UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA

FILIPE SOARES DONATO

TRABALHO DE CONCLUSÃO DE CURSO

Utilidades e Aplicações da Linguagem Processing

Campina Grande, Paraíba.
Novembro de 2023

FILIPE SOARES DONATO

Utilidades e Aplicações da Linguagem Processing

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do título de Bacharel em Engenharia Elétrica.

Área de Concentração: Controle e Automação; Eletrônica.

Orientador: Jaidilson Jó da Silva

Campina Grande, Paraíba.
Novembro de 2023

FILIPE SOARES DONATO

Utilidades e Aplicações da Linguagem Processing

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do título de Bacharel em Engenharia Elétrica.

Trabalho Aprovado em: 08 / 08 / 2023

Orientador: Jaidilson Jó da Silva

Convidado: Kyller Costa Gorgônio

Campina Grande, Paraíba.
Novembro de 2023

AGRADECIMENTOS

Dedico este trabalho à minha família, em especial aos meus pais, Josivan e Ismailda, que me auxiliaram durante todo o tempo, tendo paciência e compreensão. À minha noiva Janielle que sempre me incentivou e acreditou em mim, estando desde o início do nosso relacionamento disposta a me ouvir e dar apoio.

Ao meu professor orientador, Jaidilson Jó, por aceitar o tema proposto e tirar todas as dúvidas, sempre estando à disposição em sua sala, mostrando um trabalho exemplar na UFCG.

Do ponto de vista religioso não posso esquecer de citar dois nomes: o de Jesus Cristo, que me ajudou a ter fé em alguns momentos, assim como o de Shakyamuni Buda, cujos ensinamentos me auxiliaram a controlar a mente e a me tornar uma pessoa melhor.

Por fim, agradeço aos amigos que fiz ao longo da jornada universitária, com os quais compartilhamos bons momentos e trocas de conhecimento, em especial aos que fazem parte do grupo [OFD].

RESUMO

Embora a linguagem Processing seja inicialmente associada à criação de mídia visual e interativa, ela vem evoluindo e se expandindo para diversas áreas que podem se beneficiar de sua simplicidade e abordagem didática à programação. Uma parte significativa deste Trabalho de Conclusão de Curso concentra-se em demonstrações práticas e tutoriais que ilustram como a linguagem Processing pode ser aplicada na engenharia, na resolução de problemas matemáticos avançados, na criação de aplicações relacionadas a sinais e redes neurais pré-treinadas, bem como na comunicação com microcontroladores e aquisição de dados. Além disso, o trabalho destaca a importância de uma comunidade ativa de desenvolvedores e artistas que contribuem para o ecossistema, promovendo a colaboração e o compartilhamento de conhecimento.

Palavras-chave: Caixas de diálogo, filtragem de sinais, arduino, android, Yolo.

ABSTRACT

Although the Processing language is initially associated with the creation of visual and interactive media, it has been evolving and expanding into various areas that can benefit from its simplicity and didactic approach to programming. A significant part of this Bachelor's Thesis focuses on practical demonstrations and tutorials that illustrate how the Processing language can be applied in engineering, solving advanced mathematical problems, creating applications related to signals and pre-trained neural networks, as well as communicating with microcontrollers and data acquisition. Furthermore, the work highlights the importance of an active community of developers and artists who contribute to the ecosystem, promoting collaboration and knowledge sharing.

Keywords: Dialog boxes, signal filtering, Arduino, Android, Yolo.

Lista de Abreviaturas

YOLO	*You Only Look Once*
GPL	*General Public License*
FLOSS	*Free, Libre, Open Source Software*
IP	*Internet Protocol*
PC	*Personal Computer*
JFET	*Junction Field-Effect Transistor*
MOSFET	*Metal-Oxide-Semiconductor Field-Effect Transistor*
CMOS	*Complementar MOS*
TBJ	*Transistor Bipolar de Junção*
R-CNN	*Region-based Convolutional Neural Networks*
CUDA	*Compute Unified Device Architecture*
FFMPEG	*Fast Forward Moving Picture Experts Group*
GUI	*Graphical User Interface*
Ke:tai	*Korea-England Text And Image*
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit.*

Lista de Símbolos

Ip	Corrente de pico
Vp	Tensão de pico
Idss	Corrente de dreno em estado de saturação
In	Corrente da Harmônica n
Vds	Tensão dreno-fonte
Vgs	Tensão gate-fonte
Vx	Tensão threshold
ϕ	Ângulo
t	Tempo em segundos
K	Constante de Boltzmann
T	Temperatura
q	Carga do Elétron
Gm	Transcondutância grandes sinais
gm	Transcondutância pequenos sinais
F_{cmin}	Frequência de Corte Mínima
F_c	Frequência de Corte
β	Coeficiente de Velocidade
τ	Constante de tempo RC
Te	Período de amostragem
α	Parâmetro
ms	Milissegundos

Lista de Figuras

Lista de Tabelas

SUMÁRIO

1. Introdução

As linguagens de programação desempenham um papel fundamental no cotidiano, não apenas na computação, mas também na engenharia elétrica, tanto durante o curso como ao longo da vida profissional. Facilitando a resolução de problemas, simplificando cálculos complexos e possibilitando a criação de gráficos 2D e 3D. Estas linguagens estão em constante evolução, adaptando-se às demandas do mercado de trabalho e às necessidades dos usuários.

Durante o curso, o uso de software executável na forma de aplicativos é pouco explorado. Normalmente, os aplicativos são criados dentro de ambientes como o Matlab, onde as linhas de código são escritas, e toda vez que se deseja executá-los novamente, é necessário fazê-lo a partir do próprio programa.

Na pandemia, conheci a linguagem Processing, ao procurar formas de criar uma interface que adquirisse dados de sensores conectados ao Arduino e exibisse no monitor do computador. Sua forma de fácil utilização, gratuita, leve, aliado aos exemplos disponíveis e de uma forma que não fossem apenas números em um janela preta ou serial, me fizeram ser um admirador do projeto em si, trazendo a motivação para realização deste trabalho.

1.1. Objetivos

1.1.1 Objetivo Geral

O objetivo neste trabalho é demonstrar como a linguagem Processing pode ser uma ferramenta valiosa para os engenheiros, oferecendo funcionalidades como bibliotecas matemáticas, comunicação com smartphones, exportação de vídeo e até mesmo detecção facial, tornando mais acessível a criação de software executável e aplicativos com as dependências necessárias para executar em qualquer computador com Windows. Demonstrar as possibilidades que essa linguagem oferece ao aluno de Engenharia Elétrica também é um objetivo neste trabalho.

1.1.2 Objetivos Específicos

Como objetivos específicos podem ser citados:

- Explorar bibliotecas da comunidade;
- Comunicação com Smartphones Android;

- Exportação de vídeo;
- Detecção de Objetos com YOLO;
- Aplicação com filtro 1 Euro;
- Comunicação com microcontrolador Arduino.

1.2 Estrutura do Trabalho

Este trabalho está organizado em 5 capítulos. No primeiro capítulo é feita uma introdução e apresentação dos objetivos do trabalho. Em seguida, é apresentada uma fundamentação da linguagem para compreensão dos assuntos seguintes. O capítulo 3 descreve o desenvolvimento na criação de programas da linguagem, as bibliotecas utilizadas e problemas enfrentados. No capítulo 4, estão os resultados obtidos dos programas executados. Finalizando no capítulo 5, estão as considerações finais sobre o trabalho.

2. Fundamentação Teórica

Neste capítulo, será realizada uma revisão que abordará a história, as pesquisas, e assuntos que se enquadram no tema.

2.1 História

Disponibilizada em agosto de 2001, a Linguagem Processing surgiu com foco nas artes visuais e na alfabetização visual dentro da tecnologia. Desenvolvida por Ben Fry, que é autor de livros sobre a linguagem, em conjunto com o professor cofundador e artista Casey Reas. Em 2012 foi criada a Processing Foundation junto com Dan Shiffman, para apoiar o desenvolvimento e ensinar fundamentos de programação dentro de um contexto visual, mas acabou evoluindo para uma ferramenta de desenvolvimento para profissionais e entusiastas (PROCESSING FOUNDATION, 2021?).

Desde seu surgimento, a linguagem é de código aberto, sendo uma alternativa às ferramentas de software proprietário. De acordo com a página do projeto no Github, a licença em específico é a GPL, ou “General Public License”, tendo como objetivo garantir a liberdade de compartilhar e modificar software livre, assegurando que o software seja livre para todos os seus usuários. Além disso, ao longo dos anos surgiram colaboradores que vem contribuindo com código e construindo novas bibliotecas e consequentemente expandindo as possibilidades da linguagem, com a engenharia elétrica podendo se beneficiar de alguns dos seus recursos.

2.2 Instalação e Compatibilidade

Compatível com diversos sistemas operacionais, a instalação da interface de programação é feita através do website principal(processing.org), tendo a disposição a versão mais atualizada, além da possibilidade de encontrar versões anteriores . Em seguida, é necessário extrair o arquivo compactado para uma pasta e começar a utilizar o programa. Atualmente encontra-se na versão 4.3 para os sistemas: Windows, Mac, Linux e Raspberry Pi OS.

Atendendo a filosofia e a política do FLOSS, que significa "Free, Libre, Open Source Software", ou em português "Grátis, Livre, Software de código Aberto", é uma filosofia que enxerga o software como um meio para pensar e fazer. De acordo com a Processing Foundation: "Acreditamos que aprender a programar não é apenas adquirir um determinado

conjunto de habilidades, mas também desenvolver um processo criativo e exploratório. Acreditamos que o software e as ferramentas para aprendê-lo devem ser acessíveis a todos."

2.3 Usabilidade

A interface tem uma semelhança visual com a do arduíno, que gera uma familiaridade inicial. Sua programação é parecida com JAVA, que compartilha muitas semelhanças com a sintaxe e a semântica. Por exemplo, ambas as linguagens usam estruturas de controle de fluxo semelhantes, como if, for, while, switch, e têm uma abordagem semelhante para declaração de variáveis. Outro exemplo é que tanto o Processing quanto o JAVA são linguagens de programação orientadas a objetos (PROCESSING, 2022?).

As bibliotecas adicionais podem ser obtidas pelo próprio programa. São um benefício da linguagem, na qual os desenvolvedores criam novos recursos a serem utilizados pelos usuários, expandindo suas funcionalidades, disponibilizando seus códigos fonte gratuitamente no Github, sobre a mesma licença de código aberto e incluindo seus próprios exemplos (PROCESSING, 2022?).

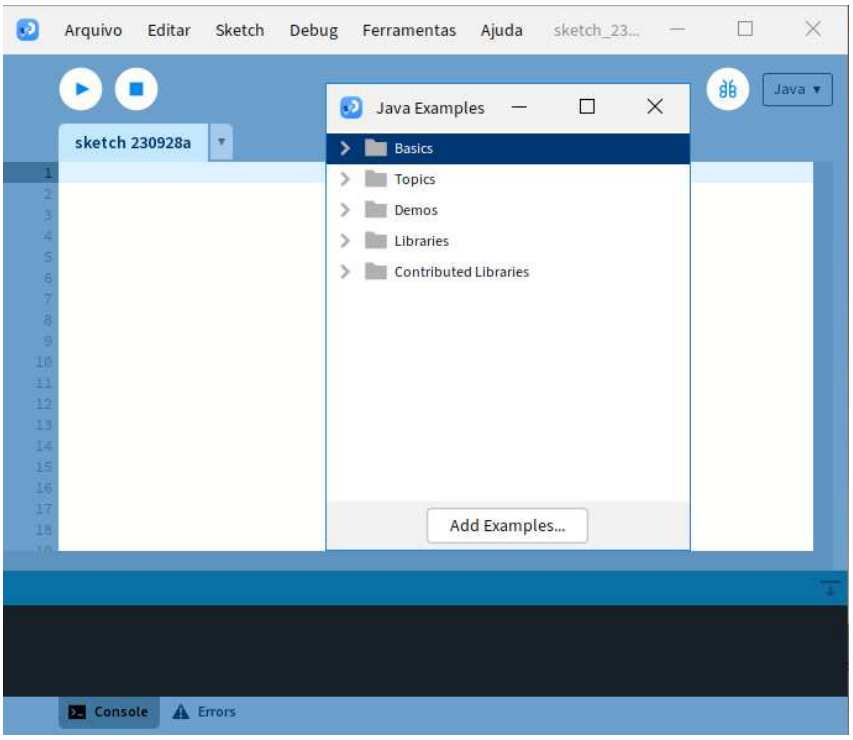
2.3.1 Esboços

A terminologia "sketches" (esboços) é utilizada para se referir aos projetos ou programas criados no ambiente de desenvolvimento do Processing por razões como: simplicidade conceitual, abordagem simplificada, foco na experimentação, intuitividade, abordagem não linear e flexibilidade para tornar a programação mais acessível. Na Figura 1 apresenta-se a tela inicial com uma lista de exemplos básicos à disposição de um usuário iniciante que deseja iniciar os estudos na linguagem.

A interface de desenvolvimento também permite programar em abas, fazendo com que um código grande fique mais organizado e melhor distribuído. O console exibe uma saída de texto para mensagens de erro completas, ou saída de texto indicadas no código pela função *println()* (PROCESSING, 2022?).

O Processing possui três renderizadores de tela integrados. O renderizador padrão *P2D* para desenhar formas bidimensionais. O renderizador *P3D* para geometria tridimensional, câmera, iluminação e materiais. Ambos são compatíveis com OpenGL. Com o lançamento do Processing 4.0, o renderizador *FX2D* está disponível como uma biblioteca separada. Útil para gráficos 2D em telas grandes ou de alta resolução e geralmente é mais rápido que o renderizador padrão (PROCESSING, 2022?).

Figura 1: Tela inicial com exemplos.



Fonte: Autoria Própria.

2.4 Sistema Operacional

No contexto desse trabalho todos os códigos foram executados no sistema operacional Windows 10. O sistema operacional Windows é desenvolvido pela Microsoft, sendo uma das plataformas de sistema operacional mais amplamente utilizadas. Considerando um ranking que inclui os smartphones e dispositivos móveis, atualmente o Windows fica em segundo lugar, com a primeira posição sendo ocupada pelo Android (LISBOA, 2022).

2.4.1 Estrutura Inicial

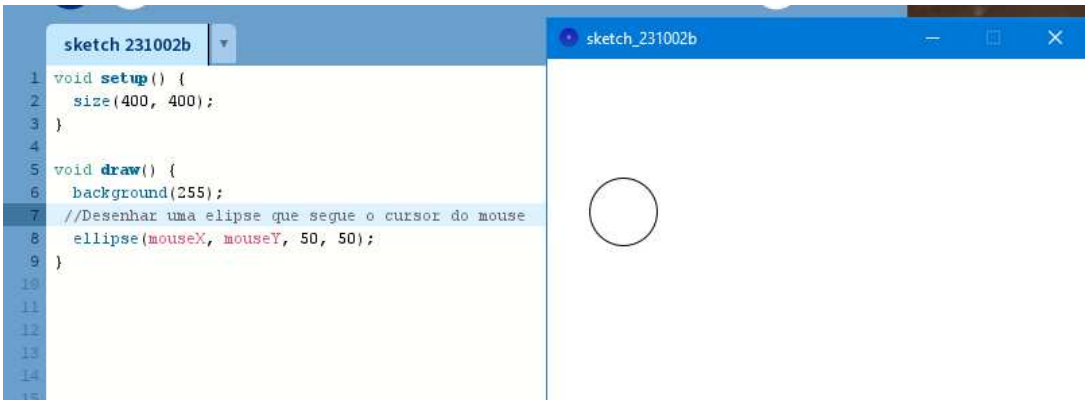
A estrutura inicial de um programa no Processing é simplificada e segue um padrão com as funções *void setup()* e *void draw()*. Com elas já conseguimos criar um primeiro sketch.

A função *setup()* é uma das funções principais, sendo chamada uma única vez quando o programa é iniciado. Geralmente é usada para realizar configurações iniciais, como definir o tamanho da tela de desenho, escolher cores de fundo, configurar variáveis iniciais, inicializar objetos, entre outras tarefas necessárias antes de iniciar o ciclo de desenho principal.

A função *draw()* é executada continuamente após a função *setup()*, criando um loop que atualiza a tela várias vezes por segundo, geralmente a 60fps, podendo ser configurado. Os comandos dentro dela são executados repetidamente até que o programa seja encerrado.

Na Figura 2 apresenta-se o primeiro programa implementado, esse programa cria uma janela de 400x400 pixels, com fundo branco e cria uma elipse que segue o cursor do mouse.

Figura 2: Primeiro programa.



Fonte: Autoria Própria.

2.4.2 Comunicação com Câmera do Smartphone

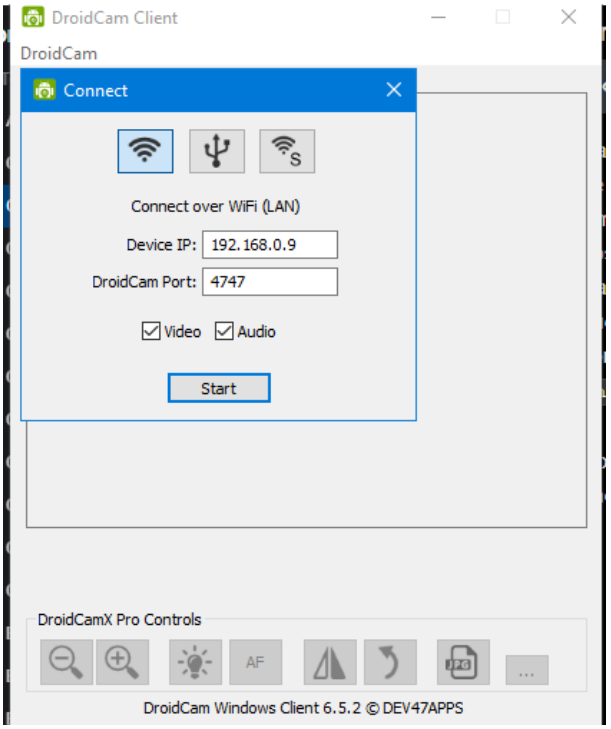
É possível integrar o ambiente Windows com várias ferramentas e aplicativos externos para expandir as funcionalidades do ambiente de desenvolvimento do Processing. Para ter comunicação com o computador e fazer testes, em vez de usar uma webcam tradicional, podemos utilizar a câmera do Smartphone. Um dos programas utilizados foi o DroidCam, que é um aplicativo que permite utilizar a câmera do smartphone Android como uma webcam no computador com Windows.

Entre as funcionalidades disponíveis, existe a interação tanto via USB como WiFi. Para usar o DroidCam com o Processing, deve-se seguir os passos gerais:

1. Instalar o DroidCam: é necessário ter o DroidCam instalado tanto no smartphone Android (Play Store) quanto no computador Windows (www.dev47apps.com).
2. Conectar o Smartphone: pode ser usando um cabo USB (com a depuração USB ativada), ou por uma conexão Wi-Fi (por endereço IP).
3. Iniciar o DroidCam: Abra o aplicativo DroidCam no computador e siga as instruções para iniciar a câmera do smartphone como uma webcam., para futuramente serem criados esboços.

Na figura 3 apresenta-se a imagem inicial do programa sendo executado no computador.

Figura 3: Programa DroidCam.



Fonte: Autoria Própria.

2.5 Protocolo Firmata e Arduino

É possível interagir com um microcontrolador arduino não somente por sua interface de desenvolvimento, mas também pelo protocolo Firmata. O Firmata é um protocolo feito para que através de um software em um computador seja estabelecida uma comunicação com um microcontrolador. Seja um smartphone, PC ou tablet, em diversas arquiteturas (FIRMATA, 2017?).

O Firmata é baseado no formato de mensagem MIDI, no qual os bytes de comando possuem 8 bits e os bytes de dados possuem 7 bits. Atualmente, a implementação mais completa é para o Arduino. Existem outras implementações em várias bibliotecas que implementam o protocolo para se comunicar com uma plataforma, podendo destacar: Processing, Python, Perl, Ruby, Java, PHP, iOS, Haskell, Dart, Elixir, Modelica, Goland, Qt, Kotlin, LabVIEW, .NET, entre outras (FIRMATA PROTOCOL, 2017?).

.

Existem dois principais modelos de uso do Firmata. No primeiro método, pode-se usar um Sketch do Arduino com os vários métodos fornecidos pela biblioteca para enviar e receber dados entre o dispositivo Arduino e o software em execução no computador.

O segundo e mais comum método é carregar um sketch de propósito geral como os códigos já prontos intitulados “SimpleDigitalFirmata”, “SimpleAnalogFirmata” ou o “StandardFirmata”. Também existem variantes, como “StandardFirmataPlus” ou “OldStandardFirmata”, dependendo das necessidades, que vai atender a comunicação com a placa Arduino e em seguida, usar exclusivamente o computador para interagir com os sensores e atuadores (FIRMATA, 2017?).

Dentro dos arquivos da biblioteca existem exemplos disponíveis para que o usuário entenda a dinâmica de escrita e funcionamento do código. Por ser de código aberto, é possível contribuir no Github reportando problemas ou bugs, abrindo um novo tópico com um problema ou sugerindo novos recursos. Mas devido à memória limitada das placas Arduino padrão, os desenvolvedores escolheram não modificar o “StandardFirmata”, sendo possível adicionar novos recursos nas outras implementações disponíveis (FIRMATA, 2017?).

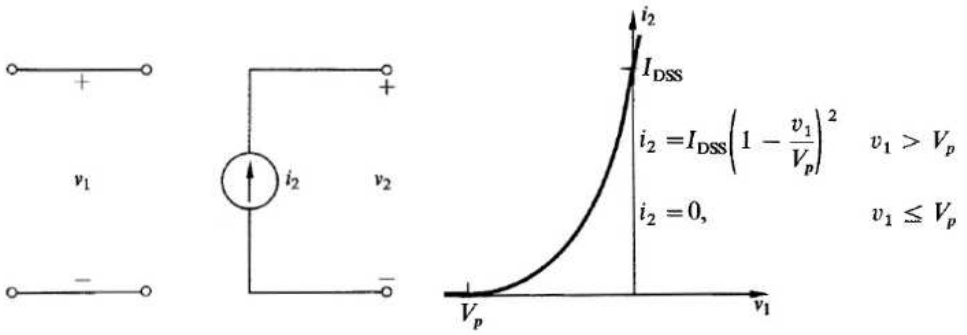
2.6 Característica Quadrática

Um assunto abordado em disciplinas como Circuitos Para Comunicação são os dispositivos eletrônicos que possuem a característica de operação quadrática como os transistores de efeito de campo, JFET e MOSFET. Na Figura 4 tem-se um diagrama de sistemas em tempo real que descreve o comportamento de um dispositivo não linear de lei quadrática com as respectivas equações.

O estudo enfatizou especificamente em analisar quando um sinal estiver além da região quadrática, como está demonstrado na Figura 5, onde não vamos ter um sinal puro na saída, mas sim um sinal amplificado e com harmônicas na corrente, formada por uma sequência periódica de ondas, com amplitude de pico dado por:

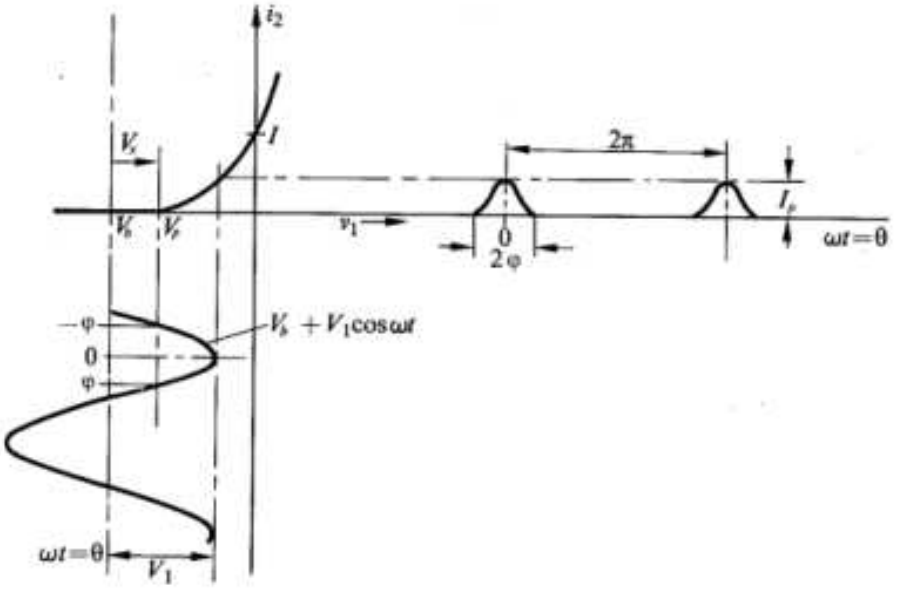
$$Ip = \frac{Idss}{Vp^2}(V1 - Vx)^2 \qquad (1)$$

Figura 4: Sistema com característica quadrática.



Fonte: CLARKE and HESS, 1971.

Figura 5: Sinal parcialmente na região Quadrática.



Fonte: CLARKE and HESS, 1971.

Para encontrar os valores das componentes harmônicas, temos a opção de utilizar um gráfico normalizado (fourier) para os 3 primeiros coeficientes em termos do ângulo de condução 2Φ que o próprio autor disponibiliza na Figura 6, onde os 3 primeiros termos são dados por:

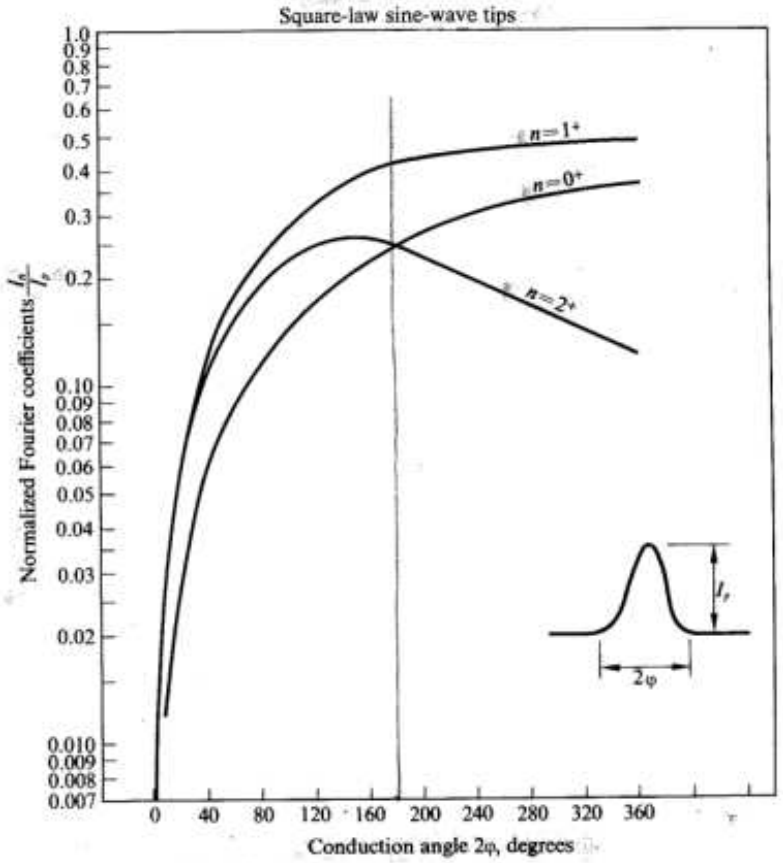
$$I0 = \left(\frac{Ip}{\pi}\right) * \left(\frac{\Phi - \left(3*\frac{sin(2*\Phi)}{4}\right) + \left(\Phi*\frac{cos(2*\Phi)}{2}\right)}{(1-cos(\Phi))^2}\right) \quad (2)$$

$$I1 = \left(\frac{2*Ip}{\pi}\right) * \left(\frac{\left(3*\frac{sin(\Phi)}{4}\right) + \left(\frac{sin(3*\Phi)}{12}\right) - (\Phi*cos(\Phi))}{(1-cos(\Phi))^2}\right) \quad (3)$$

$$I2 = \left(\frac{2*Ip}{\pi}\right) * \left(\frac{\left(\frac{\Phi}{4}\right) - \left(\frac{sin(2*\Phi)}{6}\right) + \left(\frac{sin(4*\Phi)}{48}\right)}{(1-cos(\Phi))^2}\right) \quad (4)$$

Ou podemos construir um código que também faça isso para para os 3 primeiros coeficientes, com as mesmas equações 2, 3 e 4 com a adição de utilizar qualquer valor de ângulo com maior precisão, que será visto posteriormente.

Figura 6: Coeficientes normalizados de Fourier vs ângulo de condução.

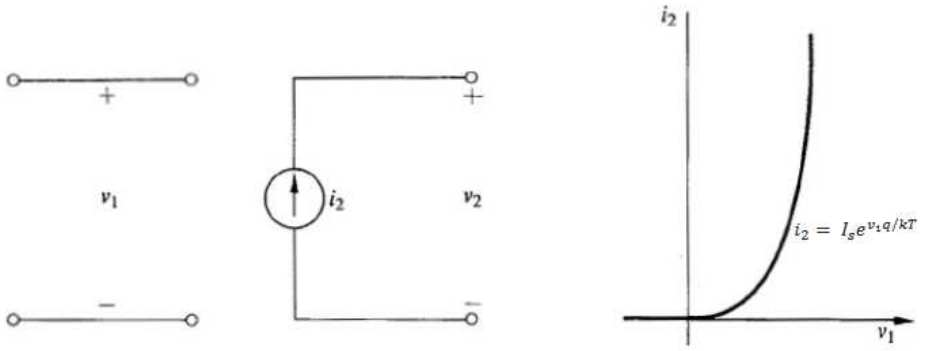


Fonte: CLARKE and HESS, 1971.

2.7 Característica Exponencial

O transistor bipolar de junção(TBJ) é um dispositivo eletrônico que possui a característica de operação exponencial. Na Figura 7 tem-se um sistema que descreve o comportamento de um dispositivo de lei exponencial com as respectivas equações, onde KT/q é aproximadamente 26mV para T=300K.

Figura 7: Característica exponencial.



Fonte: CLARKE and HESS, 1971.

Quando um dispositivo que possui essa característica é seguido por um filtro passa faixa de grande seletividade, é conveniente definir a transcondutância fundamental para grandes sinais $Gm(x)$, que é uma função da amplitude do sinal de entrada dada por:

$$Gm(x) = \frac{q*Idc}{k*T} * \frac{2*I1(x)}{x*I0(x)} \quad (5)$$

Que pode ser reduzida para:

$$Gm(x) = gm * \frac{2*I1(x)}{x*I0(x)} \quad (6)$$

Na Figura 8 temos uma série de valores em função de *x* já tabelados.

Figura 8: valores tabelados.

x	$\frac{2I_1(x)}{xI_0(x)} = \frac{G_m(x)}{g_m}$
0,0	1,0
0,2	0,995
0,5	0,970
1,0	0,893
2,0	0,698
3,0	0,540
4,0	0,432
5,0	0,357
6,0	0,304
7,0	0,264
8,0	0,234
9,0	0,210
10,0	0,190
15,0	0,129
20,0	0,0975

Fonte: CLARKE and HESS, 1971.

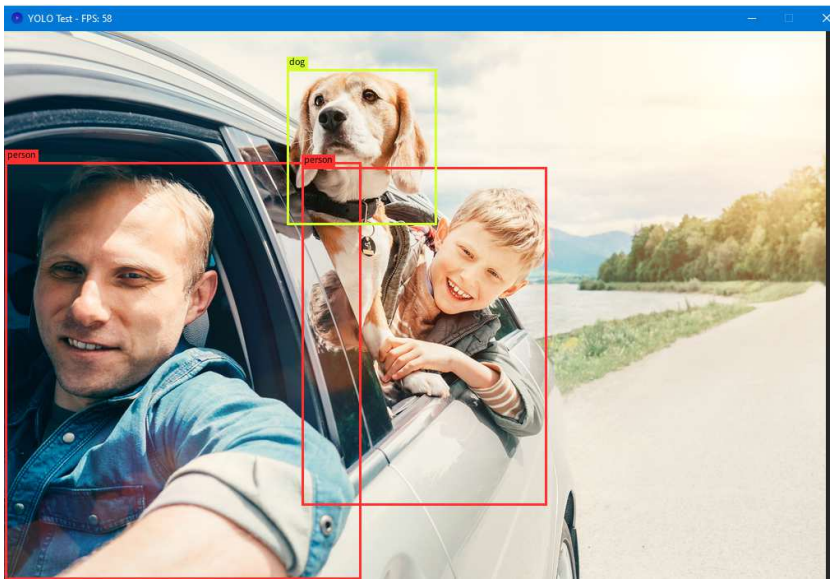
2.8 Rede YOLO

O YOLO é uma abreviação para “You Only Look Once", ou em português “Você só olha uma vez”. Foi lançado em 2015, por Joseph Redmon e Ali Farhadi, sendo reconhecido como uma técnica nova na detecção de objetos, capaz de obter uma precisão superior a de outros métodos de detecção da época (ALVES, 2020).

Em vez de percorrer a imagem em busca dos objetos treinados, o YOLO apenas precisa olhar pela imagem uma única vez e enviá-la para aplicar a uma rede neural. Essa rede divide a imagem em regiões e prevê caixas delimitadoras e probabilidades para cada região. Essas caixas delimitadoras são ponderadas pelas probabilidades previstas. De acordo com os desenvolvedores, uma das vantagens é que como só precisa de uma única avaliação, isso o torna muito mais rápido que sistemas como o R-CNN(Region-based Convolutional Neural Networks) que exigem milhares de avaliações para uma única imagem, tornando o YOLO 1000 vezes mais rápido (REDMON, 2018).

Na Figura 9 apresenta-se a utilização de um programa no Processing que utiliza o YOLO Versão 5 para indicar e classificar os objetos detectados, onde temos duas pessoas e um cachorro.

Figura 9: Localização e classificação com YOLO V5.



Fonte: criminaldefencelawyers.[1]

[1] Disponível em: <https://www.criminaldefencelawyers.com.au/blog/driving-with-pets-the-harsh-penalties-you-could-be-facing/> Acesso em 02 nov, 2023.

Existe a possibilidade de utilizar modelos pré-treinados para avaliação de imagens ou vídeos, dispensando o usuário de criar uma rede e realizar seu aprendizado. Esses modelos variam de acordo com a versão do YOLO utilizada, mas geralmente estão abaixo de 500MB, com algumas versões reduzidas abaixo de 50MB para funcionar em dispositivos como Raspberry Pi. Atualmente o projeto encontra-se na versão 8 (JOCHER, 2023).

2.8.1 Classificação, Detecção e Rastreamento

Ao mencionar o método YOLO, surgem algumas definições que precisam ser explicadas e apresentam diferenças em seus conceitos, que são a classificação de imagens, detecção de objetos e rastreamento.

- A classificação de imagens busca apenas prever a classe presente na imagem (ALVES, 2020).
- A detecção de objetos além de prever qual é a classe, precisa identificar também a localização do objeto nessa imagem e retornar o tipo do objeto (carro, placa, cachorro, bicicleta, etc). A técnica pode retornar erros, como confundir um lápis com um pincel (ALVES, 2020).
- O rastreamento de objetos traz um recurso que detecta objetos nos quadros de um vídeo, como também identifica os objetos através dos quadros do vídeo. Por exemplo, ao detectar um veículo ele deve ser rastreado nos quadros subsequentes (VARGAS, 2020).

2.8.2 CUDA

O CUDA foi desenvolvido pela NVidia para atender diferentes objetivos que envolvam tarefas com processamento intensivo, aproveitando todo o processamento da GPU para agilizar cálculos, simulações, aprendizado de máquina, processamento paralelo, análise e renderização em tempo real (FELIPE, 2015).

Como o YOLO possui alguns recursos de computação intensiva, é interessante utilizar uma placa de vídeo com suporte ao CUDA. Neste trabalho foi utilizado uma NVidia GT1030 para a realização das atividades e um processador Intel Core i7 870.

Para utilizar o CUDA é preciso acrescentar as duas linhas de código abaixo. Onde a primeira serve para ativar o recurso e a segunda retorna para o prompt se o recurso foi ativado (BRUGGISSER, 2022?).

$$DeepVision\ vision\ =\ new\ DeepVision(this,\ true);$$

$$println("CUDA: " + deepVision.isCUDABackendEnabled());$$

2.9 Filtro 1 Euro

O Filtro 1 Euro foi criado por Géry Casiez como uma ferramenta para melhorar a qualidade de um sinal ruidoso, mantendo um equilíbrio entre a filtragem e o atraso. Ele usa um filtro passa-baixa de primeira ordem com uma frequência de corte adaptável: em baixas velocidades, um corte baixo estabiliza o sinal reduzindo o jitter, mas conforme a velocidade aumenta, o corte é aumentado para reduzir o atraso (CASIEZ, 2012).

Em seu algoritmo possui apenas dois parâmetros independentes que se relacionam diretamente ao jitter e ao atraso. O autor em seu artigo original afirma que esse filtro pode tornar a filtragem de sinais de entrada melhor e mais simples (CASIEZ, 2012).

Existem dois parâmetros configuráveis no modelo que são: a frequência de corte mínima(Fc_min) e o coeficiente de velocidade(Beta). Ao diminuir FC_min, diminuímos o jitter. Ao aumentar o coeficiente de velocidade (Beta) diminuímos o atraso de velocidade (CASIEZ, 2012). Então temos que:

$$f_c = f_{cmin} + \beta|\widehat{X}_i^{'}| \qquad (7)$$

Onde,

$$\widehat{X}_i = \left(X_i + \frac{\tau}{T_e}\widehat{X}_{i-1}\right)\left(\frac{1}{1+\frac{\tau}{T_e}}\right) \qquad (8)$$

Sendo,

$$Fc = \frac{1}{2\pi RC} \qquad (9)$$

E,

$$\alpha = \frac{1}{1+\frac{\tau}{Te}} \qquad (10)$$

Uma analogia direta pode ser feita com um circuito elétrico, onde em nosso filtro passa baixas de primeira ordem temos um resistor em série com um capacitor. Podendo citar os parâmetros:

- τ - É a constante de tempo R*C;
- T_e - O período de amostragem;
- F_c - É a frequência de corte;
- α - um parâmetro calculado utilizando o período de amostragem Te e uma constante de tempo T (tal), ambos expressos em segundos.

Esse filtro está implementado em linguagens como Python, C, C++, Java, JavaScript, TypeScript, Arduino e Processing (CASIEZ, 2012). O Algoritmo se encontra no apêndice.

2.9.1 Ruído de Perlin

O ruído de Perlin é uma técnica de geração de ruído utilizado em computação gráfica, especialmente em gráficos gerados por computador, animações, efeitos visuais e simulações. Foi criado por Ken Perlin em 1983 e é amplamente utilizado em diversas aplicações como superfícies de objetos, fogo, fumo ou nuvens, uma aparência natural, imitando a aparência aleatória na natureza (SOLUTIONS, 2023).

Entre as principais características do ruído de Perlin podemos destacar: suavidade, controlabilidade, coerência e aleatoriedade controlada. No Processing podemos especificar o ruído de Perlin em uma, duas ou três dimensões, utilizando a função "*noise()"* (ERNESTO, 2005).

2.9.2 Ruído Gaussiano

O ruído Gaussiano possui esse nome porque apresenta uma função de densidade de probabilidade igual a da distribuição gaussiana (normal). No Processing existe a função "*randomGaussian()*" que retorna um float de uma série aleatória de números com média 0 e desvio padrão de 1 (PROCESSING, 2022?).

2.10 FFmpeg

O FFmpeg é uma biblioteca de código aberto amplamente utilizada, que permite manipular uma variedade de formatos de mídia, incluindo áudio, vídeo e imagem. Podendo destacar: codificar, decodificar , transcodificar , transmitir , filtrar e reproduzir. Sendo compatível com Linux, Mac OS, Windows, BSD, Solaris, etc, estando sob GPL versão 2 ou posterior (FFMPEG, 2020?).

3. Metodologia

Este capítulo abordará a metodologia utilizada para o aprendizado e dicas sobre o uso da linguagem Processing. Inicialmente, serão listadas as bibliotecas utilizadas. Em seguida, serão detalhadas as instruções para o funcionamento no ambiente Windows, incluindo informações sobre como importá-las e algumas das principais configurações.

3.1 Lista de bibliotecas utilizadas

- *javax.swing.JOptionPane* - Criação de caixas de diálogo.
- *org.apache.commons.math3* - oferece uma ampla gama de funções e utilitários matemáticos.
- *com.hamoid* - permite trabalhar com captura, reprodução e vídeo em tempo real.
- *ch.bildspur.vision* - oferece recursos avançados de visão computacional, rastreamento de objetos e detecção de características em tempo real.
- *processing.core* - fornece funcionalidades essenciais necessárias para o ambiente.
- *processing.video* - projetada para lidar com captura, webcam, efeitos e reprodução de vídeo.
- *signal.library* - permite explorar recursos como filtros e ruídos.
- *ketai.sensors* - útil para criar aplicativos interativos e projetos que respondem a entradas de sensores em dispositivos móveis.
- *processing.serial* - projetada para facilitar a comunicação serial entre um programa e dispositivos externos, como os microcontroladores.
- *cc.arduino* - usada para criar interfaces de usuário no Processing que interagem com o Arduino.

Para utilizar as funcionalidades, é necessário incluir cada biblioteca no início do código com um comando *import*. Isso deve ser feito junto com o nome da biblioteca, seguido por um asterisco, que representa chamar todos os elementos do pacote especificado. A linha de importação deve ser finalizada com um ponto e vírgula. Por exemplo:

$$import\ processing.video.*;$$

3.1.1 Caixas de Diálogo

Caixas de diálogos são elementos que fornecem uma forma de comunicação entre o software e o usuário, permitindo que o usuário forneça informações, faça escolhas ou

responda a mensagens exibidas pelo programa. Elas são frequentemente usadas para solicitar a entrada do usuário, exibir mensagens de aviso ou fornecer opções de configuração.

No Processing a função *"JOptionPane"* faz parte do pacote *"javax.swing"*, utilizado para criar e exibir caixas de diálogo simples que solicitam entrada do usuário ou exibem mensagens. Ao utilizar o código *"import javax.swing.JOptionPane"* é possível trazer para o código a possibilidade de criação de caixas de diálogo, confirmação, entradas, entre outras.

Em seguida é definida uma função principal que constrói uma caixa de diálogo de entrada de texto, garantindo que o usuário insira algum texto antes de retornar esse valor. Isso pode ser útil para obter informações ou entrada do usuário em um programa com interface gráfica, como apresentado na Figura 10.

Figura 10: Trecho de código para caixas de diálogo.

```
1  import javax.swing.JOptionPane;
2
3  String showInputDialog(String message) {
4    String input = "";
5    while (input.equals("")) {
6      input = JOptionPane.showInputDialog(null, message);
7    }
8    return input;
9  }
```

Fonte: Autoria própria.

Logo, toda vez que no código for chamado `showInputDialog`, irá aparecer uma caixa de diálogo para ser inserido um valor.

Figura 11: Código simples para exemplificar Caixas de Diálogo.

Fonte: Autoria própria.

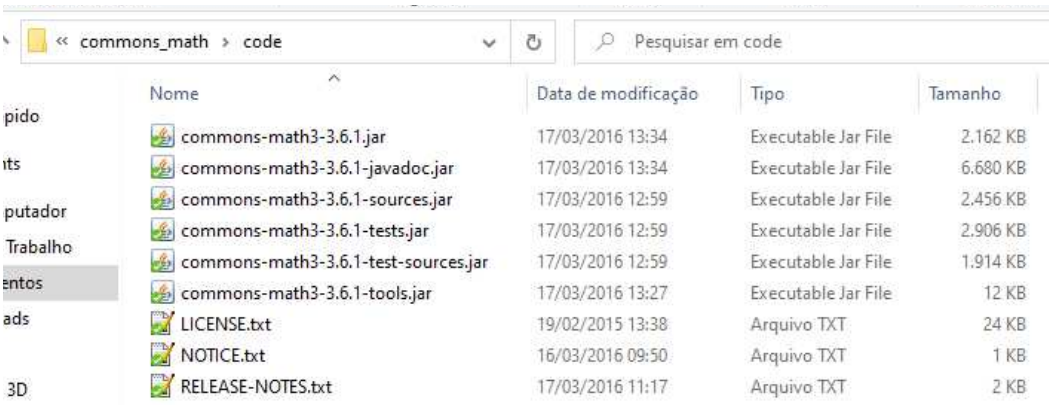
Por exemplo, na Figura 11, temos um programa que: importa a biblioteca *javax.swing.JOptionPane*, em seguida antes do *setup()* cria a *string ShowInputDialog* e declara uma variável *float X*, posteriormente no *setup()* define o tamanho da janela, o plano de fundo como branco e pede para ser inserido o valor de X. Por fim, em *draw()* é definido o tamanho do texto, a cor da fonte e é exibido o que foi digitado em uma posição (X,Y) que no Processing é iniciada (0,0) no topo superior esquerdo. Uma outra implementação mais completa deste recurso estará na seção 4.1.1, mas também será utilizado em outros trechos de código.

3.1.2 Apache Commons Math

O Apache Commons Math Project é uma biblioteca de componentes de matemática e estatística que abordam os problemas práticos mais comuns que não estão imediatamente disponíveis dentro da linguagem, possuindo mais de 80 pacotes diferentes que podem ser utilizados em um código. É útil para procedimentos de análise numérica comuns, como localização de raízes, interpolação, derivação e integração da função (APACHE, 2022).

Ele não está no gerenciador de bibliotecas da interface do Processing. Para obtê-lo é necessário acessar o website "*dlcdn.apache.org//commons/math/binaries/*" e baixar o arquivo zipado "*commons-math3-3.6.1-bin.zip*". Em seguida, devemos criar um sketch e descompactar os arquivos dentro de uma pasta com o nome "*code*", então quando chamamos a biblioteca no código ele irá procurar os arquivos dentro dessa pasta, como apresenta-se na Figura 12. Na data de publicação deste trabalho já existe a versão 4.0 beta.

Figura 12: Arquivos descompactados para utilização das funcionalidades.



Fonte: Autoria própria.

Para utilizar a biblioteca importamos no código o pacote no qual possuirá as funções que desejamos utilizar, ou podemos importar todas as funcionalidade disponíveis com a linha:

$$import\ org.apache.commons.math3.*;$$

Entre os recursos disponíveis, existe o pacote:

$$org.apache.commons.math3.analysis.function$$

Esse pacote disponibiliza o uso de diversas funções como: seno, cosseno, tangente, funções hiperbólicas, logaritmo natural, logaritmo base 10, raiz, inversa, exponencial, gaussianas, sinc, entre outras.

Existe também um pacote que possui funcionalidades para trabalhar com funções de uma ou mais variáveis, integrais ou derivadas, chamado:

$$org.apache.commons.math3.analysis.integration$$

Que possibilita calcular integrais utilizando os métodos:

- Ponto médio(regra do ponto médio)
- Rombergintegrator(algoritmo Romberg)
- SimpsonIntegrator(regra de Simpson)
- Trapezoidintegrator(regra do trapézio)

E caso necessite fazer códigos que utilizem os números complexos, existe o pacote:

$$org.apache.commons.math3.complex$$

Com esses recursos podemos criar programas que resolvam problemas como o que será visto na seção 4.1.2.

3.1.3 Vídeo Export

A biblioteca "*processing.video*" permite criar facilmente arquivos de vídeo diretamente no Processing (PAZOS, 2018). Além de possuir algumas utilidades importantes a destacar:

- Obtém comunicação com uma webcam ou câmera de celular.
- Interage com o FFmpeg e facilita a exportação de arquivos.

Para funcionar na versão 3 do Processing, basta apenas instalar pelo gerenciador de bibliotecas. Já no Processing 4 em diante precisamos fazer alguns passos, listados a seguir:

1. Ir no gerenciador de bibliotecas do Processing e baixar a biblioteca "Video Export";

2. ir no website "*www.gyan.dev/ffmpeg/builds/*", encontrar a opção "*release builds*" e baixar o "ffmpeg-5.1.2-essentials_build.7z";
3. Descompactar o pacote em alguma pasta do computador;
4. Ir no diretório:

   "C:\Users\Seu_Usuario\Documents\Processing\libraries\VideoExport"

5. Editar o arquivo "*settings.json*" onde em sua última linha deve conter a o diretório do ffmpeg que foi descompactado, no meu caso a linha ficou como:

   ffmpeg_path": "C:\\Users\\Windows\\Documents\\PROCESSING PROGRAMAS\\ffmpeg-5.1.2-essentials_build\\bin\\ffmpeg.exe

3.1.4 Deep Visions

A biblioteca Deep Visions possui recursos de visão computacional para fornecer meios simples e fáceis de utilizar algoritmos de aprendizado de máquina para tarefas dentro do ambiente Processing. A biblioteca contém uma base de dados com diversos modelos pré-treinados para serem utilizados no código. Para realizar a instalação, é utilizado o mesmo gerenciador de bibliotecas do Processing. Quando o código é testado pela primeira vez, é preciso aguardar alguns minutos dependendo da velocidade de conexão com a internet, pois os arquivos necessários são baixados e armazenados nas pastas da interface de desenvolvimento.

A lista de redes disponibilizadas é grande (BRUGGISSER, 2022?). Contendo redes das mais leves até as mais exigentes para serem aplicadas em diferentes tipos de hardware. Uma lista completa dos recursos disponíveis está no Github da biblioteca[2], mas podemos destacar:

- YOLOv3-tiny;
- YOLOv3;
- YOLOv4;
- YOLOv4-tiny;
- YOLOv5 (tipos n, s, m, l, x);
- YOLO Fastest.

[2] Disponível em: <https://github.com/cansik/deep-vision-processing>
Acesso em 02 nov, 2023.

Ao iniciar um projeto, as principais linhas que são utilizadas para importar os recursos são:

$$import\ ch.bildspur.vision.*;$$

$$import\ ch.bildspur.vision.network.*;$$

$$import\ ch.bildspur.vision.result.*;$$

A rede primeiro deve ser criada e depois configurada. por isso que em seguida criamos uma instância da classe DeepVision para configurar e executar os modelos de visão computacional:

$$DeepVision\ vision\ =\ new\ DeepVision(this);$$

Depois declaramos uma variável chamada "*yolo*" do tipo "*YOLONetwork*", que é usada para configurar e executar o modelo YOLO que foi detalhado anteriormente.

$$YOLONetwork\ yolo;$$

Também é importante declarar uma variável para armazenar os resultados da detecção de objetos.

$$ResultList < ObjectDetectionResult > \ detections;$$

Se formos utilizar uma imagem, deve-se declarar uma variável do tipo "*PImage*". Lembrando que o arquivo, seja um jpeg ou um png, deve estar dentro de uma pasta de nome "data".

$$PImage\ image;$$

No *setup()* definimos o tamanho da janela, carregamos a imagem, definimos a rede a ser utilizada (se é YOLO versão 4 ou 5 por exemplo) e configuramos o modelo. Uma linha importante é:

$$yolo.setConfidenceThreshold(0.3f);$$

Essa linha define um limite de confiança de detecção, no caso se escolhermos 0.3, significa que apenas as detecções com uma confiança maior que esse valor serão consideradas válidas. Quanto maior o limite de confiança, mais restrita vai ser a detecção.

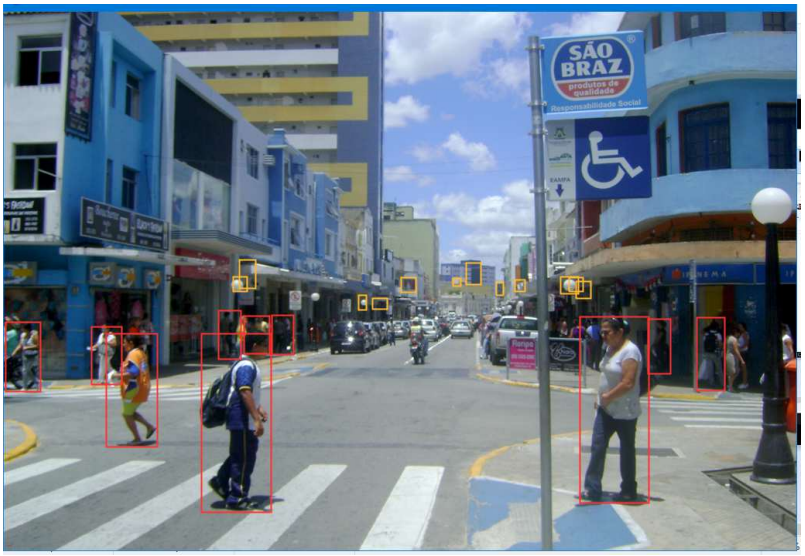
Já na função *draw()* devemos exibir a imagem na tela e desenhar retângulos ao redor dos objetos detectados, utilizando um "for" que cria um retângulo em torno do objeto detectado na imagem. Na Figura 13 apresenta-se o código completo deste primeiro exemplo e na Figura 14 o resultado da detecção na imagem intitulada CAMPINA2.jpg.

Figura 13: Código completo de um exemplo com DeepVisions.

```
import ch.bildspur.vision.*;
import ch.bildspur.vision.result.*;

DeepVision deepVision = new DeepVision(this);
YOLONetwork yolo;
ResultList<ObjectDetectionResult> detections;
PImage image;

void setup() {
  size(1024, 700);
  colorMode(HSB, 360, 100, 100);
  image = loadImage("CAMPINA2.jpg");
  yolo = deepVision.createYOLOv5n();
  yolo.setup();
  yolo.setConfidenceThreshold(0.3f);
  detections = yolo.run(image);
}

void draw() {
  background(55);
  image(image, 0, 0);
  noFill();
  strokeWeight(2f);
  for (ObjectDetectionResult detection : detections) {
    stroke((int)(360.0 / yolo.getLabels().size() * detection.getClassId()), 80, 100);
    rect(detection.getX(), detection.getY(), detection.getWidth(), detection.getHeight());
  }
  surface.setTitle("YOLO Test - FPS: " + Math.round(frameRate));
```

Fonte: Autoria própria.

Figura 14: Resultado do exemplo com DeepVision.



Fonte: flickr.com, 2011.[3]

[3] Disponível em: <https://www.flickr.com/photos/hellohellenn/5566729638/in/photostream/> Acesso em 02 nov, 2023.

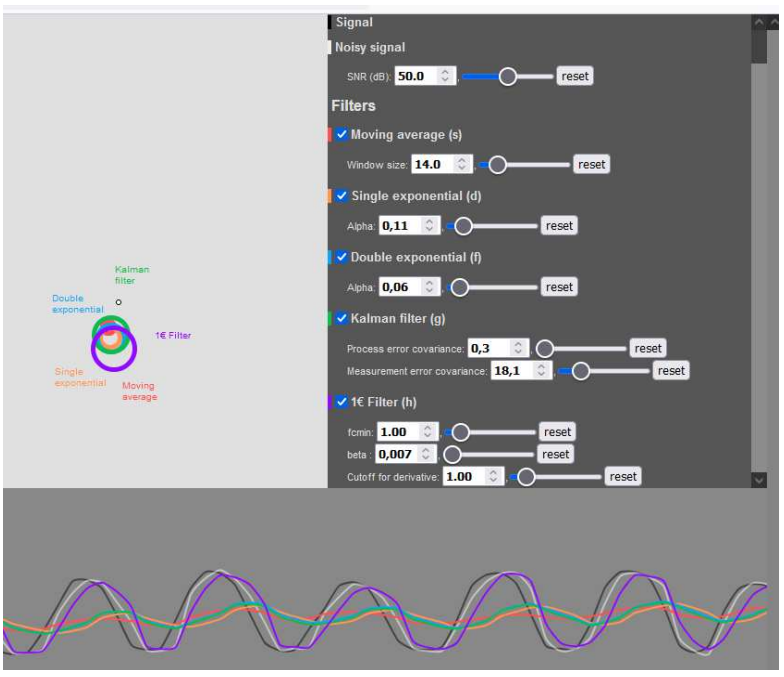
3.1.5 Signal Filter

Ruído é a designação para sinais indesejáveis que aparecem no meio da transmissão, distorcendo os sinais de informações. Podemos também dizer que é a adição espúria ao sinal de informação que tende a alterar seu conteúdo (UFRGS, 1998).

É nesse contexto onde entram os filtros. A utilização do filtro 1 Euro explicada na seção 2.9, criado por Géry Casiez, está disponível na biblioteca "*signal.library*.". Também é encontrada no Github portada para o Processing (COURVILLE, 2013?). Para realizar a instalação utiliza-se o método tradicional de encontrá-la no gerenciador de bibliotecas.

Um comparativo interativo que captura a movimentação do cursor do mouse e aplica ao mesmo tempo diversos filtros é possível ser verificado na própria página do desenvolvedor Géry Casiez. A Figura 15 exibe parâmetros dos 5 filtros disponíveis onde o usuário pode modificar sua calibração, sendo cada filtro representado por uma cor diferente. A princípio vemos que o filtro 1 Euro (em roxo) consegue ter um melhor equilíbrio e rastreamento do cursor em tempo real. Mas ao alterar os valores da calibração dos filtros é possível obter resultados similares entre todos eles.

Figura 15: Comparativo entre filtros.



FONTE: Página de Géry Casiez. [4]

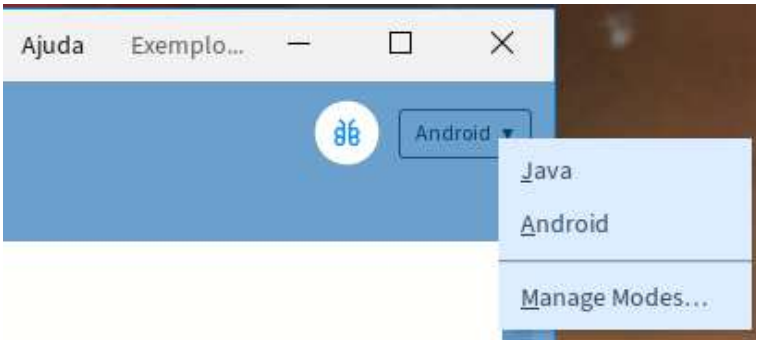
[4] Disponível em: <http://gery.casiez.net/1euro/InteractiveDemo/>
Acesso em 07 out. 2023.

Uma solução para sinais com velocidades distintas é ajustar a frequência de corte, que influencia na largura de banda do filtro, com base nas características do sinal. É nesse ponto onde entra o filtro em foco neste trabalho. Podemos alterar no website dois parâmetros que são a frequência de corte mínima(Fc_min) e o coeficiente de velocidade(Beta) e verificar o comportamento e relação aos demais filtros disponíveis.

3.2 Modo Android

É possível ter um modo Android no Processing. Para isso devemos ir no gerenciador de bibliotecas para instalar o "Android Mode for Processing 4". Em seguida devemos alterar a opção que fica na parte superior direita da interface, como mostra a Figura 16. Em um primeiro momento a aplicação vai instalar o Android SDK automaticamente.

Figura 16: Modo Android.



FONTE: Autoria própria.

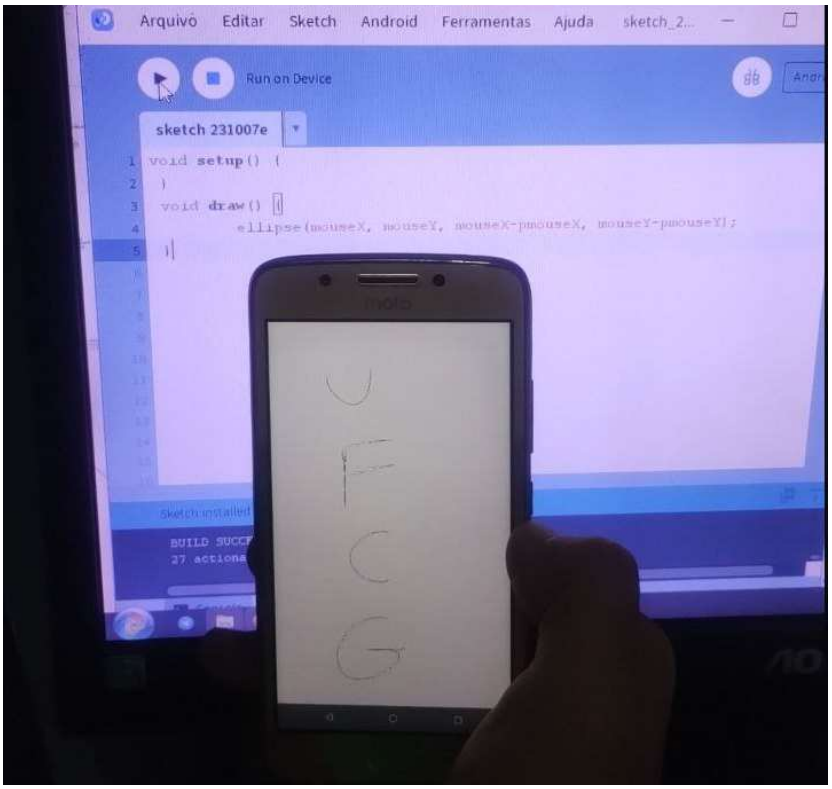
No smartphone deve-se ativar o modo desenvolvedor e a depuração usb para que se possa estabelecer uma comunicação que servirá para que quando um código for executado no Smartphone, o Processing consiga instalar a aplicação no aparelho. Caso não esteja ativado, uma mensagem de erro será exibida.

Em um primeiro momento devemos executar um código simples de duas linhas para saber se a comunicação está funcionando adequadamente. Ao clicar em "*Run*" o sketch será compilado e enviado para o smartphone. Sua função é desenhar elipses com o toque na tela do smartphone, demonstrado na Figura 17.

$$void\ setup()\ \{\ \}$$

$$void\ draw()\ \{ellipse(mouseX,\ mouseY,\ mouseX\ -\ pmouseX,\ mouseY\ -\ pmouseY);\}$$

Figura 17: Código para teste inicial do modo Android.



FONTE: Autoria própria.

3.2.1 Ketai

A biblioteca Ke:tai tem como foco proporcionar acesso aos sensores dos dispositivos Android e facilitar a criação de aplicativos móveis por meio do ambiente de desenvolvimento Processing(KETAI, 2015?). Concebida para se integrar ao ambiente de desenvolvimento, a biblioteca busca simplificar a criação de projetos interativos e visuais, dispondo de recursos e funcionalidades que incluem:

- Acesso a sensores: como acelerômetros, giroscópios e sensores de luz;
- Acesso a câmera e microfone: para captura de imagens e gravação de áudio;
- Suporte a touch: para detecção de toques na tela e gestos de deslizamento.

Um problema que essa biblioteca enfrenta é a ausência de atualizações no projeto. Sua última atualização ocorreu em 2018, época correspondente ao Android 8.0 (Oreo). Na data de publicação deste trabalho, o Android está prestes a lançar sua versão 14. Como mudanças aconteceram até o presente, algumas funcionalidades podem não operar conforme o esperado, devido a problemas de permissões, por exemplo.

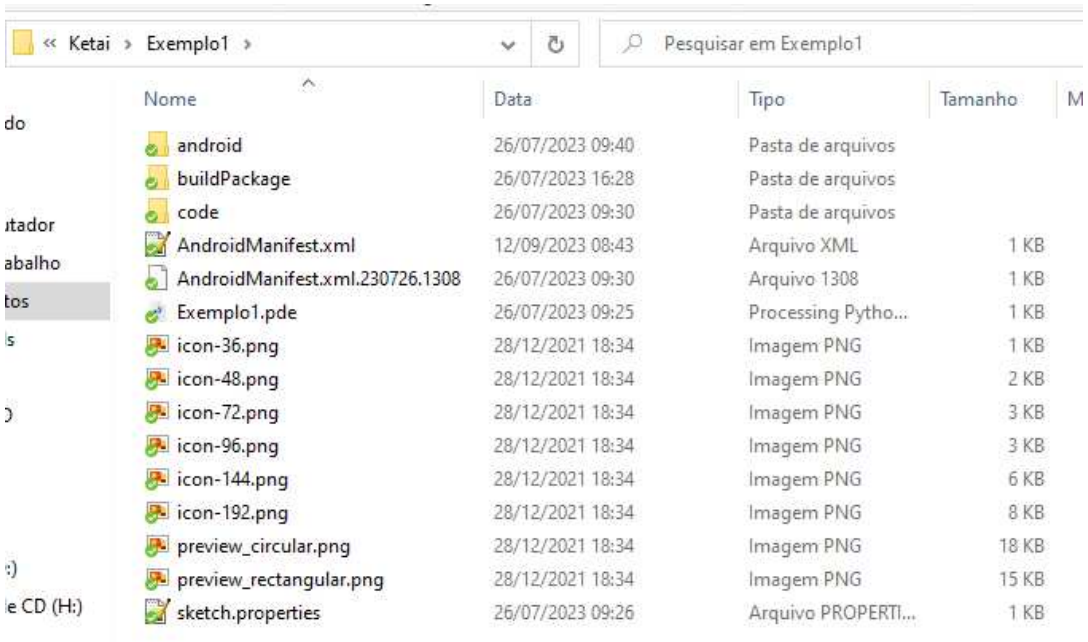
3.2.2 Exportando aplicações

Para exportar as aplicações em um arquivo (apk) para ser instalado em dispositivos Android é preciso seguir algumas exigências:

1. Ir na pasta do Sketch para fazer a alteração no arquivo "AndroidManifest.xml" na linha: package="Nome_do_programa"
2. Na pasta raiz do Sketch precisamos ter também os ícones, que devem seguir os tamanhos: 36x36, 48x48, 72x72, 96x96, 144x144, 192x192 (pixels). E devem ser nomeados como: icon-36.png, icon-48.png, icon-72.png, icon-96.png, icon-144.png e icon-192.png. Pode-se usar um conversor de arquivos para redimensionar a imagem de maior resolução em arquivos com as resoluções indicadas, ou utilizar um editor de imagem.

Com isso vai ser possível criar um arquivo de extensão .apk ido em: Arquivo, Export Signed Package. Em um primeiro momento é necessário definir uma senha que será utilizada toda vez que um arquivo .apk for criado. Na Figura 18 apresenta-se a visualização de como os arquivos devem ficar na pasta raiz.

Figura 18: Arquivos na pasta raiz para aplicações Android.



FONTE: Autoria própria.

4. Resultados

Neste capítulo serão apresentadas aplicações criadas para compreensão da metodologia exposta.

4.1 Limitações Iniciais

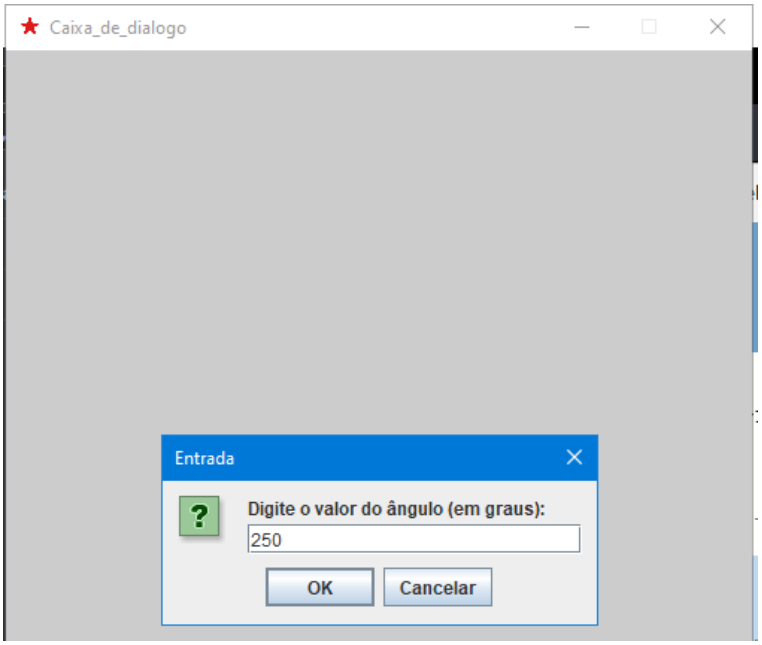
Devido ao fato das bibliotecas adicionais serem mantidas por desenvolvedores livres, em certos casos as atualizações demoram a chegar ou não chegaram ainda ao Processing 4. Tornando o funcionamento limitado a versão 3 do programa. Sendo necessário realizar algumas alterações que serão vistas ao longo do capítulo.

4.1.1 Aplicação das Caixas de Diálogo

Como exemplo, foi escrito um programa para encontrar as harmônicas utilizando o assunto da seção 2.6 que trata sobre característica quadrática para qualquer valor de ângulo e com maior precisão. Para melhorar a interação com o usuário são criadas caixas de diálogo, explicadas na seção 3.1.1, que instruem a inserir os valores por ordem estipulada pelo engenheiro que escreveu o código.

Ao executar a aplicação é requisitado que seja digitado o valor do ângulo em graus. A Figuras 19 exibe como fica uma caixa de diálogo.

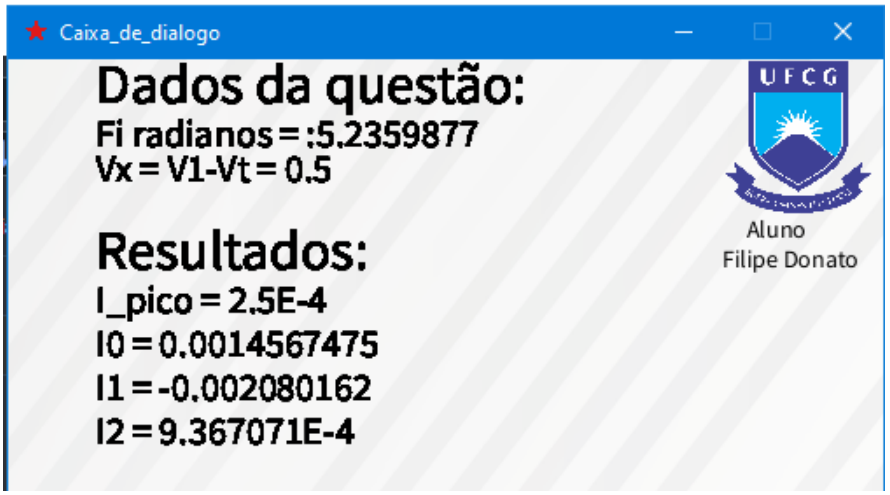
Figura 19: Caixa de diálogo no início da aplicação.



FONTE: Autoria própria.

Internamente no código, são calculados os valores de corrente de pico e as 3 primeiras harmônicas de um sinal que está parcialmente na região quadrática. Também converte o ângulo inserido pelo usuário de graus para radianos, visto que no Processing as funções *sin()* e *cos()* trabalham com valores de entrada em radianos e retornam valores em radianos. Todos os cálculos teóricos podem ser lidos no livro Clarke and Hess, página 98. Na Figura 20 apresenta-se os resultados para um ângulo de 250°.

Figura 20: Resultados obtidos para um ângulo de 250°.

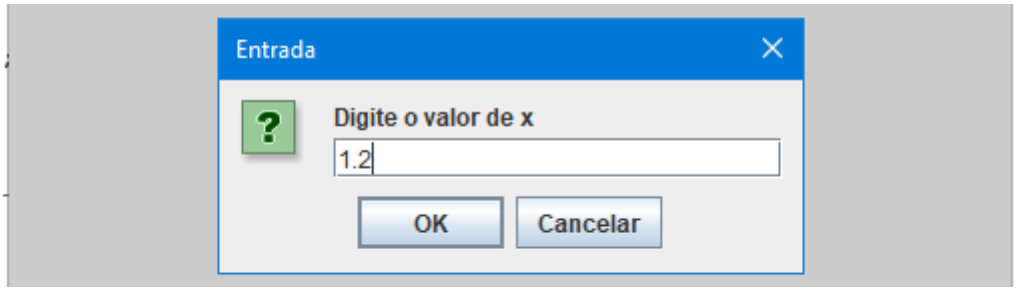


FONTE: Autoria própria[5]

4.1.2 Encontrando Harmônicas com Apache Commons Math

Relativo a seção 3.1.2 foi criado um programa que pode encontrar resultados das harmônicas, para a característica exponencial, para valores de x inseridos pelo usuário, executado em uma interface gráfica que primeiro exibe uma caixa de diálogo, demonstrado na Figura 21.

Figura 21: Entrada de valores.



FONTE: Autoria própria.

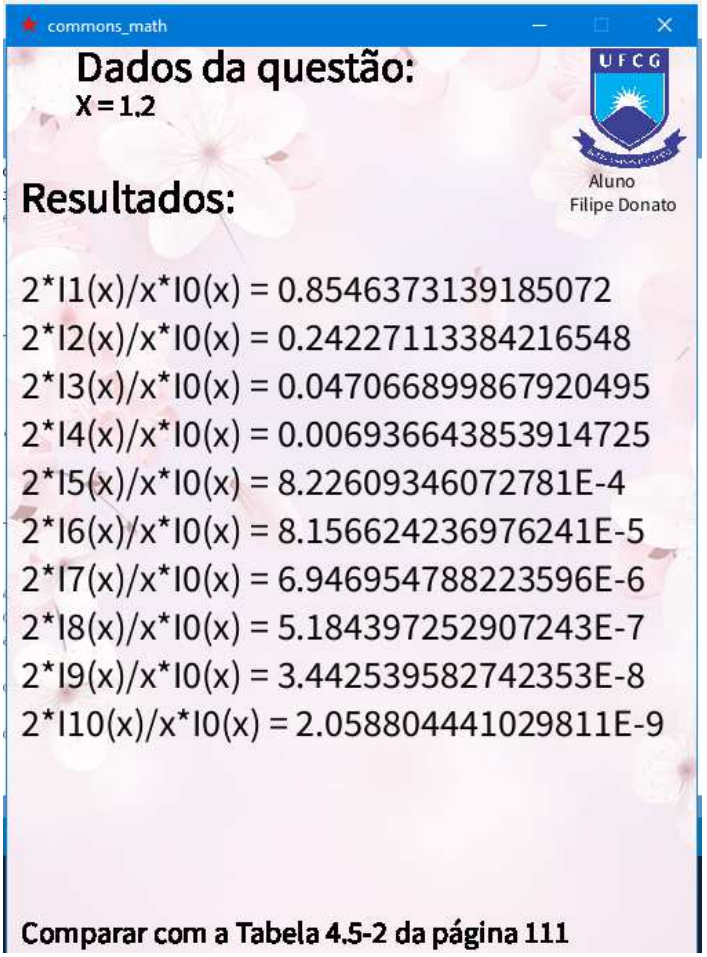
[5] Disponível em: <https://github.com/filipedonato/Processing/tree/main/Caixa_de_dialogo> Acesso em 02 nov, 2023.

Após ser inserido o valor de x, serão exibidos os valores das 10 primeiras harmônicas. Internamente, no *setup()* é criado um laço *for* que calcula valores de n=1 até n=10 harmônicas. A função *draw()* é responsável por exibir os resultados obtidos. Por fim, vem a função do Apache Commons que calcula os valores da integral, utilizando o método de Simpson. Com o valor da integral, por fim, multiplicamos por 1/(2*pi) e o retornamos em uma variável chamada "*result*". A integral calculada é dupla e está no Appendice X do livro Clarke and Hess.

$$In(x) = \frac{1}{2\pi} * \int_{-\pi}^{\pi} e^{x*cos(\theta)} * cos(n\theta)d\theta \qquad (11)$$

Na Figura 22 estão os resultados obtidos para a entrada fornecida X=1,2.

Figura 22: Resultados para a entrada X.



FONTE: Autoria própria[6]

[6] Disponível em:<https://github.com/filipedonato/Processing/tree/main/commons_math> Acesso em 02 nov, 2023.

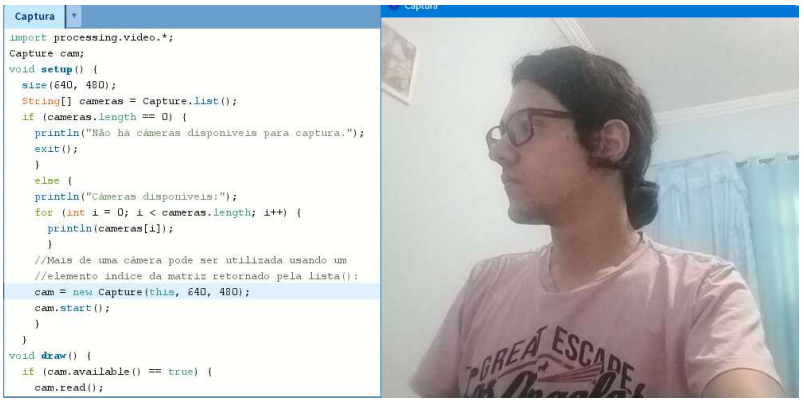
4.1.3 Capturando Imagens da Webcam

Relativo a seção 3.1.3, temos um programa simples, que utiliza a biblioteca "*processing.video*" para capturar imagens da câmera do smartphone, servindo como webcam, com a ajuda do aplicativo DroidCam visto na seção 2.4.2 para facilitar a comunicação.

O código inicialmente importa a biblioteca, em seguida cria um objeto "*cam"*. No *setup()* é definido o tamanho da tela, depois são verificadas as câmera que estão conectadas, com mensagens no console se estão disponíveis ou não, depois inicializa a câmera com a linha "*cam.start()*". Já no *draw()* se a câmera estiver disponível vai exibi-la em uma janela com a linha "*cam.read()*".

A Figura 23 apresenta o resultado da comunicação acontecendo.

Figura 23: Exemplo simples de comunicação com webcam.



FONTE: Autoria própria[7]

4.1.4 Gravando Vídeos

É possível gravar um vídeo só com inúmeras pausas, ativadas por algum botão do teclado ou clique do mouse. Para ilustrar o que gravar, podemos utilizar imagens da webcam ou também criar algum gráfico interativo.

Iniciamos a construção do esboço com "*import processing.video.**" para utilizar os recursos de captura de imagens da webcam, seguido de "*import com.hamoid.**" que traz os recursos necessários para salvar o arquivo no computador. Em seguida criamos os objetos "*videoExport*" e "*cam*". Depois utilizamos uma variável booleana, inicialmente com o estado "*boolean gravando = false*". No *setup()* escolhemos o tamanho da tela, a taxa de frames,

[7] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Video/Captura>
Acesso em 02 nov 2023.

iniciamos a câmera e damos um nome com a localização(pasta data) do vídeo que será exportado:

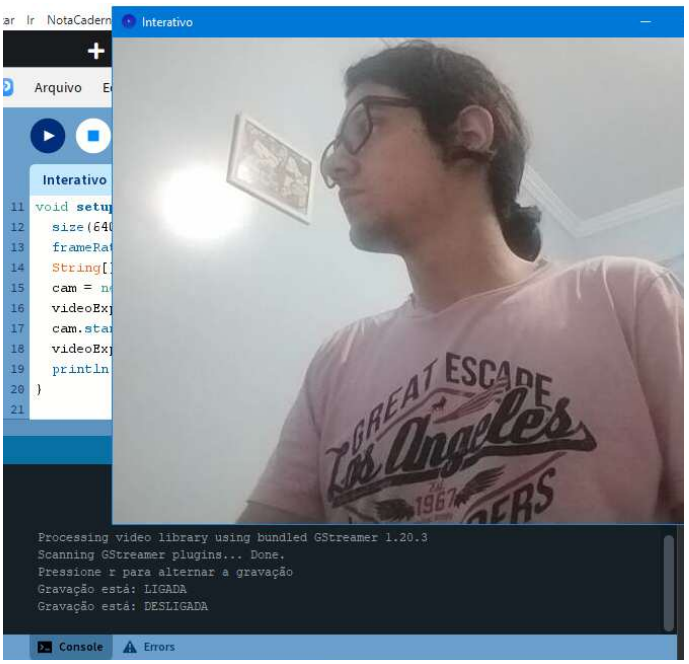
$$videoExport = new\ VideoExport(this, "/data/Interativo.mp4");$$

$$videoExport.startMovie();$$

Em *draw()* criamos uma condição "*if*" para iniciar a exibição da imagem da câmera se a mesma estiver disponível, seguido de um outro "*if*" que vai salvar o que for capturado quando a variável booleana "*gravando*" for verdadeira. Essa condição de verdadeira ou falsa é realizada por meio da função *keyPressed(),* que fica no final do código e é uma função nativa do Processing utilizada para atribuir uma função a uma entrada do teclado. Foi definido que ao apertar a tecla R(minúsculo ou maiúsculo) o estado da variável booleana mudará, funcionando como um play e pausa. Já quando for pressionada a tecla Q, a gravação deve ser encerrada e o programa deve fechar.

Na Figura 24 apresenta-se a exibição do programa sendo executado com as mensagens do console de quando a gravação foi ligada e desligada. Na estrutura de pastas o vídeo ficará na pasta "*data*" e um arquivo "*ffmpeg.txt*" sempre será criado pois é do próprio plugin..

Figura 24: Gravando vídeos com o Processing.



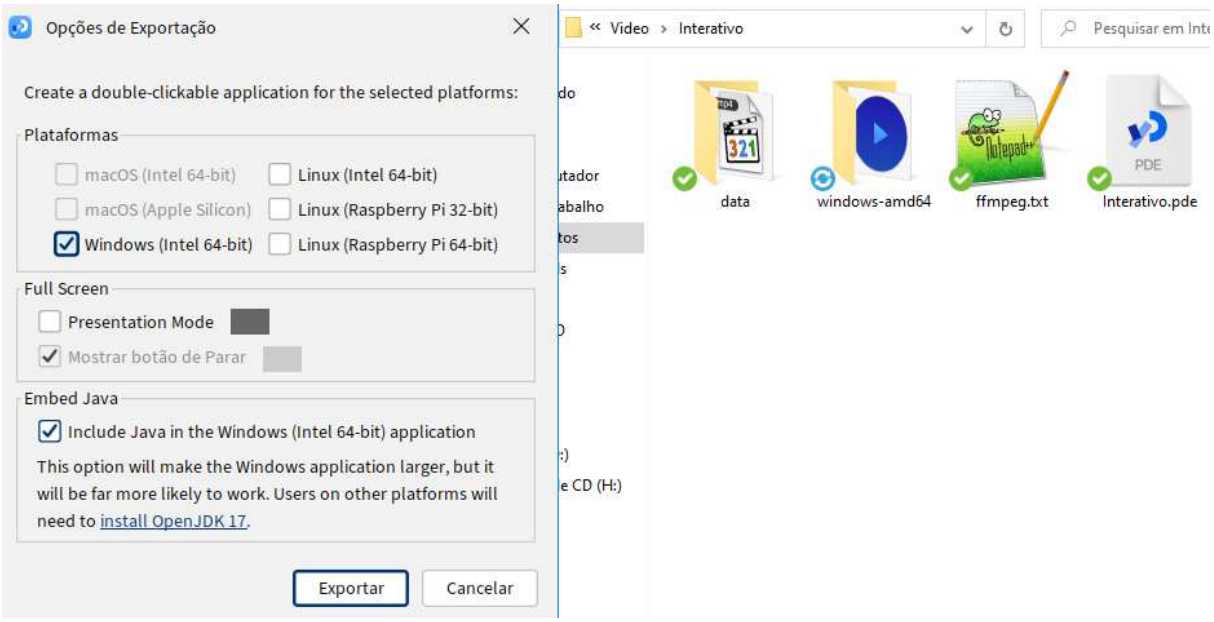
FONTE: Autoria própria[8]

[8] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Video/Interativo> Acesso em 02 nov, 2023.

Após finalizar a construção do código, pode ser preciso exportar a aplicação para fazer uso em outro computador. Para exportar a aplicação basta ir em: Arquivo, Exportar Aplicação, marcar a plataforma desejada(Windows) e marcar para incluir o Java na aplicação.

Depois disso os arquivos ficarão dentro da própria pasta do Sketch em uma subpasta chamada "*windows-amd64*" e para executá-la basta dar 2 cliques no arquivo executável. Na Figura 25 apresenta-se a janela de exportação e como vão ficar as pastas.

Figura 25: Exportando aplicações e estrutura de arquivos.



FONTE: Autoria própria.

Devido a problemas de compatibilidade, ao executar a aplicação fora do ambiente de desenvolvimento, a webcam não foi exibida por um erro do gstreamer, gerando apenas uma tela preta. Para contornar esse erro, é preciso ir na pasta "*windows-arm64*" e depois dentro da pasta "*lib*", criamos a pasta "*windows64*", em seguida copiamos os arquivos de 64bits do gstreamer que ficam localizados na pasta:

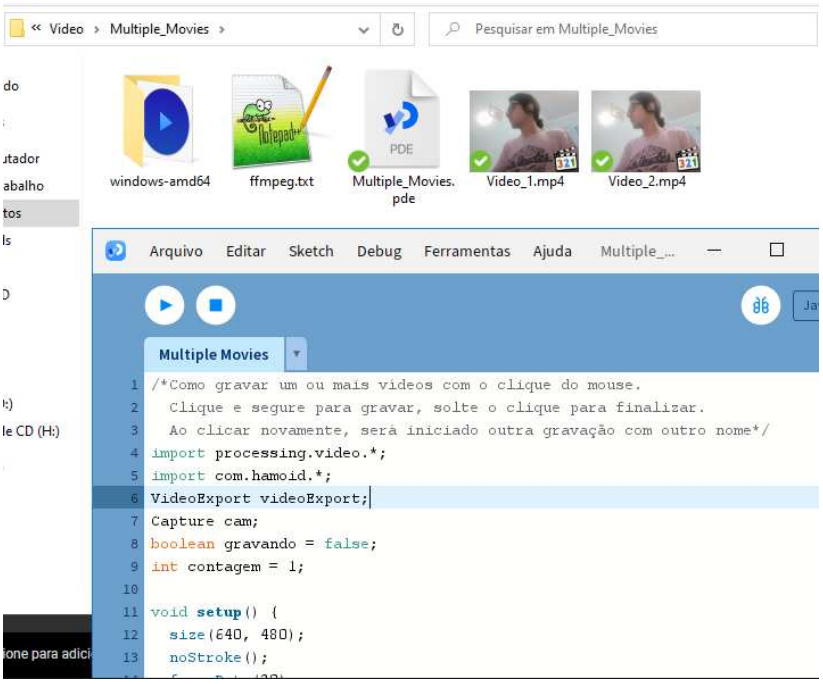
$$C:\backslash Users\backslash Windows\backslash Documents\backslash Processing\backslash libraries\backslash video\backslash library\backslash windows - amd64$$

E então colocamos na pasta que criamos anteriormente. Com isso foi possível exibir a webcam na aplicação exportada fazendo seu correto funcionamento.

Também pode ser implementado o clique do mouse para a gravação, a partir da função *mousePressed()*, onde definimos se queremos iniciar a gravação ao clicar, junto com a função *mouseReleased()* para terminar a gravação ao soltar o clique do mouse. Um exemplo está

apresentado na Figura 26, onde foi adicionado a possibilidade de cada vídeo ser gravado de forma sequencial com nome Video_1, Video_2, Video_3 e assim por diante, sendo que desta vez a localização escolhida dos vídeos foi a pasta raiz. Também foi preciso seguir a dica anterior do erro do gstreamer para que a aplicação funcionasse na sua forma de programa executável.

Figura 26: Gravando vídeos múltiplos com o clique do mouse.



FONTE: Autoria própria[9]

A biblioteca "*com.hamoid*" também possui algumas funcionalidades de ajustes que são citadas a seguir:

- videoExport.setQuality(70, 128) - onde o primeiro parâmetro define a qualidade do vídeo que vai de 0(pior) a 100(melhor mas com arquivo grande), sendo 70 o padrão. E o segundo parâmetro define a qualidade do áudio que pode assumir valores como 64, 128, 192 e 256 kbps.
- videoExport.setFrameRate(25) - Define a taxa de frames no momento da gravação, podendo ter um vídeo exibido a 30fps e gravar a 25 fps por exemplo.
- videoExport.setDebugging(false) - Cria um arquivo de depuração do vídeo.
- ideoExport.forgetFfmpegPath() - Se for alterada a localização da pasta do ffmpeg, use esse comando e ele fará com que a biblioteca peça sua localização novamente.

[9] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Video/Multiple_Movies> Acesso em 02 nov, 2023.

4.1.5 Comparando versões do YOLO no Deep Visions

Neste programa foi estudada a detecção de objetos usando a biblioteca "*ch.bildspur.vision"* em conjunto com o modelo YOLO v5n, v5m e v5x para realizar a detecção de objetos em uma imagem e visualizar os resultados. O objetivo é comparar as três versões, da mais leve (v5n) até a mais exigente (v5x), para exemplificar a diferença que faz o uso do hardware correto na performance de uma aplicação.

Foram utilizadas duas imagens, uma das ruas de campina grande e outra em uma rua aleatória, ambas com resolução próxima a 1024x720p. Foram comparados o tempo de carregamento da CPU vs GPU, que pode ser feito utilizando:

$$long\ startTime\ =\ System.currentTimeMillis();$$

$$long\ endTime\ =\ System.currentTimeMillis();$$

$$long\ processingTime\ =\ endTime\ -\ startTime;$$

Também foi comparado o número de detecções que cada algoritmo fez com o limiar de confiança escolhido em 96% e nenhum limite superior (*setTopK(0)*).

No código a detecção de objetos é realizada com o método "*yolo.run(image)*" e os resultados são armazenados na lista "*detections*". Um loop "*for"* é usado para percorrer cada resultado de detecção na lista detections, em cada detecção um retângulo é desenhado ao redor do objeto, usando as coordenadas e dimensões da detecção. A cor do retângulo é definida com base na classe do objeto detectado. Um retângulo colorido é desenhado acima do retângulo de detecção para exibir o nome da classe do objeto. Uma linha de código vai exibir o número de detecções mostradas com o auxílio de "*detections.size()*".

Na Figura 27 temos um dos métodos sendo utilizados na imagem que contém uma rua de Campina Grande, seguido da Tabela 1 com os resultados obtidos.

Tabela 1: Imagem Campina Grande.

Algoritmo	Tempo CPU	Tempo GPU	Total de detecções
YoloV5n	2948 ms	1360 ms	45
YoloV5m	7688 ms	2511 ms	64
YoloV5x	18792 ms	5451 ms	60

FONTE: Autoria própria.

Figura 27: Centro de Campina Grande.

FONTE: wikimedia[10]

Na tabela 2 temos os resultados obtidos com a segunda imagem, seguido pela Figura 28 onde temos os mesmos métodos sendo utilizados na imagem que contém a foto de uma rua no centro de Fortaleza.

Tabela 2: Imagem centro de Fortaleza.

Algoritmo	Tempo CPU	Tempo GPU	Total de detecções
YoloV5n	2537ms	1322ms	42
YoloV5m	7552ms	2453ms	94
YoloV5x	18595ms	5333ms	87

FONTE: Autoria própria[11]

[10] Disponível em: <https://commons.wikimedia.org/wiki/File:Rua_Marqu%C3%AAs_do_Herval_em_Campina_Grande.jpg>. Acesso em 02 nov, 2023.

[11] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Deep_Vision/YOLOv5> Acesso em 02 nov, 2023.

Figura 28: Centro de Fortaleza.

FONTE: diariodonordeste[12]

Para alternar entre os algoritmos é preciso apenas alterar a linha abaixo com v5n, v5m ou v5x:

$$yolo = deepVision.createYOLOv5n();$$

Foi percebido que nem sempre utilizar um algoritmo mais exigente em termos de recursos computacionais irá resultar em um maior número de detecções, visto que em ambas as imagens o *yolov5m* conseguiu um maior número de detecções com o mesmo limiar de confiança que o *Yolov5x*. Também é preciso analisar o cenário, o hardware e o limiar de confiança. Além disso, é perceptível o impacto que o uso da GPU proporciona nessa aplicação, reduzindo o tempo em mais de 3 vezes se comparado com o processamento sendo realizado apenas por CPU.

[12] Disponível em: <https://diariodonordeste.verdesmares.com.br/metro/vias-do-centro-de-fortaleza-terao-velocidade-reduzida-ciclofaixas-e-outras-mudancas-anuncia-sarto-1.3087636> . Acesso em 07 out. 2023.

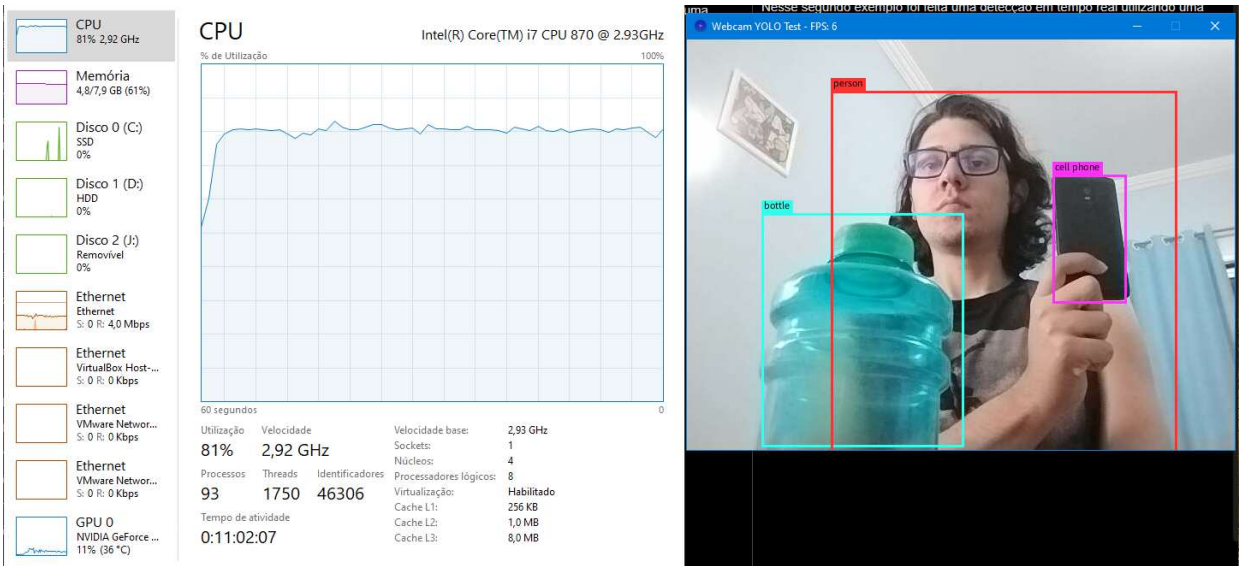
4.1.6 Deep Visions Utilizando Webcam e Gravação

Nesse programa foi feita uma detecção em tempo real utilizando uma câmera de smartphone conectada ao computador via wifi pelo DroidCam. O programa utiliza o modelo *YOLOv4 Tiny* para identificar objetos na cena capturada pela webcam, essa escolha foi feita porque o algoritmo consome menos recursos do que a versão 5. Na tela em tempo real são mostrados retângulos coloridos e rótulos de classe nos objetos detectados pelo modelo escolhido.

Na imagem da Figura 29 é possível ver a deteção de: pessoa, garrafa e celular. No código utilizamos os recursos do próprio “processing.core” e os da biblioteca DeepVision. Ao lado da imagem da webcam está o Gerenciador de Tarefas do Windows exibindo o alto consumo de CPU com a GPU sem conseguir auxiliar, a uma taxa de 6 fps. A GPU não funcionou devido a um aviso de erro no opencv.

Um segundo programa foi criado para juntar as funcionalidades das bibliotecas “processing.video”, “com.hamoid” e a “ch.bildspur.vision”. O objetivo é realizar uma gravação de um vídeo apenas quando for detectado um rosto qualquer.

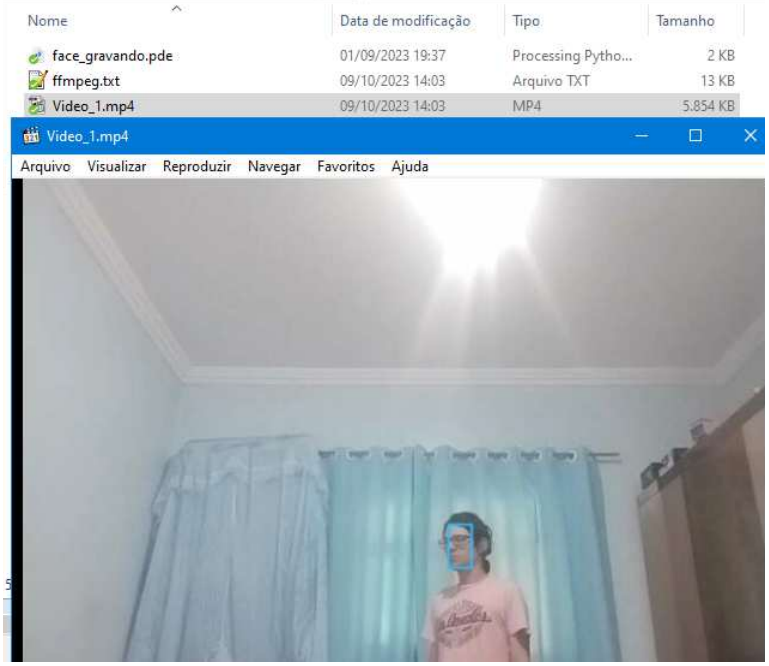
Figura 29: YOLO e Webcam em tempo real.



FONTE: Autoria própria[13]

[13] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Deep_Vision/YOLOWebcamExample> Acesso em 02 nov, 2023.

Figura 30: Gravando apenas quando reconhece um rosto.



FONTE: Autoria própria[14]

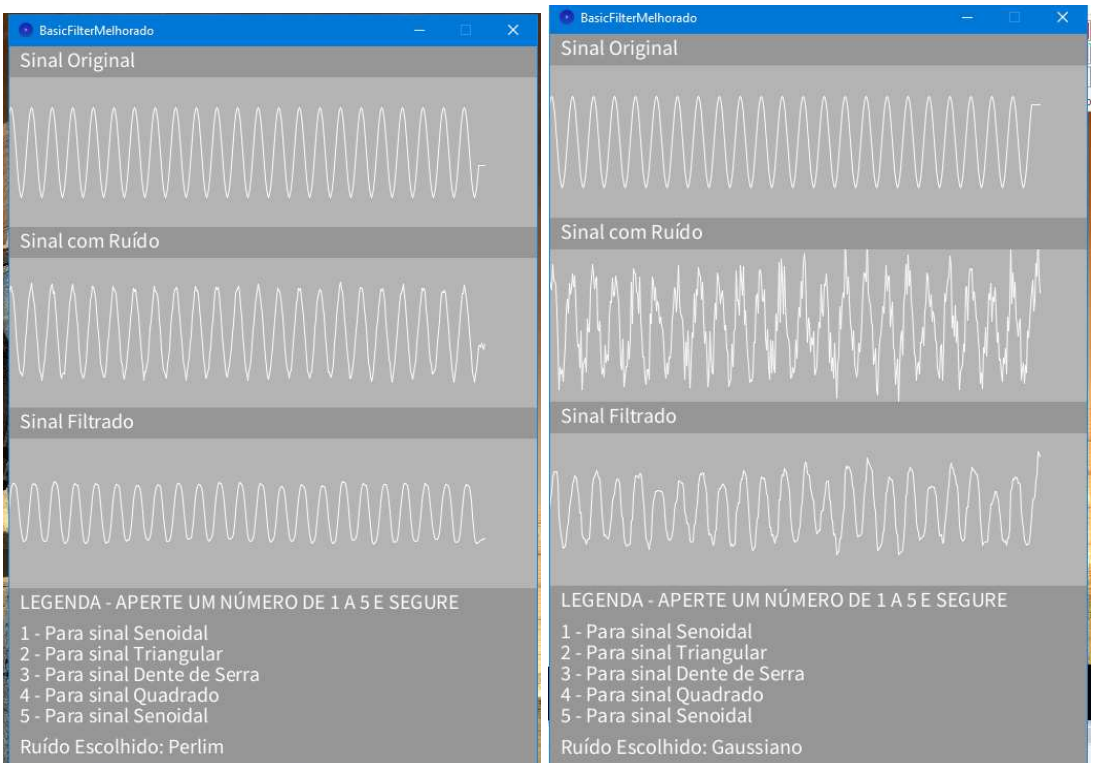
Pode ser útil na criação de um programa que receba imagens de câmeras, com intuito de economizar memória de armazenamento, para gravar apenas nos momentos em que as detecções foram feitas, sejam de pessoas (segurança), ou de objetos, como veículos em cruzamentos de trânsito. A distância da câmera foi de 3 metros, sendo possível ver que o algoritmo consegue identificar um rosto. No código a diferença está na função draw(), que inicia a captura de vídeo da webcam executando a detecção de objetos (faces), desenhando retângulos ao redor das faces detectadas e fazendo a condição de que se uma face foi detectada deve iniciar a gravação, se uma face parar de ser detectada deve-se pausar a gravação até que uma nova face seja encontrada.

4.1.7 Programa Utilizando Signal Filter

A partir da teoria da seção 2.9 e dos exemplos básicos disponíveis, foi implementado um código com funcionalidades para demonstrar tipos de sinais conhecidos como senoidal, triangular, dente de serra e quadrado, adicionados dos ruídos Perlin e Gaussiano, demonstrando a capacidade do filtro 1 Euro. Além disso, também foi utilizado o recursos de caixas de diálogo. Os resultados estão nas Figuras 31, 32 e 33. Alternando entre o sinal escolhido e o tipo de ruído aplicado.

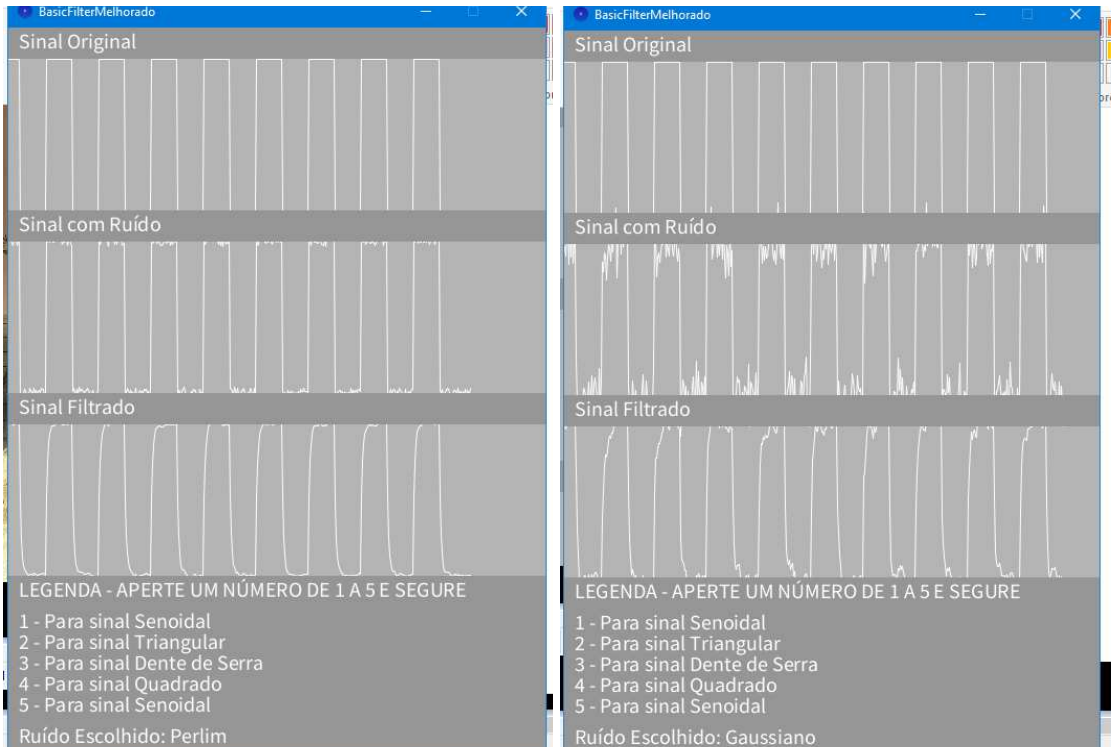
[14] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Video/face_gravando> Acesso em 02 nov, 2023.

Figura 31: Sinal senoidal.



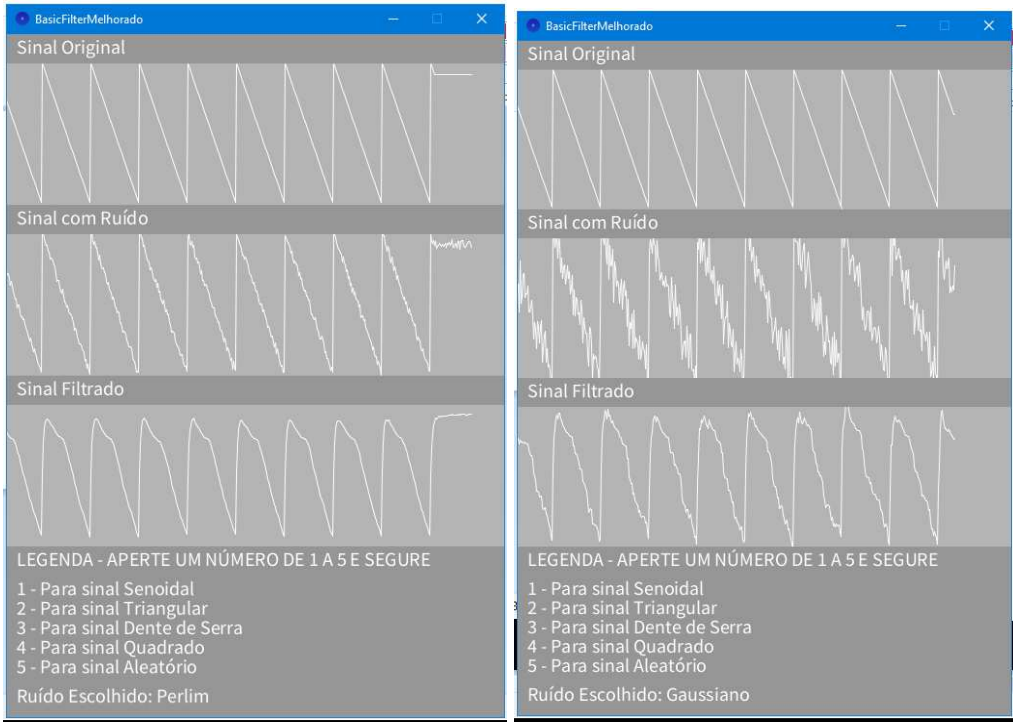
FONTE: Autoria própria.

Figura 32: Sinal quadrado.



FONTE: Autoria própria.

Figura 33: Sinal dente de serra.



FONTE: Autoria própria.

Na Figura 31 temos um comparativo de um sinal senoidal adicionado de um ruído perlim. Depois a aplicação foi aberta novamente mas agora com um ruído gaussiano. O mesmo foi feito na Figura 32 com um sinal quadrado e na Figura 33 com um sinal dente de serra.

O código ficou dividido em 6 partes, sendo cada parte de uma aba da interface do Processing, ilustrada na Figura 34. O código pode se explicado da seguinte forma:

- É Importada a biblioteca "*signal.library"* e a biblioteca "*javax.swing.JOptionPane"*. Em seguida é criado o objeto *meuFiltro* a partir de *SignalFilter.* Depois são definidos os principais parâmetros do filtro 1 Euro que são as variáveis *minCorte* e *beta*. Depois são criadas as variáveis *sinalFonte, sinalFonteAnterior, sinalRuidoso, sinalRuidosoAnterior, sinalFiltrado, sinalFiltradoAnterior*, todas do tipo float, servindo para criar, misturar e posteriormente ver o resultado da filtragem;
- É adicionada a string showInputDialog para utilizar caixas de diálogo;
- Na função *setup()* temos definidos valores como o tamanho de tela, cor de fundo e também é inicializado o filtro. Logo assim que o programa for aberto a variável X precisará de um valor ou opção que deve ser digitado na caixa de diálogo;

- Na função *draw()* passamos os valores do filtro e o necessário para desenhar os gráficos. Temos cinco opções de sinais que são: aleatório, senoidal, dente de serra, quadrado e triangular. Que podem ser escolhidos pressionando e segurando números de 1 a 5 no teclado;
- Também temos duas opções de ruídos a serem escolhidas inicialmente, sendo P para o ruído de Perlin, e caso deseje-se utilizar um ruído gaussiano basta abrir novamente o programa e digitar G;
- No restante do código o objetivo foi desenhar os sinais originais, ruidosos e filtrados, exibir valores no console e limpar a tela quando o sinal preencher toda a tela;
- Nas abas temos o código responsável por cada tipo de onda: aleatório, senoidal, dente de serra, quadrado e triangular.

Figura 34: Utilização de abas na programação.

BasicFilterMelhorado | aleatorio | dente de serra | quadrada | senoidal | triangular

```
1  import signal.library.*; // Adicione a biblioteca ao esboço
2  import javax.swing.JOptionPane;  //Serve para exibir caixas de diálogo
3
4  SignalFilter meuFiltro;   // Criando o Filtro SIGNAL FILTER
5
6  // Principais parâmetros do filtro OneEuro
7  float minCorte = 0.05; // diminuir isso para se livrar do jitter de velocidade lenta
8  float beta = 4.0;  // Aumente isso para se livrar do atraso de alta velocidade
9
10 float xPos = 0;
11 char X;            //Opção a ser Escolhida
12 String X_escolhido;  //Pra exibir na tela o que foi escolhido depois
13 float desvioPadrao = 0.1; // Ajuste o desvio padrão do ruído gaussiano aqui
14
15 float sinalFonte, sinalFonteAnterior;
16 float sinalRuidoso, sinalRuidosoAnterior;
17 float sinalFiltrado, sinalFiltradoAnterior;
18
19 int selectedWaveform = 0; // 0: Senoidal, 1: Triangular, 2: Dente de Serra, 3: Quadrado
20 int selectedNoise = 0; // 0: Ruído de Perlin, 1: Ruído Gaussiano
21
22 //-------------------------------------------------------------
23 String showInputDialog(String message) {
24   String input = "";
25   while (input.equals("")) {
26     input = JOptionPane.showInputDialog(null, message);
27   }
28   return input;
29 }
```

FONTE: Autoria própria[15]

[15] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Signal_Filter/BasicFilterMelhorado> Acesso em 02 nov, 2023.

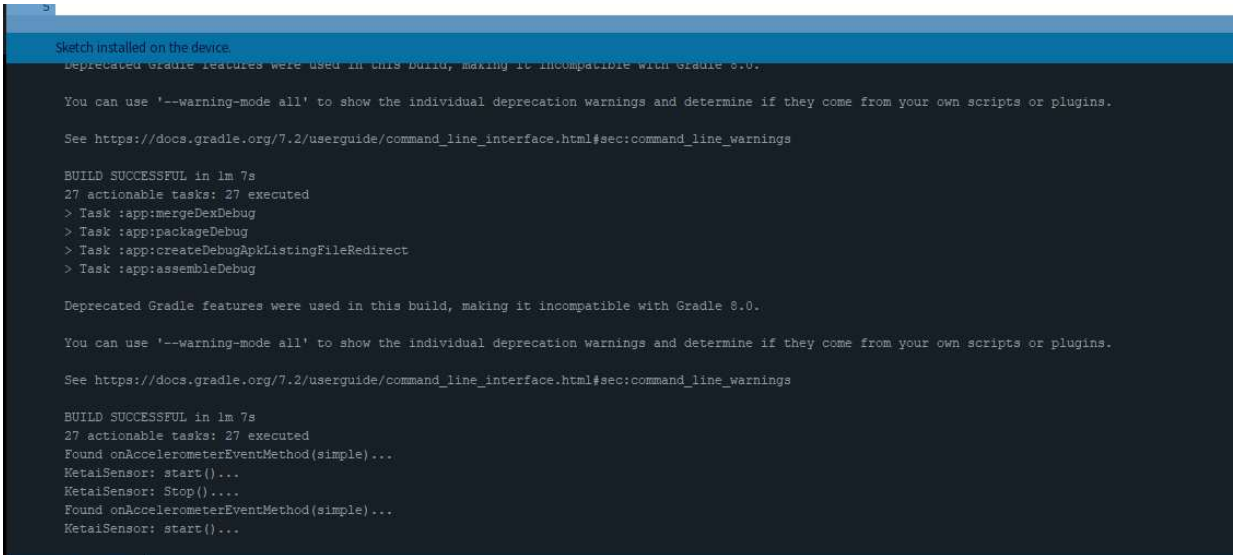
4.1.8 Aplicativo com Ketai para Android

Essa aplicação tem por objetivo mostrar a possibilidade de interação da linguagem com o Android como visto na seção 3.2.1. Iniciamos o código chamando a biblioteca "*Ketai.sensors*", em seguida criamos as variáveis necessárias para os sensores do acelerômetro nas coordenadas X, Y e Z. No *setup()* inicia-se o sensor do acelerômetro e também a leitura do acelerômetro, seguindo com alguns parâmetros como definir a orientação da tela, alinhamento do texto e tamanho da fonte. Em *draw()* exibimos na tela as coordenadas do acelerômetro com as casas decimais que podem ser quantas forem necessárias para a precisão do projeto, formatadas usando a função *nfp()* para serem exibidas centralizadas no meio da tela. Por fim, as demais funções *onAccelerometerEvent(), onLightEvent()* e *onProximityEvent()* ficam no final do código e são chamadas automaticamente sempre que ocorre um evento de leitura seja no acelerômetro, no sensor de luz ou no sensor de proximidade, atualizando as variáveis a serem exibidas na tela.

A função *mousePressed()* que utilizamos anteriormente também pode ser utilizada no modo Android, aqui a função é permitir ao usuário ativar ou desativar a captura de dados dos sensores com um toque na tela.

A Figura 35 apresenta o momento em que o programa é compilado antes de ser enviado para o aparelho. Já na Figura 36 apresenta-se um print da tela do smartphone com a aplicação funcionando.

Figura 35: Momento em que a aplicação é compilada.



FONTE: Autoria própria.

Figura 36: Aplicativo funcionando no smartphone.

FONTE: Autoria própria[16]

Para exportar em uma aplicação de extensão apk que possa ser instalado sem o modo desenvolvedor, devemos seguir os passos da seção 3.2.2.

4.1.9 Interação Processing, Firmata e Arduíno

Antes de mostrar as aplicações, é importante saber em qual porta COM o Arduino está conectado. Para isso, basta ir no Gerenciador de Dispositivos e procurar por "Portas COM e LPT". Outra forma é no próprio código do Processing incluir a linha `println(Arduino.list())` que imprime a lista de portas seriais disponíveis no ambiente de execução do programa.

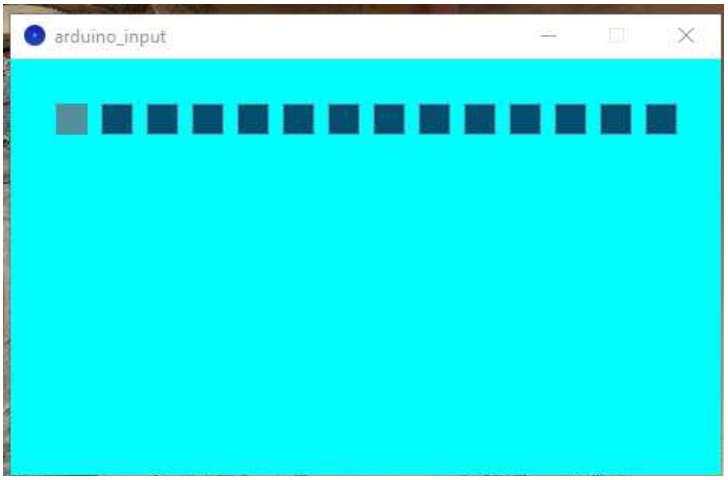
Para o assunto da seção 2.5 foi elaborado um código com os exemplos do Firmata, onde temos um programa para verificar se as portas digitais estão recebendo nível lógico alto. Inicialmente deve-se ir na interface do arduíno fazer o upload do "*SimpleDigitalFirmata*". Depois executar na interface de programação do Processing o código nomeado "ArduinoInput" que estará na pasta:

$$C:\backslash Users\backslash Windows\backslash Documents\backslash Processing\backslash libraries\backslash arduino\backslash examples\backslash arduino\_input$$

Ao executá-lo será exibida uma caixa com quadrados que representam os 13 pinos digitais, que estão em nível baixo. O usuário pode utilizar um pino de 3.3V ou de 5V do próprio arduíno e em seguida conectar essa tensão em uma das portas digitais, para visualizar na interface que o retângulo mudará de cor, indicando que aquele pino agora está em nível alto, como apresentado na Figura 37.

[16] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Ketai/Exemplo3>
Acesso em 02 nov, 2023.

Figura 37: Verificando portas digitais.



FONTE: Firmata Processing.

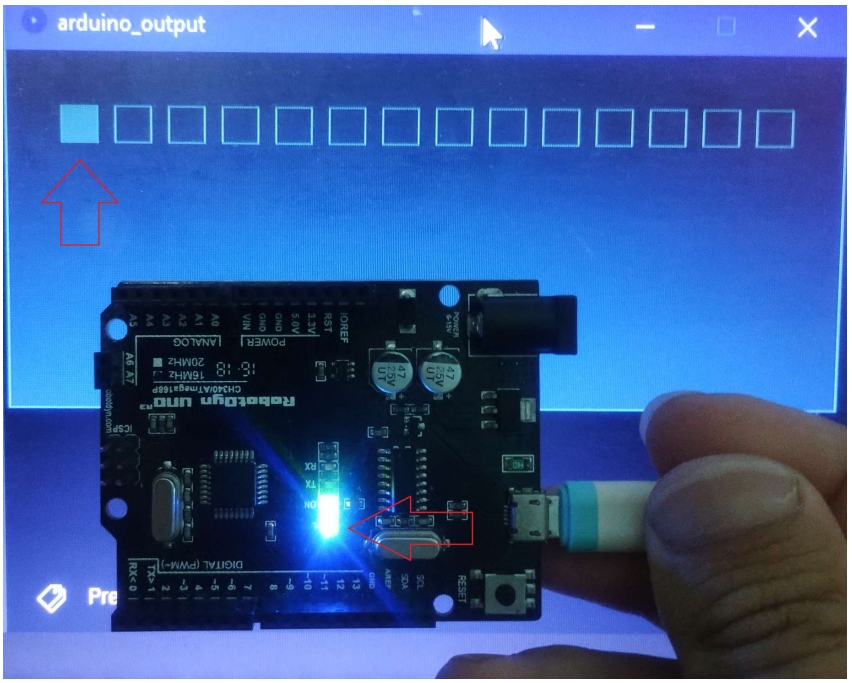
Um segundo Programa foi elaborado, mas agora para controlar as portas digitais em nível ALTO e BAIXO. No Arduíno ainda utilizaremos o código "*SimpleDigitalFirmata*".

Nele foi utilizado primeiro um recurso já visto anteriormente para ajudar no primeiro momento que um programa é executado, com o objetivo de evitar erros de porta desconectada ou errada, adicionamos a biblioteca "*javax.swing.JOptionPane*" para que seja aberta uma janela perguntando em qual porta o arduino está conectado, já que em uma interface gráfica não é exibido o comando do console "*println(Arduino.list())*" que imprime quais portas seriais estão disponíveis para se comunicar com o Arduino.

Em seguida no *setup()* é criada uma variável do tipo "*portaCOM*" que inicia com o valor 0, mas que vai receber o número informado pelo usuário declarado em uma variável do tipo string chamada input. Em seguida a linha usual "*arduino = new Arduino(this, Arduino.list()[0], 57600)*" é modificada para "*arduino = new Arduino(this, "COM"+portaCOM, 57600)*". Se a porta informada não tiver comunicação, um aviso é exibido pedindo para que seja digitado novamente outro valor até que se estabeleça uma conexão.

Em *draw()* o programa exibe se a porta está no nível alto ou baixo, como apresentado na Figura 38 em que foi ativado o próprio LED 13. Esse valor é alterado com o auxílio da função *mousePressed()* que utiliza o clique no mouse em uma região delimitada por coordenadas.

Figura 38: Led 13 do arduino ligado com indicativo na interface.



FONTE: Autoria própria[17]

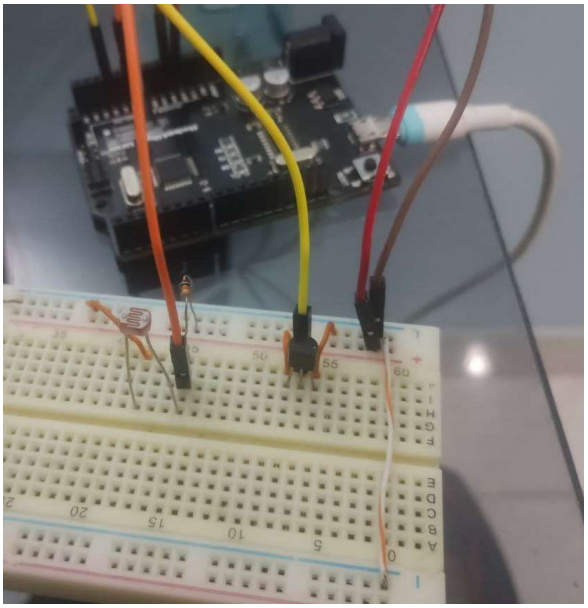
4.1.10 Obter e Gravar Dados em Arquivos de Texto

Também é possível adquirir dados de sensores conectados a um microcontrolador arduino utilizando uma interface gráfica no Processing. Nessa aplicação utilizamos os recursos do Firmata, fazendo o upload no arduíno do código "*SimpleAnalogFirmata*" já que só será necessário utilizar as portas analógicas.

Como apresentado na Figura 39, na montagem utilizamos um arduíno com chip ATMega 168, jumpers, um sensor de temperatura LM35 e um sensor de luz(LDR). O sensor LM35 possui 3 terminais, um de alimentação, um de referência e o pino do meio para saída de dados na forma de tensão. O sensor LDR possui apenas 2 terminais, que funciona fazendo um divisor de tensão com um resistor de 10000 Ohms. O arduino se comunica via porta USB com o computador, através da interface serial. O objetivo é obter esses dados e gravá-los em arquivos de texto que posteriormente podem ser utilizados para uma análise ou construção de gráficos.

[17] Disponível em: <https://github.com/filipedonato/Processing/tree/main/Arduino/Firmata/arduino_output> Acesso em 02 nov, 2023.

Figura 39: Montagem dos componentes.



FONTE: Autoria própria.

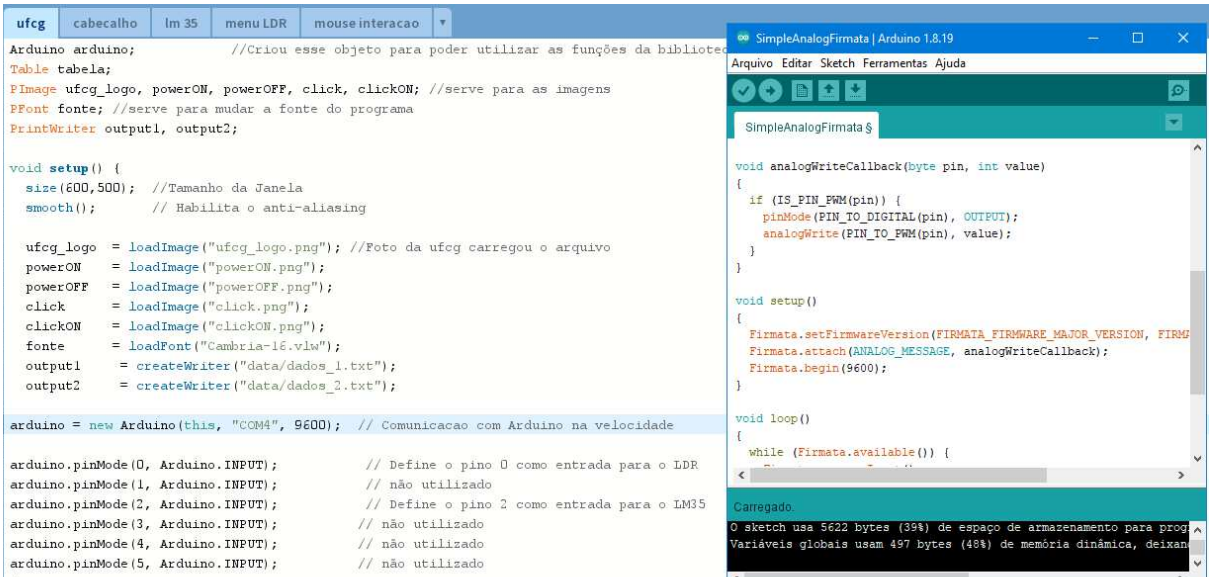
Na Figura 40 apresenta-se parte do código sendo exibida. No código temos que importar as bibliotecas "*processing.serial*" e "*cc.arduino*" que possuem as funções específicas utilizadas para facilitar a comunicação. Depois são criadas as variáveis necessárias para construção da interface como *PImage* (para imagens), *PFont* (para utilizar uma fonte específica) e *PrintWriter* (para criar arquivos de texto).

No *setup()* são definidos parâmetros como o tamanho da tela, as imagens a serem carregadas(que devem estar na pasta data) e a localização dos arquivos de texto. Também foi definido manualmente a porta COM na qual o microcontrolador estará conectado e a atribuição das portas analógicas como entrada de dados.

Em *draw()* foi definido a cor de fundo da tela, e chamadas as funções que estão nas abas laterais. A função *cabecalho()* exibe os dados na parte superior do programa. A função *menu_LDR()* é responsável por exibir os valores do sensor que varia sua resistência com a luz que incide diretamente nele. A função *lm_35()* exibe os valores de temperatura em graus celsius, fazendo sua conversão dentro do código. Por fim a função *mouse_interacao()* é responsável por desenhar na tela os botões que obtém os dados no momento em que clicamos o mouse no botão, com o auxílio de *mousePressed* que delimita o local onde é válido o clique

do mouse, alterando o valor de uma variável booleana e utilizando *output.println* para gravar esse dado no arquivo de texto dentro da pasta data.

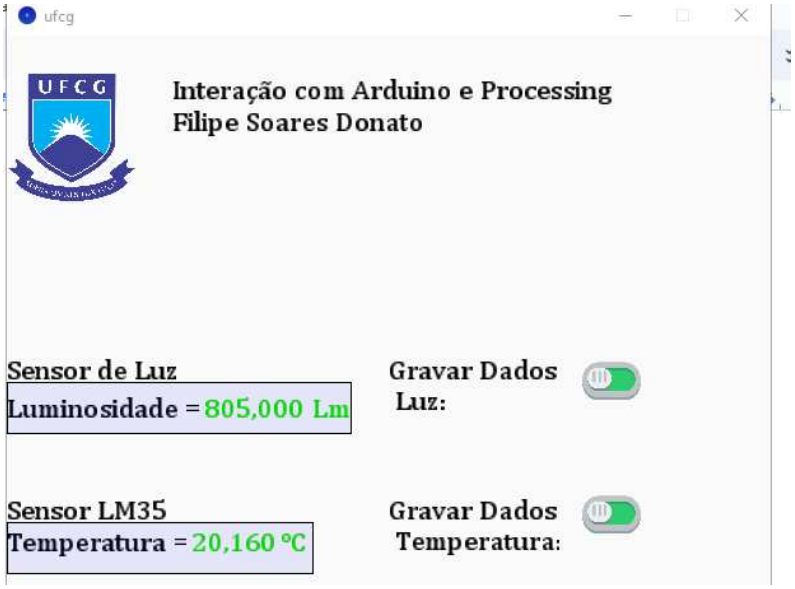
Figura 40: Parte dos códigos necessários.



FONTE: Autoria própria.

Na Figura 41 apresenta-se o resultado de uma interface gráfica, que possui design simples mas possibilita o necessário.

Figura 41: Interface gráfica

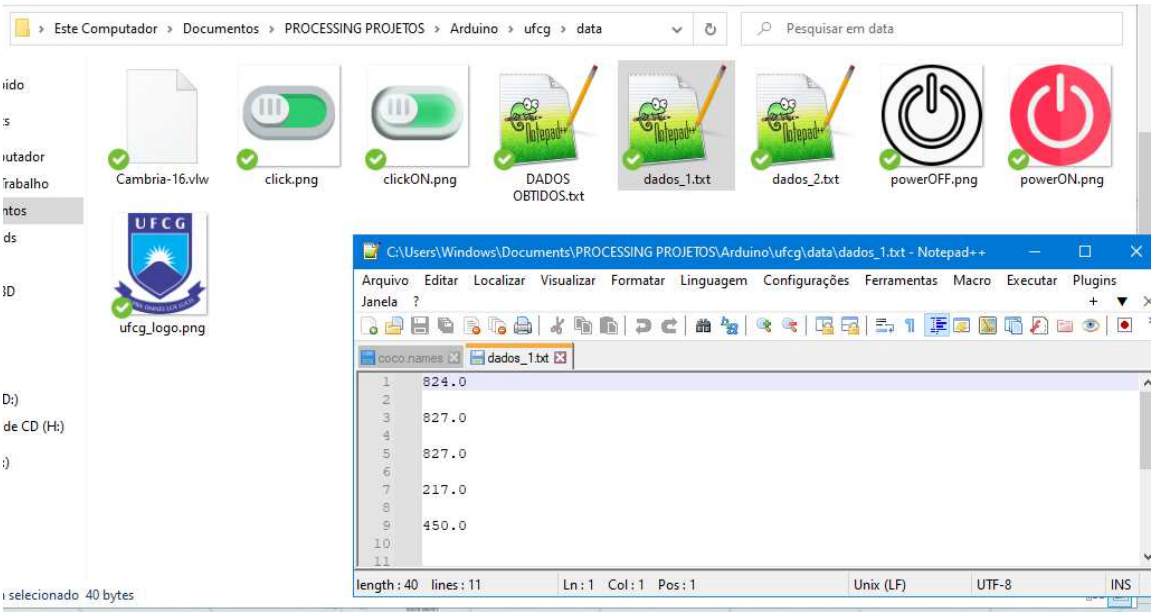


FONTE: Autoria própria[18]

[18] Disponível em:<https://github.com/filipedonato/Processing/tree/main/Arduino/ufcg> Acesso em 02 nov, 2023.

Na pasta de dados temos os arquivos de imagem dos botões e também os arquivos de texto, uma para os dados do LM35 e outro para os valores obtidos pelo LDR, como apresentado na Figura 42. Uma aplicação útil é monitorar por exemplo a temperatura de um dispositivo com o LM35 e a cada período de tempo definido pelo usuário (por exemplo 1 minuto), clicar no botão “Gravar dados Temperatura” e ao final plotar um gráfico e fazer as análises necessárias.

Figura 42: Interface gráfica.



FONTE: Autoria própria.

5. Conclusões

O presente estudo possibilitou a demonstração das diversas aplicações que a linguagem Processing oferece, com um foco especial na engenharia. Os programas desenvolvidos aproveitaram recursos disponibilizados tanto pela equipe principal quanto pela comunidade de desenvolvedores, que continuamente contribuem com novas funcionalidades de código aberto e gratuitas.

Os objetivos propostos foram alcançados com sucesso. Isso inclui a demonstração das capacidades da linguagem, a resolução de problemas matemáticos, a criação de novas aplicações com ênfase em elementos gráficos, a aplicação de filtros em sinais, a comunicação e a aquisição de dados com microcontroladores, bem como a exploração de recursos para a detecção de objetos em imagens e vídeos em tempo real.

REFERÊNCIAS BIBLIOGRÁFICAS

PROCESSING FOUNDATION. Disponível em: <https://processingfoundation.org>. Acesso em: 7 ago. 2023.

PROCESSING. Environment. Disponível em: <https://processing.org/environment>. Acesso em: 12 out. 2023.

PROCESSING. randomGaussian() / Reference. Disponível em: <https://processing.org/reference/randomGaussian_.html>. Acesso em: 12 out. 2023.

GITHUB. processing/license.txt at master · processing/processing. Disponível em: <https://github.com/processing/processing/blob/master/license.txt>. Acesso em: 12 out. 2023.

LISBOA, Alvenis. Qual é o sistema operacional mais usado do mundo? Disponível em: <https://canaltech.com.br/software/qual-e-o-sistema-operacional-mais-usado-do-mundo-223507/>.

FIRMATA PROTOCOL. Documentation. Disponível em: <https://github.com/firmata/protocol#firmata-protocol-documentation>. Acesso em: 12 out. 2023.

FIRMATA. Disponível em: <https://github.com/firmata/arduino#firmata>. Acesso em: 12 out. 2023.

ALVES, G. Detecção de Objetos com YOLO - Uma abordagem moderna. Disponível em: <https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/?doing_wp_cron=1691078976.0960600376129150390625#:~:text=Darknet&text=Para%20o%20seu%20funcionamento%20o,criador%20do%20YOLO%2C%20Joseph%20Redmon.>. Acesso em: 4 ago. 2023.

VARGAS, D. Rastreamento de Objetos x Detecção de Objetos. Disponível em: <https://iaexpert.academy/2020/06/24/rastreamento-de-objetos-x-deteccao-de-objetos/?doing_wp_cron=1697128738.3889830112457275390625>. Acesso em: 12 out. 2023.

FELIPE, G. O que é CUDA? Disponível em: <https://www.oficinadanet.com.br/post/14818-o-que-e-cuda>. Acesso em: 12 out. 2023.

BRUGGISSER, F. Deep Vision Processing. Disponível em: <https://github.com/cansik/deep-vision-processing>. Acesso em: 3 ago. 2023.

WOBBROCK, J.; WILSON, A.; LI, Y. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. [s.l: s.n.]. Disponível em: <https://faculty.washington.edu/wobbrock/pubs/uist-07.01.pdf>. Acesso em: 12 out. 2023.

CASIEZ, G. ROUSSEL, N.; VOGEL, D. 1 € filter. HAL (Le Centre pour la Communication Scientifique Directe), 5 mai 2012.

CASIEZ, G. 1€ Filter. Disponível em: <https://gery.casiez.net/1euro/>. Acesso em: 4 ago. 2023.

SOLUTIONS, N. O que é Perlin Noise e Como Funciona? Disponível em: <https://novageo.pt/novageo/displayArticles?numero=38577&_que_perlin_noise_como_funciona>. Acesso em: 12 out. 2023.

ERNESTO, L. noise()» Linguagem (API)» Processing 1.0 (BETA). Disponível em: <https://pessoal.dainf.ct.utfpr.edu.br/merkle/processing/reference/ptBR/noise_.html>. Acesso em: 12 out. 2023.

FFMPEG. About. Disponível em: <https://ffmpeg.org/about.html>. Acesso em:17 out. 2023.

APACHE. Apache Commons Math 3.6.1 API. Disponível em: <https://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/index.html>. Acesso em:10 out. 2023.

UFRGS. Ruídos. Disponível em: <http://penta2.ufrgs.br/Antonio/ruido.html>. Acesso em:10 out. 2023.

COURVILLE, R. DE. Signal Filter. Disponível em: <https://github.com/SableRaf/signalfilter>. Acesso em: 10 out. 2023.

PAZOS, A. Video Export. Disponível em: <https://funprogramming.org/VideoExport-for-Processing/>. Acesso em:10 out. 2023.

KETAI. Examples. Disponível em: <http://ketai.org/examples/>. Acesso em: 10 out. 2023.

REDMON, J. YOLO: Real-Time Object Detection. Disponível em: <https://pjreddie.com/darknet/yolo/>. Acesso em:10 out. 2023.

JOCHER, G. ultralytics/yolov5. Disponível em: <https://github.com/ultralytics/yolov5>. Acesso em:10 out. 2023.

FILIPEDONATO. Processing. Disponível em: <https://github.com/filipedonato/Processing/tree/main>. Acesso em: 2 nov. 2023.

ANEXOS

ANEXO A. Algoritmo do Filtro 1 Euro

Algorithm 1: 1€ filter

EXT: First time flag: *firstTime* set to *true*
Data update rate: *rate*
Minimum cutoff frequency: *mincutoff*
Cutoff slope: *beta*
Low-pass filter: *xfilt*
Cutoff frequency for derivate: *dcutoff*
Low-pass filter for derivate: *dxfilt*
IN : Noisy sample value: *x*
OUT: Filtered sample value

```
1 if firstTime then
2 |   firstTime ← false
3 |   dx ← 0
4 else
5 |   dx ← (x - xfilt.hatxprev()) * rate
6 end
7 edx ← dxfilt.filter(dx, alpha(rate, dcutoff))
8 cutoff ← mincutoff + beta * |edx|
9 return xfilt.filter(x, alpha(rate, cutoff))
```

Algorithm 2: *Filter* method of Low-pass filter

EXT: First time flag: *firstTime* set to *true*
IN : Noisy sample value : *x*
Alpha value : *alpha*
OUT: Filtered value

```
1 if firstTime then
2 |   firstTime ← false
3 |   hatxprev ← x
4 end
5 hatx ← alpha * x + (1 - alpha) * hatxprev
6 hatxprev ← hatx
7 return hatx
```

Algorithm 3: Alpha computation

IN : Data update rate in Hz: *rate*
Cutoff frequency in Hz: *cutoff*
OUT: Alpha value for low-pass filter

```
1 tau ← 1.0 / (2*π*cutoff)
2 te ← 1.0 / rate
3 return 1.0 / (1.0 + tau/te)
```