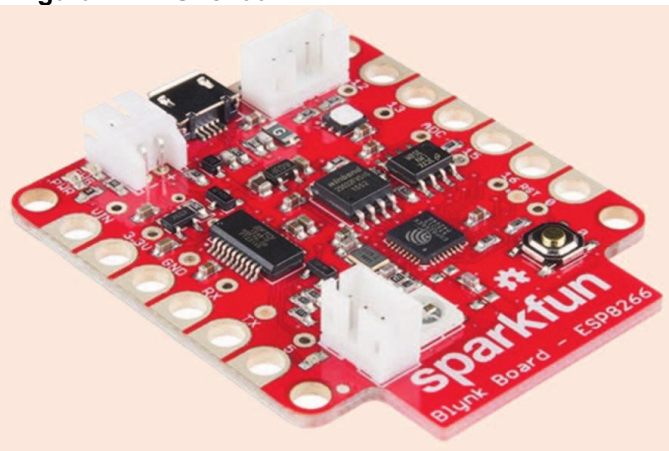
Fonte: Kapoor; Singh. (2017)

O modelo apresentado acima possui capacidade de processamento alta, além de contar com 4 portas USB, 1GB de memória RAM, 40 pinos de entrada e saída de sinal e uma entrada para cartão micro-SD. Seu preço aproximado é de R$270,00.

Já o ESP8266 (Figura 11) é mais compacto que o Raspberry e foi fabricado pela Espressif Systems.

Este dispositivo possui sistema de comunicação WiFi próprio, fazendo com que possa ser usado como módulo WiFi para outros microcontroladores e possui também características sofisticadas como processamento de sinais avançado, modo de economia de energia e Bluetooth. Um de seus aspectos mais atraente é o preço, custando em torno de R$50,00. (OLIVEIRA, 2017, p. 3)

Figura 11 - ESP8266



Fonte: Kapoor; Singh. (2017)

Neste trabalho, decidiu-se usar o microcontrolador Arduino por conta de seu preço acessível e sua simplicidade de uso, visto que isso torna desnecessário o uso de uma plataforma mais complexa e dispendiosa como um *Raspberry Pi*.

O Arduino surgiu em 2005, desenvolvido por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis. O objetivo de seus criadores era o mesmo que os de *Raspberry Pi*: criar um sistema de prototipagem de orçamento baixo e que pudesse ser utilizado amplamente em projetos escolares.

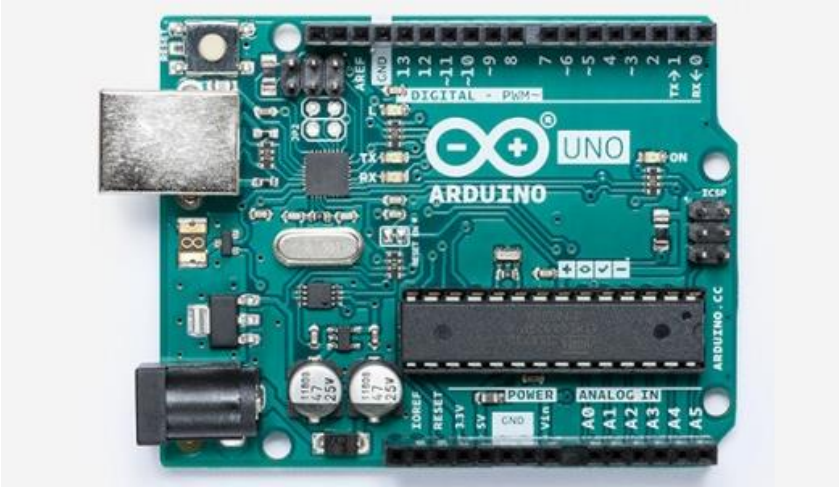
> “O Arduino é uma plataforma eletrônica de código aberto, que possui seus próprios microcontroladores e a sua própria IDE. O Arduino tem uma grande comunidade de usuários, diversos trabalhos e projetos que se utilizam dessa plataforma, pela sua facilidade de uso, eficiência e sua IDE gratuita, além do grande número de ferramentas já implementadas para o mesmo.” (OLIVEIRA, 2017, p. 8)

Recursos online oferecidos pelo fabricante fazem com que qualquer pessoa que possui a placa Arduino possa compartilhar e discutir com outros usuários todos seus projetos, criações, códigos e dúvidas. Além disso,

> The Arduino platform was designed to provide an inexpensive and easy way for hobbyists, students and professionals to create devices that interact with their environment using sensors and actuators. Based on simple microcontroller boards, it is an open source computing platform that is used for constructing and programming electronic devices. (LOUIS, 2016, p. 21)

Além disso, o Arduino possui inúmeras versões como UNO (Figura 12), MEGA, Leonardo, NANO entre várias outras. Seus preços variam entre R$50,00 até aproximadamente R$350,00, o que os torna relativamente acessíveis.

Figura 12 - Arduino UNO



Fonte: Store Arduino CC (2019). Acesso em: 22/10/2019.

A maioria vem com conexão USB, facilitando sua comunicação com o computador e, consequentemente, com sua IDE criada justamente para a placa.

2.2.2 Sensores

Sensores são aparelhos eletrônicos capazes de responder a um estímulo físico e traduzir esse dado para um valor numérico a fins de medição ou monitoramento. Esse estímulo pode ser causado por diversas formas, como luz, pressão, calor, entre outras possibilidades.

Esses aparelhos são essenciais para nossa sociedade, sendo amplamente usados na área de controle de qualidade, segurança, medicina e inúmeras aplicações industriais.

Felizmente, muitos desses sensores já possuem versões compatíveis com a placa Arduino e estão disponíveis por um preço acessível. São sensores simples, mas que podem ser muito úteis dependendo de suas aplicações.

Este trabalho apresenta uma análise de quais tipos de sensores são mais convenientes para modificar uma onda sonora usando diversos gestos. Os gestos mais simples que se pode pensar são o afastar e aproximar da mão, assim como o

movimento de rotação do pulso. Sendo assim, os sensores escolhidos e citados abaixo foram pensados nessas condições.

2.2.2.1 Sensor ultrassônico HC-SR04

O sensor Ultrassônico HC-SR04 (Figura 13) tem como princípio de funcionamento um emissor que produz um som em alta frequência, imperceptível para os ouvidos humanos, e quando esse som atinge um objeto ele é refletido e captado pelo receptor. Com essa comunicação é possível saber se há um objeto na frente do sensor, e mais ainda, qual a distância desse objeto.

Figura 13 - Sensor ultrassônico HC-SR04

Fonte: Create Arduino CC (2019). Acesso em: 09/09/2019.

Suas aplicações se resumem em medições como distâncias e níveis de enchimento para controle contra transbordamento de tanques. Apesar de simples e barato, esse sensor possui uma vantagem pois sua operação não é prejudicada pela opacidade de um objeto, ou seja, desde que o objeto reflita as ondas sonoras, é possível utilizá-lo. Devido a todas essas características, os sensores ultrassônicos são amplamente utilizados na indústria.

Neste trabalho será aplicado o princípio de medição *time-of-flight*.

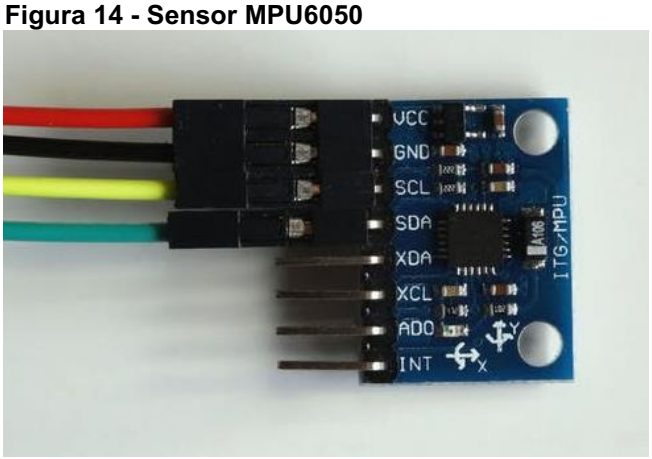
> The basic principle for the use of ultrasound as a measurement tool is the time-of-flight technique. The pulse-echo method is one example. In the pulse-echo method, a pulse of ultrasound is transmitted in a medium. When the pulse reaches an another medium, it is totally or partially reflected, and the elapsed time from emission to detection of the reflected pulse is measured. (PEDERSEN; KARLSSON, 1999, p.183)

Esta técnica utiliza da distância e do tempo de velocidade do som e será explicada com mais detalhes no desenvolvimento deste trabalho.

2.2.2.2 Sensor MPU6050

O MPU6050 (Figura 14) é um componente capaz de se localizar no espaço utilizando dois tipos de sensores: acelerômetro e giroscópio. Ambos têm a capacidade de reconhecer sua orientação no espaço, porém possuem princípios de funcionamento diferentes.

Figura 14 - Sensor MPU6050

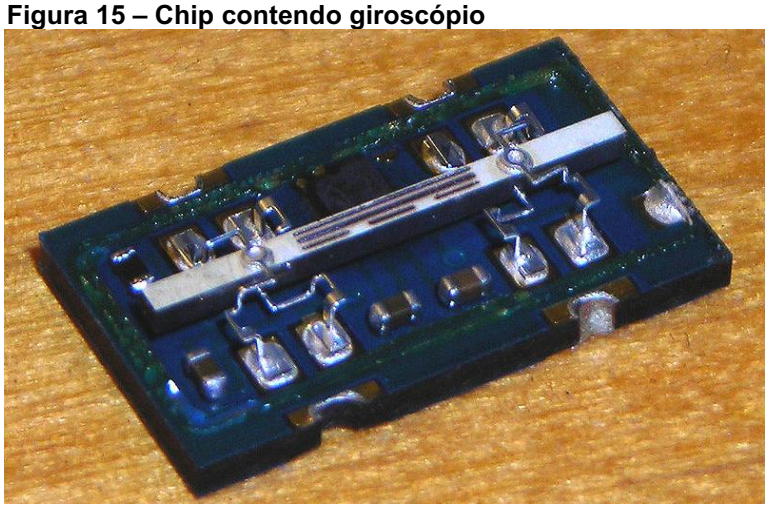


Fonte: Create Arduino CC (2017). Acesso em: 09/09/2019.

O acelerômetro usa como princípio de funcionamento o efeito piezoelétrico, que é a capacidade de certos cristais de gerarem tensão em resposta a uma pressão mecânica. Dentro do MPU6050 há um material com esse exato efeito, possibilitando uma passagem de corrente pelo componente que identifica com precisão a inclinação do mesmo. (STEDHAM et al., 1999, p. 321)

Já o giroscópio utiliza o princípio da inércia. Este dispositivo consiste em uma ou mais rodas livres que giram ao redor de um eixo e o mesmo é suspenso fazendo com que as rodas tenham liberdade de movimento. O movimento intenso de rotação faz com que o dispositivo tenha a tendência de se opor a qualquer tentativa de mudar sua direção original. Com essa resistência é possível saber exatamente se há mudanças na orientação de qualquer objeto que possua um giroscópio acoplado a ele. Na Figura 15 temos um exemplo de chip que se encontra dentro de helicópteros de controle remoto e utilizam do giroscópio para manter o equilíbrio do objeto no ar.

Figura 15 – Chip contendo giroscópio



Fonte: Wikimedia Commons (2004). Acesso em: 07/10/2020.

Ambos sensores citados acima são muito presentes em nosso dia a dia. Esses princípios são usados para acionar a rotação de tela de *tablets* e *smartphones*, e em situações mais extremas, os sensores mais sensíveis são usados como sismógrafos, no monitoramento de terremotos e de vibrações sísmicas. Os aparelhos que possuem acelerômetro também possibilitam mostrar exatamente a direção que o celular está apontado, facilitando navegação no GPS. Na aviação, giroscópios são essenciais na função de piloto automático, pois garantem que o avião permaneça estável em determinada rota. (STEDHAM et al., 1999, p. 320)

3 MATERIAIS E MÉTODOS

3.1 HARDWARE SELECIONADO

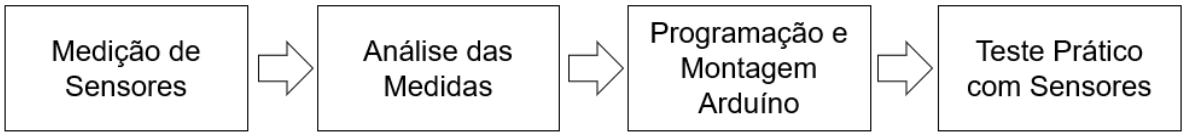
Para começar a aplicação dos sensores, decidiu-se que seria usado o dispositivo Arduino modelo UNO, por consequência de disponibilidade e por já constar com as funções e programações embutidas para a realização do trabalho. O dispositivo Raspberry Pi, apesar de possuir funções semelhantes ao Arduino, não constava disponível na instituição de ensino.

O primeiro passo para a criação de um movimento mais intuitivo seria escolher qual tipo de sensor será usado. Como o mais básico dos movimentos seria o aproximar e afastar das mãos e a rotação do pulso, os aparelhos escolhidos foram os sensores HC-SR04 e MPU6050, também por questão de disponibilidade.

3.2 METODOLOGIA DE TESTES

Os procedimentos para a realização dos testes estão resumidos conforme mostra o fluxograma abaixo (Figura 16).

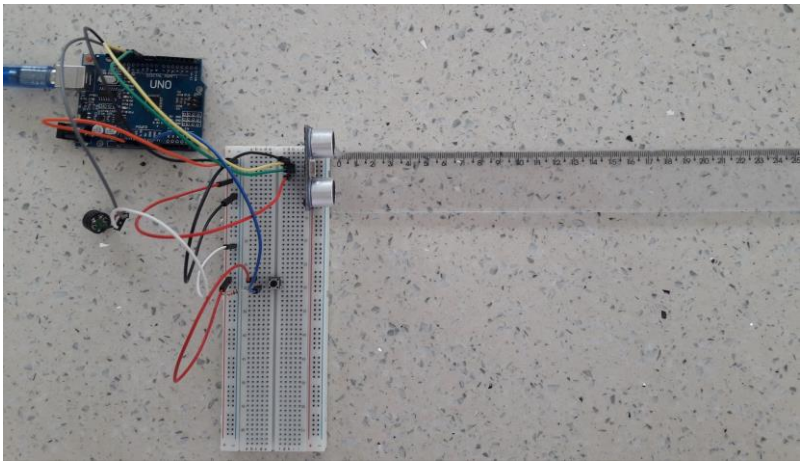
Figura 16 – Etapas da realização dos testes



Fonte: Autoria própria

Primeiramente, para fins de teste e confiabilidade, foi verificada a precisão do sensor. Para isso foi colocada uma régua sobre uma superfície nivelada e, com um anteparo posicionado logo à frente do sensor, foi medida a distância lida pelo mesmo e comparada com a medida real. A Fotografia 1 mostra como os testes foram realizados.

Fotografia 1 - Teste com sensor HC-SR04



Fonte: Autoria própria

Deve-se ressaltar que esse teste está sujeito a algumas fontes de erro. Apesar de usar fita adesiva para fixar a régua e a *protoboard,* qualquer movimento ou vibração pode alterar o resultado da medição lida pelo sensor. Como nem todos os testes foram feitos no mesmo local, várias vezes a posição da régua em relação ao sensor pode ter mudado, apesar de não ser uma mudança drástica. Outro adendo seria o quão imóvel é possível deixar o anteparo enquanto se faz a medida.

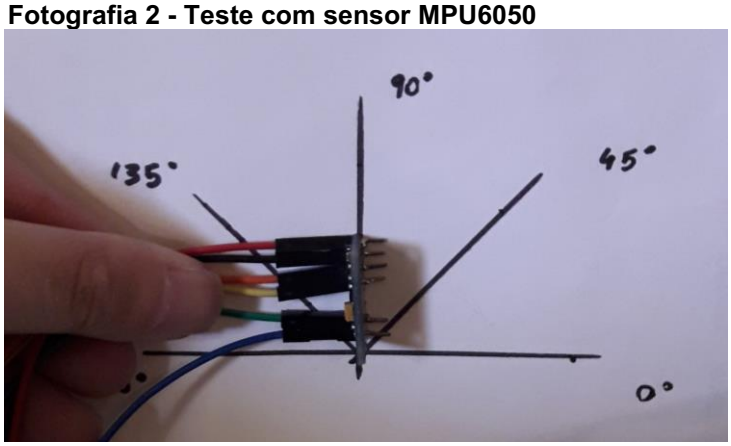
Fatores como temperatura e umidade também afetam o funcionamento do sensor, logo, como os testes foram feitos em dias diferentes, pode haver certa mudança em sua leitura final.

Antes de começar os testes de fato, era necessário saber qual era o alcance mínimo e máximo desse sensor. Para isso foi colocado um anteparo o mais próximo e depois o mais longe possível do HC-SR04, sem que houvesse flutuações nos valores.

Essas flutuações existem por dois motivos. O primeiro é devido ao HC-SR04 usar o som como princípio de funcionamento. Ondas sonoras não podem ser direcionadas, logo, se o som que o emissor emite bate em qualquer objeto em sua volta que não seja o anteparo e retorne ao receptor, esse valor será lido. O segundo é devido à distância mínima que o sensor consegue detectar. Quando o anteparo se encontra muito próximo ao emissor, o receptor não consegue receber a onda sonora refletida, portanto qualquer valor fora do comum foi ignorado.

Obviamente, se a proposta é usar os membros do corpo para o controle do sensor, não há necessidade de testar a sua confiabilidade em distâncias muito grandes. Logo, o primeiro teste consistiu em verificar a sua precisão em 1m de distância, medindo de 5 em 5 centímetros.

Fotografia 2 - Teste com sensor MPU6050



Fonte: Autoria própria

Após a verificação dos valores obtidos, decidiu-se fazer um segundo teste um pouco mais minucioso. A partir dos 5cm, foi medida a distância de 1 em 1cm até 25cm com o intuito de estudar o desvio padrão do sensor em distâncias menores.

Já com o sensor MPU6050, o teste foi baseado em seu ângulo de rotação. Nesse caso, foram desenhados ângulos de 45 em 45° em uma folha e a mesma foi fixada em uma parede, sendo possível assim o posicionamento do sensor logo à frente (Fotografia 2). Assim como o HC-SR04, as leituras podem variar dependendo da temperatura e umidade do ambiente.

4 DESENVOLVIMENTO

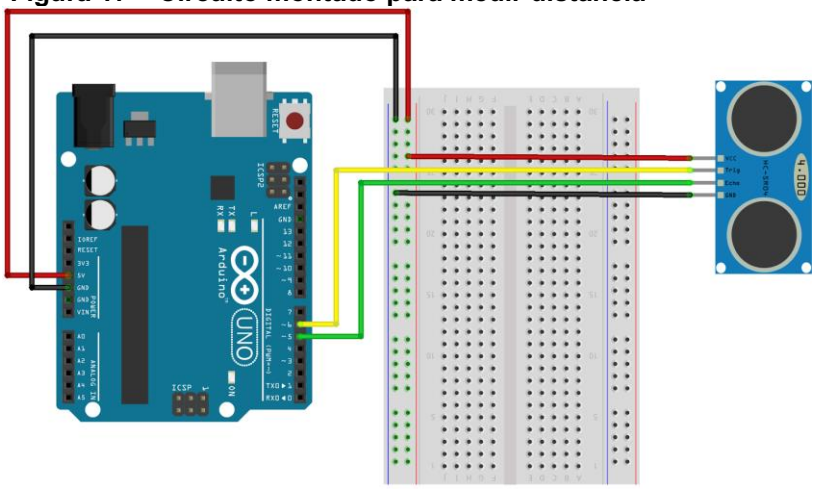
O desenvolvimento deste trabalho consiste em duas sessões. A primeira se resume em apresentar os dados coletados pelos sensores e depois realizar uma breve discussão sobre cada caso. A segunda fundamenta-se na aplicação dos sensores em um sintetizador, a fim de modificar a frequência das notas reproduzidas.

Todos os códigos usados para os testes estão disponíveis na internet e podem ser facilmente encontrados em comunidades como Arduino Create e GitHub.

4.1 TESTES COM SENSORES

Para os testes envolvendo distâncias foi montado o circuito conforme mostrado abaixo (Figura 17).

Figura 17 – Circuito montado para medir distância



Fonte: Autoria própria

A técnica usada para medir distância com este sensor é chamada de *time-of-flight*, como citada anteriormente, e o cálculo usado para isso é muito simples. Sabendo-se que a velocidade do som é de 340m/s, pode-se facilmente utilizar da equação da velocidade:

$$V = \frac{S}{t}$$

Onde *V* é a velocidade, *S* é a distância e *t* é o tempo. Para descobrir a distância basta isolar a variável *s*.

$$S = V.t$$

Como o sensor responde quando o som vai até um objeto e volta, é necessário que o resultado seja dividido por dois, pois assim tem-se a distância apenas de ida. Sabendo-se que a velocidade do som no ar é de 343 m/s, obtém-se a seguinte equação:

$$S = 343\ .t.\frac{1}{2}$$

Por conta dos testes, era conveniente obter o resultado em centímetros. Logo, aplicando uma simples conversão de unidades, tem-se:

$$S = 34300\ .t.\frac{1}{2}$$

E assim foram coletadas as primeiras medidas feitas pelo sensor, apresentadas na Tabela 1 abaixo.

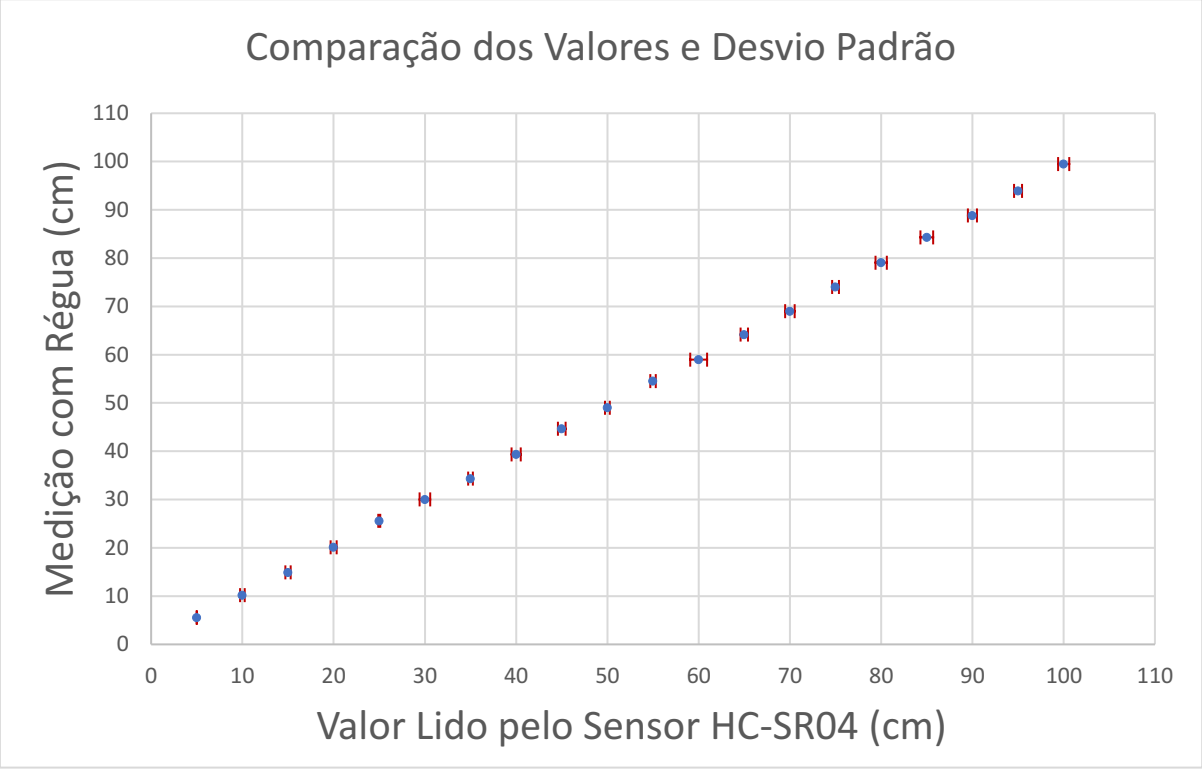
Tabela 1 - Distâncias lidas pelo sensor HC-SR04 em 1m

Sensor Ultrassônico HC-SR04 (Teste 1)										
Régua (cm)	**Teste 1**	**Teste 2**	**Teste 3**	**Teste 4**	**Teste 5**	**Teste 6**	**Teste 7**	**Teste 8**	**Média**	**Desvio Padrão**
5	5,5	5,8	5,22	5,68	5,71	5,35	5,93	5,32	5,56	0,254162
10	10,6	10,08	10,44	9,96	10,52	9,91	10,25	9,86	10,20	0,291339
15	15,3	14,99	14,43	14,58	14,78	15,41	14,93	14,92	14,92	0,330530
20	20,01	20,3	20,04	20,14	20,12	20,22	20,02	20,01	20,11	0,108332
25	24,5	26,2	25,47	25,06	26,15	26,02	25,62	25,73	25,59	0,582407
30	30,3	29,6	30,28	29,82	29,94	30,28	30,13	29,89	30,03	0,257627
35	34,66	34,66	34,2	33,34	34,23	35,06	34,22	34,34	34,34	0,502748
40	38,61	39,81	39,47	39,12	39,9	39,21	39,1	39,53	39,34	0,421424
45	44,81	44,73	44,44	44,22	45,05	44,74	44,83	44,51	44,67	0,261585
50	48,56	48,9	49,25	48,74	48,98	49,11	49,53	49,01	49,01	0,299142
55	55,34	56,1	53,43	53,43	54,22	54,67	54,11	54,91	54,53	0,924074
60	59,62	59,09	58,79	58,91	58,87	58,33	58,74	59,42	58,97	0,404949
65	64,86	64,77	64,07	63,36	64,32	63,88	64,43	63,69	64,17	0,521913
70	68,43	69	69,33	68,48	69,34	69,19	68,97	69,34	69,01	0,372559
75	73,24	73,7	75	73,42	74,21	74,73	74,21	73,72	74,03	0,620701
80	80,07	79,54	79,49	78,29	78,01	79,02	78,62	79,32	79,05	0,696091
85	84,64	84,46	83,96	83,42	84,35	84,89	84,72	83,89	84,29	0,496515
90	89,47	88,4	88,88	88,08	88,99	89,01	88,73	89,17	88,84	0,437866
95	93,4	93,8	95,37	93,74	93,73	93,54	93,84	93,95	93,92	0,610326
100	100,1	99,74	98,6	99,06	99,84	100,3	99,39	98,77	99,48	0,634101

Pode-se notar que o sensor é suficientemente preciso para um intervalo de 5 em 5cm, o que o torna viável para algo como a mudança de uma nota para outra sem muitas complicações.

Com os dados obtidos foi possível montar o Gráfico 1. Esse gráfico mostra a média dos valores obtidos pelo sensor e a marcação em vermelho representa seu desvio padrão. É possível notar que é um desvio extremamente baixo e que pode ser facilmente ignorado.

Gráfico 1 - Desvio padrão de 5 em 5cm na distância de 1m



Fonte: Autoria propria

Porém isso nos leva a uma pergunta. Até que ponto seria possível diminuir a distância entre notas sem que houvesse inconveniências como ruído entre elas?

Foi pensando nisso que o segundo teste foi feito. A distância foi medida de 1 em 1cm, no intervalo entre 5 e 25cm. Os resultados obtidos se encontram na Tabela 2 e o Gráfico 2 mostra o desvio padrão desse teste.

Pode-se notar um desvio muito maior nesse caso. Em valores como 14 e 15cm temos um desvio tão grande que chega muito próximo da medição seguinte.

Isso indica que seria inviável uma distância muito curta para uma mudança de nota. Porém, deve ser levado em consideração a precisão que um músico conseguiria movimentar a mão à frente do sensor. Uma mudança de tom em uma distância de 1

em 1cm é relativamente pequena para um controle de movimento. Logo, apesar do sensor não apresentar precisão suficiente para uma distância desta dimensão, em intervalos maiores não haveria problema.

Tabela 2 - Distâncias lidas pelo sensor HC-SR04 em 20cm

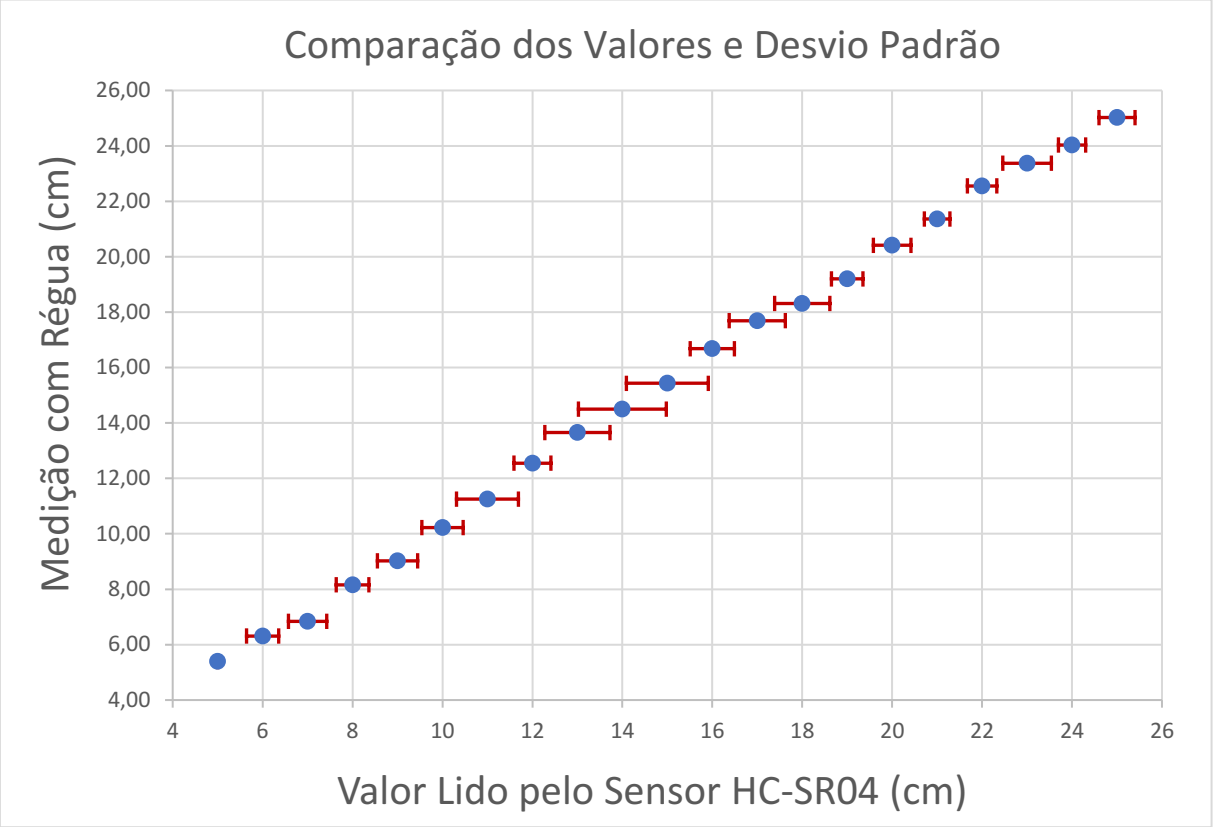
Sensor Ultrassônico HC-SR04 (Teste 2)										
Régua (cm)	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Teste 6	Teste 7	Teste 8	Média	Desvio Padrão
5	4,69	5,56	5,36	5,3	5,54	5,49	5,95	5,25	5,39	0,357281
6	6,41	6,58	5,95	5,85	6,72	6,6	6,72	5,66	6,31	0,425422
7	6,66	6,1	6,66	7,11	7,26	6,95	6,97	7,02	6,84	0,363139
8	7,84	8,77	7,67	8,77	7,67	8,26	8,33	7,97	8,16	0,447501
9	8,53	9,54	9,59	9,33	9,08	9,11	8,48	8,53	9,02	0,459003
10	11,75	9,69	10,03	10,64	9,98	9,78	9,74	10,18	10,22	0,688413
11	10,74	11,7	11,39	11,31	10,64	11,63	11	11,59	11,25	0,410644
12	13,63	13,16	13,26	12,61	11,85	12,09	12,1	11,7	12,55	0,724352
13	14,43	14,88	14,47	14,38	12,55	12,99	12,51	13,01	13,65	0,976974
14	15,03	15,54	15,8	14,52	13,41	13,4	13,92	14,37	14,50	0,910595
15	16,44	15,59	15,67	15,39	15,01	15,37	14,98	14,98	15,43	0,491337
16	16,95	17,66	17,09	17,1	15,73	16,26	16,29	16,34	16,68	0,623853
17	17,17	17,71	18,75	18,51	17,17	17,39	17,24	17,6	17,69	0,613858
18	18,29	18,22	18,9	17,75	18,14	18,24	18,26	18,7	18,31	0,350214
19	19,28	18,99	19,96	19,16	18,73	19,6	18,75	19,11	19,20	0,417672
20	20,18	20,28	20,86	20,52	20,65	20,57	20,03	20,18	20,41	0,284074
21	21,64	21,25	21,51	21,49	21,66	21,17	20,67	21,49	21,36	0,327109
22	21,79	22,61	22,47	22,44	22,41	22,08	23,29	23,36	22,56	0,540316
23	23,1	23,02	23,04	23,5	23,46	23,53	23,44	23,9	23,37	0,302510
24	24,82	23,61	23,68	23,7	24,07	24,04	24	24,33	24,03	0,400801
25	25,02	24,97	25,4	24,85	24,82	24,96	24,9	25,23	25,02	0,199244

Fonte: Autoria própria

Tendo em mente o alcance médio do braço humano, uma distância aproximada de 50cm seria ideal para distribuir uma oitava sem que houvesse problemas de ruído entre notas. Por exemplo, numa distância de 45cm, ignorar-se-iam os 5 primeiros centímetros e, a partir dali, haveria uma distribuição de uma oitava em 40cm, 5cm para cada tom. Além de tornar o ato mais cômodo, essa distância entre notas possibilita ao músico uma memória muscular melhor em relação a um intervalo menor.

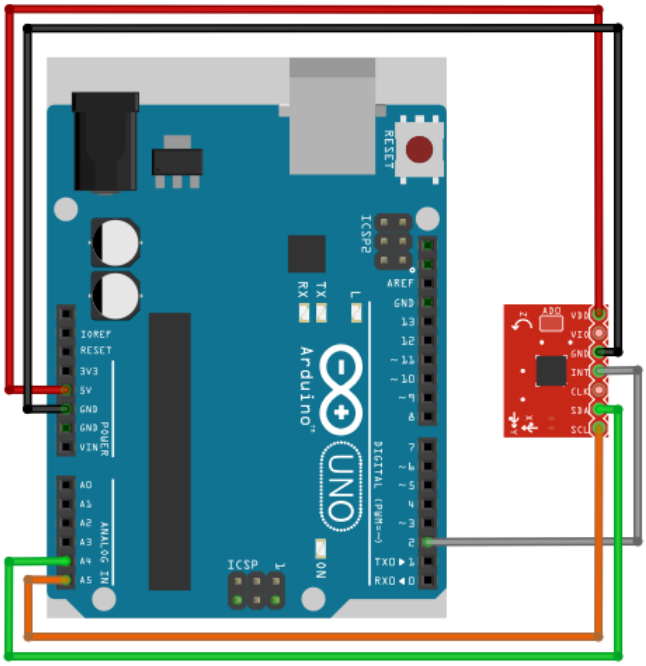
Depois disso foram realizados os testes com o acelerômetro e giroscópio MPU6050, visando novamente a confiabilidade do sensor. A Figura 18 mostra o esquema utilizado para tais testes.

Gráfico 2 - Desvio padrão de 1 em 1cm na distância de 20cm



Fonte: Autoria própria

Figura 18 – Circuito montado para medir ângulos



Fonte: Autoria própria

Houve certa dificuldade no início pois a maioria dos códigos encontrados utilizavam apenas o princípio do giroscópio para ler os ângulos, porém isso vinha com

um problema que seria mais tarde descoberto como sendo uma característica natural desse sensor. O giroscópio possui grande precisão em ângulos pequenos, até 90° por exemplo, mas ao rotacionar mais o aparelho, esse sensor tende a perder sua calibração inicial, desestabilizando o giroscópio e mostrando valores de ângulos longe dos reais.

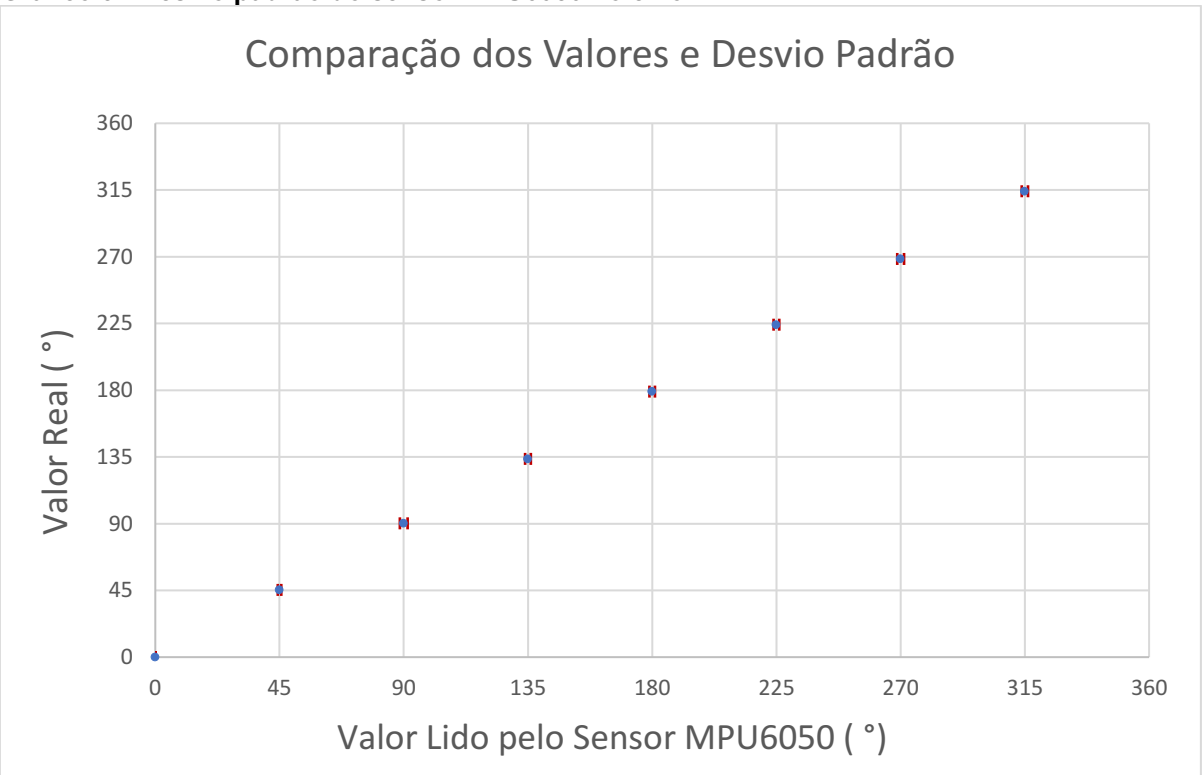
Porém, com a adição do uso do acelerômetro esse problema se extinguia, fazendo com que a leitura dos ângulos se tornasse muito mais precisa. Os valores obtidos neste teste se encontram na Tabela 3 abaixo.

Tabela 3 - Valores obitdos com o sensor MPU6050

Valor Real (°)	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Teste 6	Teste 7	Teste 8	Média	Desvio Padrão
0	-0,7	0,87	0,61	-0,04	0,47	-0,7	-0,54	0,41	0,04	0,629415
45	44,02	43,87	46,55	48,09	45,65	44,69	45,08	44,9	45,35	1,399366
90	88,79	91,03	89,87	92,04	90,69	89,79	89,27	90,45	90,24	1,034648
135	134,17	133,97	132,27	133,57	135,72	133,43	133,08	133,38	133,69	0,999691
180	176,75	179,72	179,41	179,96	178,78	179,8	179,09	179,1	179,07	1,023340
225	222,6	222,44	224,93	225,8	224,26	225,49	224,23	223,79	224,19	1,229793
270	266,5	267,69	267,96	269,56	269,33	269,69	268,14	269,28	268,51	1,128303
315	313,3	316,48	315,85	311,2	313,47	314,94	314,19	313,98	314,17	1,641227

Fonte: Autoria própria

Gráfico 3 - Desvio padrão do sensor MPU6050 no eixo X



Fonte: Autoria propria

No Grafico 3 pode-se perceber que o desvio padrão desse sensor na angulação do eixo “x” é muito baixo, se tornando até difícil de se visualizar. Os ângulos lidos nas orientações “y” e “z” se mostraram igualmente precisos, isso indica que esse sensor é preciso o suficiente para qualquer mudança gradual, seja de frequência ou de qualquer outra característica do som. Não só isso, cada eixo pode carregar uma variável diferente e alterá-la simultaneamente, abrindo um leque muito maior e mais experimental na criação de melodias.

Apesar do MPU6050 se apresentar mais preciso que o HC-SR04, ambos se mostraram precisos o suficiente para alterar o som e criar melodias a partir de gestos.

Há no mercado atualmente um instrumento com a mesma premissa chamado Karlax (2010). O Karlax (Figura 19) é um instrumento musical eletrônico que possui inúmeros sensores e teclas mecânicas. Suas funções podem ser alteradas por um *software* disponibilizado pela própria empresa que o fabrica, fazendo com que possua inúmeras maneiras de ser configurado de acordo com a preferência do usuário.

Entretanto, o Karlax possui uma inconveniência. Seu modelo mais simples custa €4700,00, ou seja, em torno de R$20.000,00. Esse preço o torna um instrumento pouco acessível e sem qualquer possibilidade de uso para instituições de ensino e músicos amadores.

Figura 19 - Karlax

Fonte: DAFACT (2010)

4.2 SINTETIZADOR

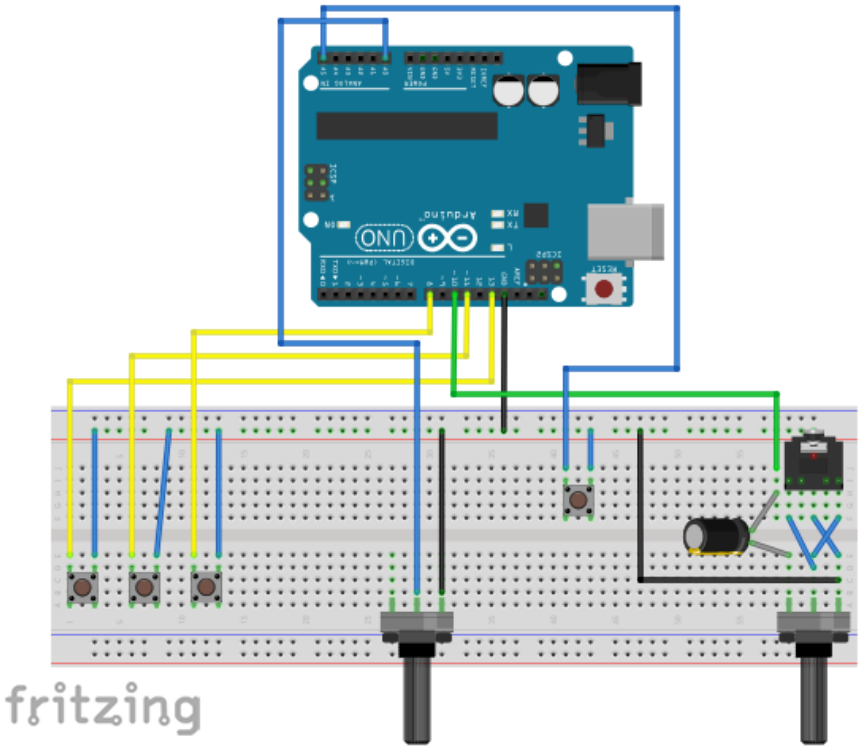
Para os testes com o sintetizador, foi utilizado o código feito por Rolf Oldeman (2019) presente no Anexo A, o qual possui um total de 12 timbres diferentes que se assemelham a instrumentos variados.

No código é possível observar a modulação do som acontecendo nas linhas 19 a 28 por meio de síntese FM e ADSR, fazendo com que cada instrumento receba seu timbre característico.

Primeiramente, para a tentativa de inclusão dos sensores como meio de controle do som, foram adicionadas duas variáveis chamadas “Gteste” e “Gteste2”. Essas variáveis são colocadas posteriormente dentro do *loop* de execução, das linhas 405 a 419. Esse loop é responsável por modificar a variável “Gteste2” de modo gradual para depois ser inserida dentro de um incremento do tom, modificando sua frequência.

Logo em seguida, foi declarado que “Gteste2” receberia os valores do pino A0. Assim, antes de conectar os sensores, foi feito um teste com um potenciômetro ligado a essa entrada. A Figura 20 mostra o esquema do circuito com o potenciômetro.

Figura 20 - Circuito para gerar um sintetizador



Fonte: Autoria própria

Ao compilar o código no Arduino, o mesmo funcionou da maneira esperada, modificando a frequência do som executado ao alterar o valor do potenciômetro. Porém, ao substituir o potenciômetro pelos sensores HC-SR04 e MPU6050, notou-se que o mesmo não acontecia.

Esse problema pode ser estudo de trabalhos futuros, os quais envolveriam um mergulho maior nas características físicas e construtivas dos sensores, verificando também se não haveria modelos melhores disponíveis para tal aplicação.

5 CONCLUSÃO

O avanço da eletrônica e da tecnologia possibilitou melhorar a experiência e interatividade na criação de música, dando oportunidade para músicos, desde amadores e principiantes até profissionais da área.

Este trabalho apresentou uma forma alternativa de controlar sons a partir de sensores simples e acessíveis, visando melhorar a interação entre o músico e o instrumento e tornando o processo de criação mais intuitivo.

A falta de um ambiente isolado e controlado fez com que houvesse certas imprecisões na execução dos testes, como por exemplo a necessidade de montar e desmontar os circuitos várias vezes entre a coleta de dados. Isso somado com a mudança de lugar dos experimentos, posição da *protoboard* em relação a régua e vibrações no anteparo fazem com que haja maior probabilidade de erros. Entretanto, este trabalho não exigia a princípio um grande nível de precisão na avaliação de comportamento dos sensores, tornando tais contratempos mais amenos.

Em relação aos códigos utilizados para a realização dos testes, todos se mostraram fáceis de se encontrar, entretanto a maioria dos códigos para o sensor MPU6050, ofereceram problemas na questão da leitura de ângulos por conta de um desvio natural do giroscópio. Porém, com uma busca mais ampla, foi possível achar códigos que contornassem esse problema.

Outro problema encontrado foi o fato do modelo Arduino UNO não possuir uma conexão USB nativa, logo não era possível usar *softwares* com comunicação MIDI.

Com mais tempo, seria interessante em trabalhos futuros usar um modelo mais avançado do Arduino para que o mesmo pudesse ser usado como controlador MIDI em *softwares* mais avançados como *FL Studio*. Seria interessante também a possibilidade de fazer mais testes com sensores diferentes, ampliando a variedade de gestos que podem ser usados na criação de melodias.

Ainda há muito a ser feito em relação a instrumentos para que sejam mais inclusivos e seria conveniente a criação de um sistema musical que não possua muita dependência visual e física, até certo ponto. O sintetizador apresentado anteriormente mostra que com algumas linhas de código, pode-se criar um instrumento que depende de movimentos mais simples para a execução de uma melodia, consequentemente tornando-o mais interativo e intuitivo.

REFERÊNCIAS

BIANCHI, A. J.; QUEIROZ, M. G. **Real time digital audio processing using Arduino**. In: Sound and Music Computing Conference, 2013, Estocolmo. Proceedings of the Sound and Music Computing Conference 2013. p. 538-545.

CANN, S. **How to Make a Noise:** A Comprehensive Guide to Synthesizer Programming. 1. Ed. New Malden: Coombe Hill, 2007, p 13-16.

COUTURIER, J. M. **Utilisation avancée d'interfaces graphiques dans le contrôle gestuel de processus sonores**. Tese (Doutorado ATIAM), Universidade Aix-Marseille II 2004, p. 9-18.

CREATE ARDUINO CC. **Convert acceleration to angle from MPU6050 I2C sensor.** 24 mar. 2017. Disponível em: <https://create.arduino.cc/projecthub/mitov/convert-acceleration-to-angle-from-mpu6050-i2c-sensor-ca6e4e?ref=search&ref_id=MPU6050&offset=1>. Acesso em: 9 set. 2019.

CREATE ARDUINO CC. **How to use na ultrasonic sensor.** 2 dez. 2018. Disponível em: **<**https://create.arduino.cc/projecthub/MisterBotBreak/how-to-use-an-ultrasonic-sensor-181cee**>.** Acesso em: 9 set. 2019.

DANTAS, J.D.; CRUZ S. S. Um olhar físico sobre a teoria musical. **Revista Brasileira de Ensino de Física**, Cuité (PB), v. 41, n. 1, p. 3, jun. 2019.

EREN, H.; FUNG, C. C. Inertial Navigation. In: WEBSTER, J. G. **The Measurement, Instrumentation and Sensors Handbook,** 1. ed. Florida: CRC Press, 1999. cap. 10.3, p. 337-342.

FL STUDIO. **FL Studio**. <https://www.image-line.com/flstudio/> Acesso em: 1 nov. 2019.

FOLLE, L. **Implementação de Síntese FM na Plataforma Arduino Due.** Monografia (Especialização) – Instituto de Informática, Universidade Federal do Rio Grande do Sul. Porto Alegre, 2015, p. 12.

HOLMES, T. B. **Electronic and Experimental Music:** Pioneers in Technology and Composition. 2. ed. New York: Routledge 2002, p.147-148.

KANE, S. K.; WOBBROCK, J. O.; LADNER, R. E. **Usable Gestures for Blind People: Understanding Preference and Performance.** 2018. Dissertação (Mestrado) - The Information School and Computer Science & Engineering, DUB Group, University of Washington. 2018, p.1-3.

KOLLING, M. Educational Programming on the Raspberry Pi. **Electronics**, Southampton (UK), v. 5, n. 3, p. 33, jun. 2016.

KUNG, D. **How Music and Mathematics Relate.** 1. ed. Virginia: The Great Courses, 2013**,** p. 99-105.

LOUIS, L. Working Principle of Arduino and Using It as a Tool for Study and Research. **International Journal of Control, Automation, Communication and Systems.** Ahmedabad (India), v. 1, n. 2, p. 21-29, abril 2016.

MARSHALL M.T.; WANDERLEY M.M. Evaluation of Sensors as Input Devices for Computer Music Interfaces. In: INTERNATIONAL SYMPOSIUM ON COMPUTER MUSIC MODELING AND RETRIEVAL. 3., 2005, Pisa. **Lecture Notes in Computer Science**, Montreal: McGill University, 2006. p. 130-139.

MENEZES JUNIOR, C. R. F. **Editor MIDI para violão com articulação humanizada nota a nota e qualidade acústica em linguagem funcional pura.** Dissertação (Mestrado), Universidade Federal de Uberlândia, 2007, p. 2-12.

OLDEMAN, R. **Arduino Synthesizer With FM.** Fev. 2019. Disponível em: <https://www.instructables.com/id/Arduino-Synthesizer-With-FM/> Acesso em: 3 set. 2019.

OLIVEIRA NETO, B. B; MONTEIRO, P. F.; QUEIROGA, S. L. M. Aplicabilidade dos Microcontroladores em Inovações Tecnológicas. In: VII CONGRESSO NORTE-NORDESTE DE PESQUISA E INOVAÇÃO DA REDE FEDERAL DE EDUCAÇÃO TECNOLÓGICA, 2012, Palmas. **Ciência, tecnologia e inovação: ações sustentáveis para o desenvolvimento regional.** Palmas: IFAM-AM, 2012. p. 3.

OLIVEIRA, R. R. **Uso do Microcontrolador ESP8266 para Automação Residencial.** 55 f. Trabalho de Conclusão de Curso (Graduação) - Projeto de Graduação, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2017. p. 3-9.

PEDERSEN, T.O; KARLSSON, N. Time-of-Flight Ultrasonic Displacement Sensors. In: WEBSTER, J. G. **The Measurement, Instrumentation and Sensors Handbook,** 1. ed. Florida: CRC Press, 1999. cap. 6.7, p. 183.

PEREIRA, M. C. **Matemática e Música De Pitágoras aos Dias de Hoje.** Dissertação (Mestrado), Universidade Federal do Estado do Rio de Janeiro, 2013, p. 16-23.

ROTHSTEIN, J. **MIDI:** A Comprehensive Introduction. 2. ed. Madison, Wisconsin: A-R Editions, Inc., 1995. p. 1-2.

STEDHAM, M. A, et al. Altitude Measurement. In: WEBSTER, J. G. **The Measurement, Instrumentation and Sensors Handbook,** 1. ed. Florida: CRC Press, 1999. cap. 10.2, p. 320-322.

STORE ARDUINO CC. **Arduino UNO ver 3.** Disponível em: <https://store.arduino.cc/usa/arduino-uno-rev3>. Acesso em: 22 out. 2019.

SUPERCOLLIDER. **SuperCollider**. Disponível em: <https://supercollider.github.io/>. Acesso em: 1 nov. 2019.

WIKIMEDIA COMMONS. **Gyro Chip-Esky-Lama v3 Model Helicopter**. Disponível em: <https://commons.wikimedia.org/wiki/File:Gyro_chip-Esky-Lama_v3_model_helicopter.jpg>. Acesso em: 07 out. 2020.

YAMAHA CORPORATION. **Yamaha Design "Synapses" DX7**. Disponível em: <https://www.yamaha.com/en/about/design/synapses/id_009/>. Acesso em: 11 dez. 2019.

ANEXO A - Código Utilizado Para Sintetizador

```
1. // https://www.instructables.com/id/Arduino-Synthesizer-With-
   FM/
2.
3. // Arduino polyphonic FM sound
4. // * 31250 Hz sampling rate
5. // * 9-bit resolution
6. // * 4-fold polyphony (4 different tones can play
   simulatenously)
7. // * FM-synthesis with time-varying modulation amplitude
8. // * ADSR envelopes
9. // * 12 preset instruments
10.     // Through PWM with timer1, sound is generated on pin 9
11.     // instrument-select button on A5
12.     // 18 sound keys on the remaining i/o pins
13.     // by Rolf Oldeman Feb 2019
14.     // Licence CC BY-NC-SA 2.5
   https://creativecommons.org/licenses/by-nc-sa/2.5/
15.
16.
17.     //instrument definitions
18.     #define ninstr 12           //   piano xlphn guitar cmbll
   bell funky vibr metal violin bass trumpt harm
19.     unsigned int ldness[ninstr]  = {   64,   64,   64,   64,
   64,   64,   64,   64,   64,   64,   64,   64}; // loudness
20.     unsigned int pitch0[ninstr]  = {   12,   12,   12,   12,
   24,   24,    0,   12,   24,   12,   12,   24}; // pitch of key0
21.     unsigned int ADSR_a[ninstr]  = { 4096, 8192, 8192, 8192,
   4096,  512,  512, 8192,  128,  128,  256,  256}; // attack
   parameter
22.     unsigned int ADSR_d[ninstr]  = {    8,   32,   16,   16,
   8,   16,   16,    8,   16,   16,   64,   32}; // decay
   parameter
23.     unsigned int ADSR_s[ninstr]  = {    0,    0,    0,    0,
   0,    0,    0,    0,  240,  240,  192,  192}; // sustain
   parameter
24.     unsigned int ADSR_r[ninstr]  = {   64,  128,   32,   32,
   16,   32,   32,   32,   32,   32,   64,   64}; // release
   parameter
25.     unsigned int FM_inc[ninstr]  = {  256,  512,  768,  400,
   200,   96,  528,  244,  256,  128,   64,  160}; // FM frequency
   wrt pitch
26.     unsigned int FM_a1[ninstr]  =  {  128,  512,  512, 1024,
   512,    0, 1024, 2048,  256,  256,  384,  256}; // FM amplitude
   start
27.     unsigned int FM_a2[ninstr]  =  {   64,    0,  128,  128,
   128,  512,  768,  512,  128,  128,  256,  128}; // FM amplitude
   end
28.     unsigned int FM_dec[ninstr]  = {   64,  128,  128,  128,
   32,  128,  128,  128,  128,  128,   64,   64}; // FM decay
29.
30.
31.
32.     //define the pitch2key mapping
33.     #define keyC4   0
34.     #define keyC4s  1
35.     #define keyD4   2
```

```
36.     #define keyD4s  3
37.     #define keyE4   4
38.     #define keyF4   5
39.     #define keyF4s  6
40.     #define keyG4   7
41.     #define keyG4s  8
42.     #define keyA4   9
43.     #define keyA4s 10
44.     #define keyB4  11
45.     #define keyC5  12
46.     #define keyC5s 13
47.     #define keyD5  14
48.     #define keyD5s 15
49.     #define keyE5  16
50.     #define keyF5  17
51.
52.     #define nokey 255
53.     #define instrkey 254
54.
55.     //define the pin to key mapping for 18-key keyboard
56.     #define pinD0 keyC5    //Arduino pin D0
57.     #define pinD1 keyB4    //Arduino pin D1
58.     #define pinD2 keyA4s   //Arduino pin D2
59.     #define pinD3 keyA4    //Arduino pin D3
60.     #define pinD4 keyG4s   //Arduino pin D4
61.     #define pinD5 keyG4    //Arduino pin D5
62.     #define pinD6 keyF4s   //Arduino pin D6
63.     #define pinD7 keyF4    //Arduino pin D7
64.     #define pinB0 keyE4    //Arduino pin D8
65.     #define pinB1 nokey    //Arduino pin D9  used for audio
   out
66.     #define pinB2 keyD4s   //Arduino pin D10
67.     #define pinB3 keyD4    //Arduino pin D11
68.     #define pinB4 keyC4s   //Arduino pin D12
69.     #define pinB5 keyC4    //Arduino pin D13
70.     #define pinB6 nokey    //Arduino pin D14 inexistent
71.     #define pinB7 nokey    //Arduino pin D15 inexistent
72.     #define pinC0 keyC5s   //Arduino pin A0
73.     #define pinC1 keyD5    //Arduino pin A1
74.     #define pinC2 keyD5s   //Arduino pin A2
75.     #define pinC3 keyE5    //Arduino pin A2
76.     #define pinC4 keyF5    //Arduino pin A3
77.     #define pinC5 instrkey //Arduino pin A4
78.     #define pinC6 nokey    //Arduino pin A5 inexistent
79.     #define pinC7 nokey    //Arduino pin A6 inexistent
80.
81.
82.     //set up array with sine values in signed 8-bit numbers
83.     const float pi = 3.14159265;
84.     char sine[256];
85.     void setsine() {
86.         for (int i = 0; i < 256; ++i) {
87.              sine[i] = (sin(2 * 3.14159265 * (i + 0.5) /
   256)) * 128;
88.       }
89.     }
```

```
90.
91.     //setup frequencies/phase increments, starting at C3=0 to
   B6. (A4 is defined as 440Hz)
92.     unsigned int tone_inc[48];
93.     void settones() {
94.       for (byte i=0; i<48; i++){
95.        tone_inc[i]= 440.0 * pow(2.0, ( (i-21) / 12.0)) *
   65536.0 / (16000000.0/512) + 0.5;
96.       }
97.     }
98.
99.     byte butstatD=0;
100.    byte butstatB=0;
101.    byte butstatC=0;
102.    byte prevbutstatD=0;
103.    byte prevbutstatB=0;
104.    byte prevbutstatC=0;
105.
106.    byte instr=0;
107.
108.    void setup() {
109.
110.    //disable all inerrupts to avoid glitches
111.    noInterrupts();
112.
113.    //setup the array with sine values
114.    setsine();
115.
116.    //setup array with tone frequency phase increments
117.    settones();
118.
119.    //Set a fast PWM signal on TIMER1A, 9-bit resolution,
   31250Hz
120.    pinMode(9, OUTPUT);
121.    TCCR1A = 0B10000010; //9-bit fast PWM
122.    TCCR1B = 0B00001001;
123.
124.    //setup pins for input with pull-up
125.    if(pinD0 != nokey){DDRD &=~(1<<0);PORTD |=(1<<0);};
126.    if(pinD1 != nokey){DDRD &=~(1<<1);PORTD |=(1<<1);};
127.    if(pinD2 != nokey){DDRD &=~(1<<2);PORTD |=(1<<2);};
128.    if(pinD3 != nokey){DDRD &=~(1<<3);PORTD |=(1<<3);};
129.    if(pinD4 != nokey){DDRD &=~(1<<4);PORTD |=(1<<4);};
130.    if(pinD5 != nokey){DDRD &=~(1<<5);PORTD |=(1<<5);};
131.    if(pinD6 != nokey){DDRD &=~(1<<6);PORTD |=(1<<6);};
132.    if(pinD7 != nokey){DDRD &=~(1<<7);PORTD |=(1<<7);};
133.    if(pinB0 != nokey){DDRB &=~(1<<0);PORTB |=(1<<0);};
134.    if(pinB1 != nokey){DDRB &=~(1<<1);PORTB |=(1<<1);};
135.    if(pinB2 != nokey){DDRB &=~(1<<2);PORTB |=(1<<2);};
136.    if(pinB3 != nokey){DDRB &=~(1<<3);PORTB |=(1<<3);};
137.    if(pinB4 != nokey){DDRB &=~(1<<4);PORTB |=(1<<4);};
138.    if(pinB5 != nokey){DDRB &=~(1<<5);PORTB |=(1<<5);};
139.    if(pinB6 != nokey){DDRB &=~(1<<6);PORTB |=(1<<6);};
140.    if(pinB7 != nokey){DDRB &=~(1<<7);PORTB |=(1<<7);};
141.    if(pinC0 != nokey){DDRC &=~(1<<0);PORTC |=(1<<0);};
```

```
142.    if(pinC1 != nokey){DDRC &=~(1<<1);PORTC |=(1<<1);};
143.    if(pinC2 != nokey){DDRC &=~(1<<2);PORTC |=(1<<2);};
144.    if(pinC3 != nokey){DDRC &=~(1<<3);PORTC |=(1<<3);};
145.    if(pinC4 != nokey){DDRC &=~(1<<4);PORTC |=(1<<4);};
146.    if(pinC5 != nokey){DDRC &=~(1<<5);PORTC |=(1<<5);};
147.    if(pinC6 != nokey){DDRC &=~(1<<6);PORTC |=(1<<6);};
148.    if(pinC7 != nokey){DDRC &=~(1<<7);PORTC |=(1<<7);};
149.
150.    //store button setting at startup
151.    butstatD = PIND;
152.    butstatB = PINB;
153.    butstatC = PINC;
154.    }
155.
156.
157.    //initialize the main parameters of the pulse length
   setting
158.    #define nch 4 //number of channels that can produce sound
   simultaneously
159.    unsigned int phase[nch]  = {0,0,0,0};
160.    int          inc[nch]    = {0,0,0,0};
161.    byte         amp[nch]    = {0,0,0,0};
162.    unsigned int FMphase[nch]= {0,0,0,0};
163.    unsigned int FMinc[nch]  = {0,0,0,0};
164.    unsigned int FMamp[nch]  = {0,0,0,0};
165.
166.    // main function (forced inline) to update the pulse
   length
167.    inline void setPWM() __attribute__((always_inline));
168.
169.    inline void setPWM() {
170.
171.    //wait for the timer to complete loop
172.    while ((TIFR1 & 0B00000001) == 0);
173.
174.    //Clear(!) the overflow bit by writing a 1 to it
175.    TIFR1 |= 0B00000001;
176.
177.    //increment the phases of the FM
178.    FMphase[0] += FMinc[0];
179.    FMphase[1] += FMinc[1];
180.    FMphase[2] += FMinc[2];
181.    FMphase[3] += FMinc[3];
182.
183.    //increment the phases of the note
184.    phase[0] += inc[0];
185.    phase[1] += inc[1];
186.    phase[2] += inc[2];
187.    phase[3] += inc[3];
188.
189.    //calculate the output value and set pulse width for
   timer2
190.    int val = sine[(phase[0]+sine[FMphase[0]>>8]*FMamp[0]) >>
   8] * amp[0];
191.    val += sine[(phase[1]+sine[FMphase[1]>>8]*FMamp[1]) >> 8]
   * amp[1];
```

```
192.    val += sine[(phase[2]+sine[FMphase[2]>>8]*FMamp[2]) >> 8]
   * amp[2];
193.    val += sine[(phase[3]+sine[FMphase[3]>>8]*FMamp[3]) >> 8]
   * amp[3];
194.
195.    //set the pulse length
196.    OCR1A = val/128 + 256;
197.    }
198.
199.    //properties of each note played
200.    byte         iADSR[nch]     = {0, 0, 0, 0};
201.    unsigned int envADSR[nch]   = {0, 0, 0, 0};
202.    unsigned int ADSRa[nch]     = {0, 0, 0, 0};
203.    unsigned int ADSRd[nch]     = {0, 0, 0, 0};
204.    unsigned int ADSRs[nch]     = {0, 0, 0, 0};
205.    unsigned int ADSRr[nch]     = {0, 0, 0, 0};
206.    byte         amp_base[nch]  = {0, 0, 0, 0};
207.    unsigned int inc_base[nch]  = {0, 0, 0, 0};
208.    unsigned int FMa0[nch]      = {0, 0, 0, 0};
209.    int          FMda[nch]      = {0, 0, 0, 0};
210.    unsigned int FMinc_base[nch]= {0, 0, 0, 0};
211.    unsigned int FMdec[nch]     = {0, 0, 0, 0};
212.    unsigned int FMexp[nch]     = {0, 0, 0, 0};
213.    unsigned int FMval[nch]     = {0, 0, 0, 0};
214.    byte         keych[nch]     = {0, 0, 0, 0};
215.    unsigned int tch[nch]       = {0, 0, 0, 0};
216.
217.
218.    //Variável nova inserida para a mudança do tom com sensor
219.    unsigned int Gteste = 0;
220.    unsigned int Gteste2 = 0;
221.
222.
223.    // main loop. Duration of loop is determined by number of
   setPWM calls
224.    // Each setPWMcall corresponds to 512 cylcles=32mus
225.    // Tloop= 32mus * #setPWM. #setPWM=15 gives Tloop=0.48ms
226.    void loop() {
227.
228.    //read and interpret input buttons
229.    prevbutstatD = butstatD;
230.    prevbutstatB = butstatB;
231.    prevbutstatC = butstatC;
232.    butstatD = PIND;
233.    butstatB = PINB;
234.    butstatC = PINC;
235.    byte keypressed = nokey;
236.    byte keyreleased = nokey;
237.    if(butstatD!=prevbutstatD){
238.       if ( pinD0!=nokey and (butstatD & (1<<0)) == 0 and
   (prevbutstatD & (1<<0)) >  0 ) keypressed  = pinD0;
239.       if ( pinD0!=nokey and (butstatD & (1<<0)) >  0 and
   (prevbutstatD & (1<<0)) == 0 ) keyreleased = pinD0;
240.       if ( pinD1!=nokey and (butstatD & (1<<1)) == 0 and
   (prevbutstatD & (1<<1)) >  0 ) keypressed  = pinD1;
```

```
241.       if ( pinD1!=nokey and (butstatD & (1<<1)) >  0 and
   (prevbutstatD & (1<<1)) == 0 ) keyreleased = pinD1;
242.       if ( pinD2!=nokey and (butstatD & (1<<2)) == 0 and
   (prevbutstatD & (1<<2)) >  0 ) keypressed  = pinD2;
243.       if ( pinD2!=nokey and (butstatD & (1<<2)) >  0 and
   (prevbutstatD & (1<<2)) == 0 ) keyreleased = pinD2;
244.       if ( pinD3!=nokey and (butstatD & (1<<3)) == 0 and
   (prevbutstatD & (1<<3)) >  0 ) keypressed  = pinD3;
245.       if ( pinD3!=nokey and (butstatD & (1<<3)) >  0 and
   (prevbutstatD & (1<<3)) == 0 ) keyreleased = pinD3;
246.       if ( pinD4!=nokey and (butstatD & (1<<4)) == 0 and
   (prevbutstatD & (1<<4)) >  0 ) keypressed  = pinD4;
247.       if ( pinD4!=nokey and (butstatD & (1<<4)) >  0 and
   (prevbutstatD & (1<<4)) == 0 ) keyreleased = pinD4;
248.       if ( pinD5!=nokey and (butstatD & (1<<5)) == 0 and
   (prevbutstatD & (1<<5)) >  0 ) keypressed  = pinD5;
249.       if ( pinD5!=nokey and (butstatD & (1<<5)) >  0 and
   (prevbutstatD & (1<<5)) == 0 ) keyreleased = pinD5;
250.       if ( pinD6!=nokey and (butstatD & (1<<6)) == 0 and
   (prevbutstatD & (1<<6)) >  0 ) keypressed  = pinD6;
251.       if ( pinD6!=nokey and (butstatD & (1<<6)) >  0 and
   (prevbutstatD & (1<<6)) == 0 ) keyreleased = pinD6;
252.       if ( pinD7!=nokey and (butstatD & (1<<7)) == 0 and
   (prevbutstatD & (1<<7)) >  0 ) keypressed  = pinD7;
253.       if ( pinD7!=nokey and (butstatD & (1<<7)) >  0 and
   (prevbutstatD & (1<<7)) == 0 ) keyreleased = pinD7;
254.    }
255.    if(butstatB!=prevbutstatB){
256.       if ( pinB0!=nokey and (butstatB & (1<<0)) == 0 and
   (prevbutstatB & (1<<0)) >  0 ) keypressed  = pinB0;
257.       if ( pinB0!=nokey and (butstatB & (1<<0)) >  0 and
   (prevbutstatB & (1<<0)) == 0 ) keyreleased = pinB0;
258.       if ( pinB1!=nokey and (butstatB & (1<<1)) == 0 and
   (prevbutstatB & (1<<1)) >  0 ) keypressed  = pinB1;
259.       if ( pinB1!=nokey and (butstatB & (1<<1)) >  0 and
   (prevbutstatB & (1<<1)) == 0 ) keyreleased = pinB1;
260.       if ( pinB2!=nokey and (butstatB & (1<<2)) == 0 and
   (prevbutstatB & (1<<2)) >  0 ) keypressed  = pinB2;
261.       if ( pinB2!=nokey and (butstatB & (1<<2)) >  0 and
   (prevbutstatB & (1<<2)) == 0 ) keyreleased = pinB2;
262.       if ( pinB3!=nokey and (butstatB & (1<<3)) == 0 and
   (prevbutstatB & (1<<3)) >  0 ) keypressed  = pinB3;
263.       if ( pinB3!=nokey and (butstatB & (1<<3)) >  0 and
   (prevbutstatB & (1<<3)) == 0 ) keyreleased = pinB3;
264.       if ( pinB4!=nokey and (butstatB & (1<<4)) == 0 and
   (prevbutstatB & (1<<4)) >  0 ) keypressed  = pinB4;
265.       if ( pinB4!=nokey and (butstatB & (1<<4)) >  0 and
   (prevbutstatB & (1<<4)) == 0 ) keyreleased = pinB4;
266.       if ( pinB5!=nokey and (butstatB & (1<<5)) == 0 and
   (prevbutstatB & (1<<5)) >  0 ) keypressed  = pinB5;
267.       if ( pinB5!=nokey and (butstatB & (1<<5)) >  0 and
   (prevbutstatB & (1<<5)) == 0 ) keyreleased = pinB5;
268.       if ( pinB6!=nokey and (butstatB & (1<<6)) == 0 and
   (prevbutstatB & (1<<6)) >  0 ) keypressed  = pinB6;
269.       if ( pinB6!=nokey and (butstatB & (1<<6)) >  0 and
   (prevbutstatB & (1<<6)) == 0 ) keyreleased = pinB6;
```

```
270.       if ( pinB7!=nokey and (butstatB & (1<<7)) == 0 and
   (prevbutstatB & (1<<7)) >  0 ) keypressed  = pinB7;
271.       if ( pinB7!=nokey and (butstatB & (1<<7)) >  0 and
   (prevbutstatB & (1<<7)) == 0 ) keyreleased = pinB7;
272.    }
273.    if(butstatC!=prevbutstatC){
274.       if ( pinC0!=nokey and (butstatC & (1<<0)) == 0 and
   (prevbutstatC & (1<<0)) >  0 ) keypressed  = pinC0;
275.       if ( pinC0!=nokey and (butstatC & (1<<0)) >  0 and
   (prevbutstatC & (1<<0)) == 0 ) keyreleased = pinC0;
276.       if ( pinC1!=nokey and (butstatC & (1<<1)) == 0 and
   (prevbutstatC & (1<<1)) >  0 ) keypressed  = pinC1;
277.       if ( pinC1!=nokey and (butstatC & (1<<1)) >  0 and
   (prevbutstatC & (1<<1)) == 0 ) keyreleased = pinC1;
278.       if ( pinC2!=nokey and (butstatC & (1<<2)) == 0 and
   (prevbutstatC & (1<<2)) >  0 ) keypressed  = pinC2;
279.       if ( pinC2!=nokey and (butstatC & (1<<2)) >  0 and
   (prevbutstatC & (1<<2)) == 0 ) keyreleased = pinC2;
280.       if ( pinC3!=nokey and (butstatC & (1<<3)) == 0 and
   (prevbutstatC & (1<<3)) >  0 ) keypressed  = pinC3;
281.       if ( pinC3!=nokey and (butstatC & (1<<3)) >  0 and
   (prevbutstatC & (1<<3)) == 0 ) keyreleased = pinC3;
282.       if ( pinC4!=nokey and (butstatC & (1<<4)) == 0 and
   (prevbutstatC & (1<<4)) >  0 ) keypressed  = pinC4;
283.       if ( pinC4!=nokey and (butstatC & (1<<4)) >  0 and
   (prevbutstatC & (1<<4)) == 0 ) keyreleased = pinC4;
284.       if ( pinC5!=nokey and (butstatC & (1<<5)) == 0 and
   (prevbutstatC & (1<<5)) >  0 ) keypressed  = pinC5;
285.       if ( pinC5!=nokey and (butstatC & (1<<5)) >  0 and
   (prevbutstatC & (1<<5)) == 0 ) keyreleased = pinC5;
286.       if ( pinC6!=nokey and (butstatC & (1<<6)) == 0 and
   (prevbutstatC & (1<<6)) >  0 ) keypressed  = pinC6;
287.       if ( pinC6!=nokey and (butstatC & (1<<6)) >  0 and
   (prevbutstatC & (1<<6)) == 0 ) keyreleased = pinC6;
288.       if ( pinC7!=nokey and (butstatC & (1<<7)) == 0 and
   (prevbutstatC & (1<<7)) >  0 ) keypressed  = pinC7;
289.       if ( pinC7!=nokey and (butstatC & (1<<7)) >  0 and
   (prevbutstatC & (1<<7)) == 0 ) keyreleased = pinC7;
290.    }
291.
292.    setPWM(); //#1
293.
294.    //change instrument if instrument select button is
   pressed
295.    if ( keypressed==instrkey) {
296.       instr++;
297.       if (instr>=ninstr) instr=0;
298.       keypressed=keyA4;
299.    }
300.    if (keyreleased==instrkey) keyreleased=keyA4;
301.
302.    setPWM(); //#2
303.
304.    //find the best channel to start a new note
305.    byte nextch = 255;
306.    //first check if the key is still being played
```

```
307.    if (iADSR[0] > 0 and keypressed == keych[0])nextch = 0;
308.    if (iADSR[1] > 0 and keypressed == keych[1])nextch = 1;
309.    if (iADSR[2] > 0 and keypressed == keych[2])nextch = 2;
310.    if (iADSR[3] > 0 and keypressed == keych[3])nextch = 3;
311.    //then check for an empty channel
312.    if (nextch == 255) {
313.       if (iADSR[0] == 0)nextch = 0;
314.       if (iADSR[1] == 0)nextch = 1;
315.       if (iADSR[2] == 0)nextch = 2;
316.       if (iADSR[3] == 0)nextch = 3;
317.    }
318.    //otherwise use the channel with the longest playing note
319.    if (nextch == 255) {
320.       nextch = 0;
321.       if (tch[0] > tch[nextch])nextch = 0;
322.       if (tch[1] > tch[nextch])nextch = 1;
323.       if (tch[2] > tch[nextch])nextch = 2;
324.       if (tch[3] > tch[nextch])nextch = 3;
325.    }
326.
327.
328.
329.    nextch = 0;
330.
331.    setPWM(); //#3
332.
333.
334.    //initiate new note if needed
335.    if (keypressed != nokey) {
336.
337.       phase[nextch]=0;
338.       amp_base[nextch] = ldness[instr];
339.       inc_base[nextch] = tone_inc[pitch0[instr]+keypressed];
340.       ADSRa[nextch]=ADSR_a[instr];
341.       ADSRd[nextch]=ADSR_d[instr];
342.       ADSRs[nextch]=ADSR_s[instr]<<8;
343.       ADSRr[nextch]=ADSR_r[instr];
344.       iADSR[nextch] = 1;
345.       FMphase[nextch]=0;
346.       FMinc_base[nextch] =
   ((long)inc_base[nextch]*FM_inc[instr])/256;
347.       FMa0[nextch] = FM_a2[instr];
348.       FMda[nextch] = FM_a1[instr]-FM_a2[instr];
349.       FMexp[nextch]=0xFFFF;
350.       FMdec[nextch]=FM_dec[instr];
351.       keych[nextch] = keypressed;
352.       tch[nextch] = 0;
353.    }
354.
355.    setPWM(); //#4
356.
357.    //stop a note if the button is released
358.    if (keyreleased != nokey) {
359.        if (keych[0] == keyreleased)iADSR[0] = 4;
360.        if (keych[1] == keyreleased)iADSR[1] = 4;
361.        if (keych[2] == keyreleased)iADSR[2] = 4;
```

```
362.        if (keych[3] == keyreleased)iADSR[3] = 4;
363.    }
364.
365.    setPWM(); //#5
366.
367.    //update FM decay exponential
368.    FMexp[0]-=(long)FMexp[0]*FMdec[0]>>16;
369.    FMexp[1]-=(long)FMexp[1]*FMdec[1]>>16;
370.    FMexp[2]-=(long)FMexp[2]*FMdec[2]>>16;
371.    FMexp[3]-=(long)FMexp[3]*FMdec[3]>>16;
372.
373.    setPWM(); //#6
374.
375.    //adjust the ADSR envelopes
376.    for (byte ich = 0; ich < nch; ich++) {
377.        if (iADSR[ich] == 4) {
378.            if (envADSR[ich] <= ADSRr[ich]) {
379.              envADSR[ich] = 0;
380.              iADSR[ich] = 0;
381.            }
382.            else envADSR[ich] -= ADSRr[ich];
383.        }
384.        if (iADSR[ich] == 2) {
385.           if (envADSR[ich] <= (ADSRs[ich] + ADSRd[ich])) {
386.              envADSR[ich] = ADSRs[ich];
387.              iADSR[ich] = 3;
388.           }
389.        else envADSR[ich] -= ADSRd[ich];
390.        }
391.        if (iADSR[ich] == 1) {
392.            if ((0xFFFF - envADSR[ich]) <= ADSRa[ich]) {
393.               envADSR[ich] = 0xFFFF;
394.               iADSR[ich] = 2;
395.            }
396.            else envADSR[ich] += ADSRa[ich];
397.        }
398.        tch[ich]++;
399.        setPWM(); //#7-10
400.    }
401.
402.
403.    //Código para executar a mudança de valor no tom
404.
405.    if (Gteste < 10) {
406.        Gteste += 1;
407.    } else {
408.        Gteste = 0;
409.
410.      if (Gteste2 < 800) {
411.        Gteste2 += 10;
412.      } else {
413.        Gteste2 = 0;
414.      }
415.
416.    }
417.
```

```
418.
419.    Gteste2 = analogRead(A0);
420.
421.
422.
423.    //update the tone for channel 0
424.    amp[0] = (amp_base[0] * (envADSR[0] >> 8)) >> 8;
425.    inc[0] =  Gteste2 + inc_base[0];
426.    FMamp[0] = FMa0[0] + ((long)FMda[0] * FMexp[0]>>16);
427.    FMinc[0] = FMinc_base[0];
428.    setPWM(); //#11
429.
430.    //update the tone for channel 1
431.    amp[1] = (amp_base[1] * (envADSR[1] >> 8)) >> 8;
432.    inc[1] = inc_base[1];
433.    FMamp[1] = FMa0[1] + ((long)FMda[1] * FMexp[1]>>16);
434.    FMinc[1] = FMinc_base[1];
435.    setPWM(); //#12
436.
437.    //update the tone for channel 2
438.    amp[2] = (amp_base[2] * (envADSR[2] >> 8)) >> 8;
439.    inc[2] = inc_base[2];
440.    FMamp[2] = FMa0[2] + ((long)FMda[2] * FMexp[2]>>16);
441.    FMinc[2] = FMinc_base[2];
442.    setPWM(); //#13
443.
444.    //update the tone for channel 3
445.    amp[3] = (amp_base[3] * (envADSR[3] >> 8)) >> 8;
446.    inc[3] = inc_base[3];
447.    FMamp[3] = FMa0[3] + ((long)FMda[3] * FMexp[3]>>16);
448.    FMinc[3] = FMinc_base[3];
449.    setPWM(); //#14
450.
451.    //update counters
452.    tch[0]++;
453.    tch[1]++;
454.    tch[2]++;
455.    tch[3]++;
456.
457.    setPWM(); //#15
458.
459.    }
```