CATÓLICA
ESCOLA DAS ARTES
PORTO

# DIGIUNIVERSAL: ESTUDO DE CASO SOBRE O DESENVOLVIMENTO DE UM INSTRUMENTO MUSICAL DIGITAL

Relatório de Estágio apresentado à Universidade Católica Portuguesa
para obtenção do grau de Mestre em Som e Imagem.

*Miguel Guilherme de Resende Ramos Bastos*

Porto, setembro de 2024

CATÓLICA
ESCOLA DAS ARTES
PORTO

# DIGIUNIVERSAL: ESTUDO DE CASO SOBRE O DESENVOLVIMENTO DE UM INSTRUMENTO MUSICAL DIGITAL

Relatório de Estágio apresentado à Universidade Católica Portuguesa
para obtenção do grau de Mestre em Som e Imagem

- Especialização em Design de Som-

*Miguel Guilherme de Resende Ramos Bastos*

Trabalho efetuado sob a orientação de

Prof. Doutora Cristina Sá

Porto, setembro de 2024

*Aos meus avós Fátima, Dulce e Ernesto.*

# Agradecimentos

Este trabalho não teria sido possível a orientação da Professora Cristina Sá, ou sem a colaboração dos meus colegas da Digitópia Óscar Rodrigues, Ricardo Vieira, Filipe Fernandes e Sérgio Coutinho, a quem agradeço também pelo companheirismo e amizade.

*Um agradecimento especial à Filipa, à minha família e amigos, que me acompanham e aturam incondicionalmente.*

# Resumo

A democratização do acesso à tecnologia trouxe consigo o aumento da capacidade de computação, a proliferação de microcomputadores como o "Raspberry Pi" e "Bela" e uma cada vez maior disponibilidade de sensores prontos a utilizar, o que levou a que a investigação e criação de instrumentos musicais digitais aumentasse de forma sustentada nas duas últimas décadas. Acompanhando o crescente alerta para as problemáticas da inclusão social, aumentou também o interesse na investigação e criação de instrumentos musicais digitais acessíveis.

Inspirando-se no conceito de design universal, definido originalmente por Ronald Mace como o design de produtos utilizáveis por todas as pessoas, sem a necessidade de adaptação ou design especializado, a Digitópia lançou o projeto DigiUniversal, com o objetivo de criar e desenvolver instrumentos universais, passíveis de serem tocados por qualquer pessoa, quaisquer que sejam suas características e aptidões físicas, intelectuais e musicais.

Este relatório assume a forma de um estudo de caso acerca do processo de desenvolvimento de um destes instrumentos — a Caixa da Música — concluindo que, apesar de o principal objetivo ser a sua universalidade, estes não podem deixar de ser pensados como instrumentos musicais destinados a serem tocados com uma determinada fisicalidade, sendo fundamental explorar técnicas e estratégias de desenvolvimento e design de som que potenciem uma estrita ligação entre gesto físico e resultado sonoro.

**Palavras Chave:** Instrumento musical digital, síntese sonora, síntese digital por guias de onda, Pure Data, SuperCollider, Bela.

**Keywords:** Digital musical instrument, sound synthesis, digital waveguide synthesis, Pure Data, SuperCollider, Bela.

**Índice**

**Lista de Figuras**

# 1 Introdução

## 1.1 Instituição acolhedora: Digitópia, Casa da Música

A Fundação Casa da Música é uma instituição cultural localizada no Porto, dedicada à promoção da música e à educação musical. Inaugurada em 2005, é uma das principais referências culturais do país e a sua sede é um dos edifícios mais emblemáticos da cidade, projetado pelo arquiteto holandês Rem Koolhaas. O objetivo da Fundação é promover atividades artísticas e culturais, oferecendo uma programação variada que inclui concertos de música clássica, contemporânea, jazz, pop e outros géneros, além de workshops, exposições e iniciativas educativas.

A Digitópia é a plataforma artística da Casa da Música, dedicada à criação de música através da utilização de recursos digitais. Como coletivo criativo, especializa-se no desenvolvimento de ferramentas para exploração musical, tais como o design de instrumentos digitais, a criação de hardware e software próprios e *circuit-bending*. Utilizando meios eletrónicos de geração sonora e sistemas digitais interativos, a Digitópia explora a relação entre imagem e som, pratica VJ'ing e DJ'ing, criando diversos tipos de espetáculo e concertos. (*Digitópia*, sem data)

Uma das funções desempenhadas pela Digitópia é a preparação e operação de media digitais, em colaboração com os conjuntos residentes da Casa da Música, assumindo a componente eletrónica em tempo real na interpretação do repertório dos séculos XX e XXI. (*Agenda 2024 - Casa da Música*, sem data)

## 1.2 Breve descrição das funções assumidas

Ao longo do estágio, procurei dar o meu contributo enquanto músico, programador e designer de som para o sucesso dos projetos em curso. Trabalhei com software como Max, Pure Data, SuperCollider, Digital Audio Workstations e outros softwares de edição de áudio, para além de programar microcomputadores como Arduino ou Bela.

Dadas as minhas competências e formação anterior como compositor, colaborei também na preparação, ensaio e performance em tempo real de eletrónica em concerto.

## 1.3 Estrutura do relatório

Apesar de o estágio ter sido de grande abrangência, requerendo uma variedade de competências

nas áreas da música, som, áudio, programação e eletrónica e de as atividades realizadas ao longo do estágio refletirem esta abrangência, optei por concentrar este relatório no desenvolvimento da Caixa da Música: um dos instrumentos universais que a Digitópia está a desenvolver como parte do projeto DigiUniversal, passíveis de serem tocados por qualquer pessoa, quaisquer que sejam as suas características e aptidões físicas, intelectuais e musicais. O relatório engloba três fases do projeto: conceção, desenvolvimento do primeiro protótipo e desenvolvimento do segundo protótipo.

O meu envolvimento em cada uma das fases foi bastante distinto, sendo que a conceção do instrumento já estava praticamente concluída quando dei início ao estágio. O meu papel no desenvolvimento do primeiro protótipo foi principalmente de programador assistente, ainda que tenha feito alguma consultoria na fase de design de som. Só na terceira fase assumi o papel de desenvolvedor principal, com maior autonomia para tomar decisões.

No final do relatório existe um resumo das restantes atividades que decorreram ao longo do estágio.

## 2 Cronograma de atividades

Duração do estágio: 6 meses (24 semanas).

Data de início: 18 de setembro.

Data de fim: 18 de fevereiro.

### 2.1 Ferrofone

- Descrição: Criação de uma instalação musical interativa.
- Encarregado: Filipe Fernandes.
- Minha função: Programador assistente.
- Duração: Setembro a outubro.

### 2.2 DigiUniversal: Caixa da Música

- Descrição: Criação de um instrumento musical digital universal.
- Encarregado: Ricardo Vieira (primeiro protótipo); eu (segundo protótipo).
- Minha função: Programador assistente; desenvolvedor principal; designer de som.
- Duração: Setembro até ao presente (apresentação do segundo protótipo ao público em Outubro).

### 2.3 Orelhudo!

- Descrição: Plataforma pedagógica para alunos do 1º ciclo.
- Minha função: Editor de áudio.
- Duração: Janeiro até ao presente.

### 2.4 Du cristal

- Descrição: Peça orquestral com sintetizador.
- Minha função: Técnico de sintetizador.
- Duração: Fevereiro (concerto em Março).

### 2.5 A Madrugada

- Descrição: Peça para coro, orquestra e eletrónica.
- Minha função: Foley, gravação e mistura.
- Duração: Janeiro a fevereiro (concerto em Abril).

# 3 Caixa da Música

## 3.1 Proposta: Sanfona Digital

A proposta inicial para a Caixa da Música, redigida por Ricardo Vieira, partiu da ideia de criar uma Sanfona Digital inspirada, como o nome indica, no instrumento acústico homónimo (também conhecido em inglês como Hurdy-Gurdy), bem como as suas versões eletrónicas já existentes: o Digi-Gurdy (*Digi-Gurdy*, sem data) e o MidiGurdy (*MidiGurdy*, sem data).

*Figura 2: Tocador de sanfona numa feira medieval.*

Segundo o sistema de classificação de instrumentos Hornbostel-Sachs (Henrique, 2004), a sanfona é um cordofone friccionado, com caixa de ressonância e braço à semelhança de instrumentos mais conhecidos, como o violino, a viola d'arco, o violoncelo e o contrabaixo, imprescindíveis em qualquer orquestra. O que distingue a sanfona dos restantes cordofones friccionados é o método utilizado para friccionar as cordas. Enquanto que os seus parentes orquestrais fazem uso de um arco de cerdas enresinadas para excitar as cordas e produzir som, na sanfona as cordas são friccionadas por uma roda de madeira, também ela enresinada, cuja rotação é acionada por meio de uma manivela situada na extremidade da caixa oposta ao braço. O braço do instrumento é largo, em forma de caixa e aloja um mecanismo semelhante a um teclado, que aciona pequenas linguetas de madeira tangentes às cordas, dividindo-as em pontos precisamente calculados, alterando assim o seu comprimento vibrante e consequentemente, a nota produzida (Henrique, 2004). Como tal, pode dizer-se que a sanfona é um instrumento de afinação absoluta.

*Figura 1: Representação de um organistrum no Pórtico de la Gloria, na Catedral de Santiago de*

Como termo de comparação, num violino, por exemplo, o comprimento vibrante das cordas é alterado por ação dos dedos do instrumentista, que pisam as cordas em posições aproximadas e afinadas pelo próprio ao tocar, sendo o braço estreito para acomodar a posição da mão. Como tal, o violino, bem como a sua restante família, é classificado como um instrumento de afinação relativa.

A origem da sanfona remonta à Idade Média e, segundo as representações iconográficas da época, era tocada por dois executantes sentados lado a lado, com o instrumento pousado em cima dos joelhos: um dava à manivela e o outro acionava o teclado. O instrumento chegava a ter um comprimento de 1,5 m e recebia o nome de *organistrum* ou *symphonie*. Atualmente tem dimensões menores e é tocado por um só instrumentista, que dá à manivela com a mão direita e aciona o teclado com a mão esquerda, podendo ser pousado no colo ou pendurado ao pescoço através de um tirante. A sanfona era principalmente um instrumento popular, muitas vezes tocado por músicos ambulantes, sendo hoje utilizado sobretudo na Europa Central (Henrique, 2004).

O Digi-Gurdy e o MidiGurdy (descontinuado) são versões comerciais eletrónicas da sanfona, que podem ser utilizados como instrumento ou como controlador MIDI. O primeiro foi pensado como instrumento de estudo ou para principiantes, sendo consideravelmente mais barato que uma sanfona acústica.

A proposta para a Sanfona Digital (atualmente Caixa da Música) procurou não necessariamente replicar uma sanfona eletrónica — pois seria redundante — mas sim criar um Instrumento Universal cujo modo de produção sonora fosse tão simples como rodar uma manivela, sendo que o som seria assumidamente eletrónico, ao invés de simular o som de uma sanfona.

A proposta considerava duas possibilidades para o instrumento: apenas uma manivela ou duas manivelas. Um instrumento com duas manivelas permitiria controlar de forma independente um bordão e um gerador de melodias, podendo até ser tocado simultaneamente por duas pessoas, tal como era originalmente tocado o *organistrum*. Por outro lado, a estrutura do instrumento teria de ser resistente a forças e movimentos distintos e, para poder ser tocado por duas pessoas com mobilidade reduzida (em cadeira de rodas, por exemplo), teria de ser acoplado a uma estrutura com dimensões e robustez adequadas. Um instrumento com uma manivela, destinado a ser tocado por apenas uma pessoa, poderia ser mais pequeno e portátil o suficiente para ser pousado sobre o colo do executante. No entanto, a relação entre bordão e melodia teria de ser pré-estabelecida, perdendo-se a independência entre os dois elementos, que era de interesse musical.

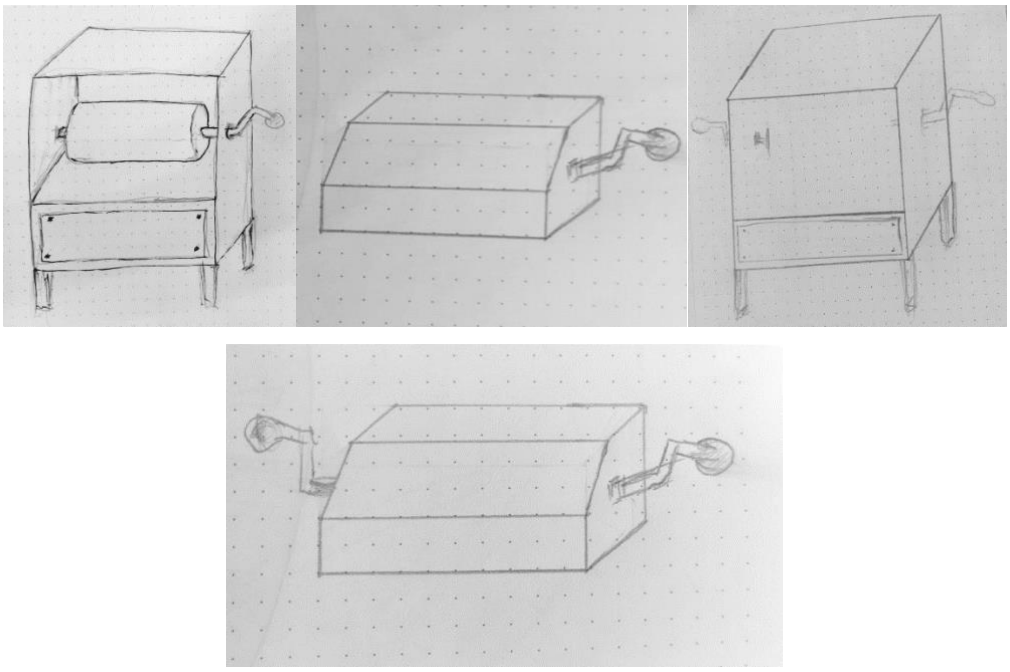


*Figura 3: Alguns esboços iniciais da Caixa da Música, desenhados por Ricardo Vieira.*

Quanto à relação entre gesto e som, propôs-se uma ligação entre a rapidez do gesto e a complexidade harmónica, sendo que gestos lentos corresponderiam a uma menor densidade de vozes (menor quantidade de camadas sonoras sobrepostas e harmonicamente mais simples), enquanto que gestos rápidos corresponderiam a uma maior densidade de vozes (maior quantidade de camadas sobrepostas e harmonicamente mais complexo). Como se o movimento da manivela fosse convertido em energia introduzida na máquina e alimentasse mais motores de vozes. O segundo aspeto proposto para a relação entre gesto e som prende-se com a acentuação musical. À semelhança do que acontece com a sanfona, uma aceleração repentina do gesto corresponderia a uma acentuação no timbre (um *buzz* obtido através de distorção harmónica), o que permitiria que o bordão pudesse ser também um elemento gerador de pulsação. Finalmente, propôs-se também que a inversão do sentido de rotação da manivela correspondesse a um efeito de rebobinagem, sendo o som reproduzido em reverso.

***Nota:*** *Quando dei início ao estágio, o conceito do instrumento já estava completamente elaborado e o projeto já tinha avançado para a fase de desenvolvimento do primeiro protótipo. Havia já uma pequena prova de conceito do mecanismo da manivela construída, mas toda a parte eletrónica, software e design de som estavam por começar. Foi-me proposto que acompanhasse e colaborasse no desenvolvimento deste instrumento.*

### 3.2 Rotary encoders (codificadores rotativos)

O primeiro problema levantado pela proposta prende-se com a deteção do movimento da manivela. Verificámos que, quer o Digi-Gurdy, quer o MidiGurdy usam um *incremental rotary encoder* (codificador rotativo relativo) para este efeito. Um encoder é um componente eletromecânico que lê a posição de um elemento rotativo, normalmente associado a um veio central, e gera um sinal elétrico associado ao movimento de rotação. Existem três tipos de encoders (*Rotary Encoders | What is a Rotary Encoder? Rotary Encoders Explained*, sem data):

- Encoders magnéticos fazem uso de uma roda magnetizada ligada a um veio central. O movimento rotativo do eixo é transmitido para a roda, cuja posição angular é lida por um sensor magnético. (É assim que funciona um velocímetro de uma bicicleta, por exemplo).
- Encoders óticos transmissivos funcionam de forma semelhante, mas fazem uso de um disco ranhurado, em vez de uma roda magnetizada. De um lado do disco, um LED emite um feixe de luz que passa através das ranhuras e atinge um sensor fotoelétrico posicionado do lado oposto. À medida que o veio e o disco vão rodando, o movimento das ranhuras provoca uma interrupção periódica do feixe de luz, causando a geração de sinal elétrico correspondente ao movimento de rotação à saída do sensor.

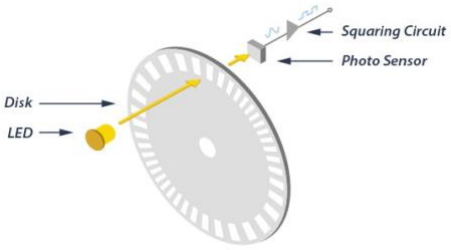


*Figura 4: Diagrama de encoder ótico.*

- Encoders óticos refletivos funcionam de forma semelhante aos transmissivos, mas em vez de o feixe atravessar o disco, este reflete a luz para um sensor fotoelétrico posicionado do mesmo lado que o LED. Este formato permite o fabrico de encoders mais compactos.

Os encoders também podem ser distinguidos de acordo com o tipo de informação que são capazes de ler e transmitir, podendo a leitura da posição angular ser feita de forma relativa (*incremental rotary encoders*) ou absoluta (*absolute rotary encoders)* (*Rotary Encoder - Northwestern Mechatronics Wiki*, sem data). A distinção deve-se à forma como a posição é codificada no próprio disco.

Os modelos mais simples de encoder ótico possuem uma só fileira de ranhuras igualmente espaçadas e são capazes de determinar apenas a posição relativa e a velocidade de rotação. O resultado é um sinal de onda quadrada (binário), cuja frequência está associada à velocidade de

rotação. A resolução do encoder depende do número de ranhuras no disco e é indicada pelo fabricante em ppr (*pulses per revolution)* — o número de impulsos por cada volta completa do disco. Quanto mais lento o movimento medir, maior terá de ser a resolução do encoder.

Para ser possível distinguir o sentido da rotação, é necessário acrescentar uma segunda fileira (sinal $\beta$), paralela à primeira (sinal $\alpha$) e com o mesmo número de ranhuras, bem como um segundo feixe de luz (e respetivo sensor fotoelétrico). As fileiras devem estar desfasadas de forma que os dois sinais tenham entre si uma relação de fase de 90°. O sentido da rotação pode ser determinado da seguinte forma:

- Sentido positivo:
  - $\alpha$: transição $0 \rightarrow 1$, então $\beta = 1$;
  - $\alpha$: transição $1 \rightarrow 0$, então $\beta = 0$.
- Sentido negativo:
  - $\alpha$: transição $0 \rightarrow 1$, então $\beta = 0$;
  - $\alpha$: transição $1 \rightarrow 0$, então $\beta = 1$.

$\therefore$ Quando $\alpha$ em transição:

- Se $\alpha = \beta \Rightarrow$ sentido positivo;
- Se $\alpha \neq \beta \Rightarrow$ sentido negativo.

Num encoder absoluto, o disco é dividido em segmentos com uma codificação individual única, que permite determinar a sua posição angular absoluta. Este método requer múltiplos sensores e fileiras de ranhuras organizadas num padrão complexo. Aumentar a resolução implica um maior número de fileiras e sensores, fazendo com que os encoders absolutos sejam mais difíceis e dispendiosos de fabricar.

#### Fabricar um encoder

Uma vez que se trata de um instrumento musical, era importante para nós que a deteção de movimento da manivela tivesse a maior resolução possível, para que a relação entre gesto e som tivesse a maior granularidade e o mínimo de latência possível. Por outro lado, uma vez que o instrumento deve ser o mais universalmente acessível possível e tem inclusivamente um fim pedagógico, o custo de fabrico é também uma questão relevante. Ao pesquisar os encoders disponíveis no mercado, rapidamente percebemos que seria impossível encontrar um compromisso viável entre custo e resolução (um custo de 50 € a 60 € seria incomportável).

Optamos então por produzir o nosso próprio encoder ótico relativo, fazendo uso de um sensor de velocidade por infravermelhos *Whadda WPSE347* (que atualmente pode ser adquirido por 8,14 €), que nos permitiu imprimir o nosso próprio disco, com uma resolução de 200 ppr. Por motivos de acessibilidade optamos por não distinguir o sentido de rotação, portanto bastou-nos desenhar um disco com uma única fileira de ranhuras.

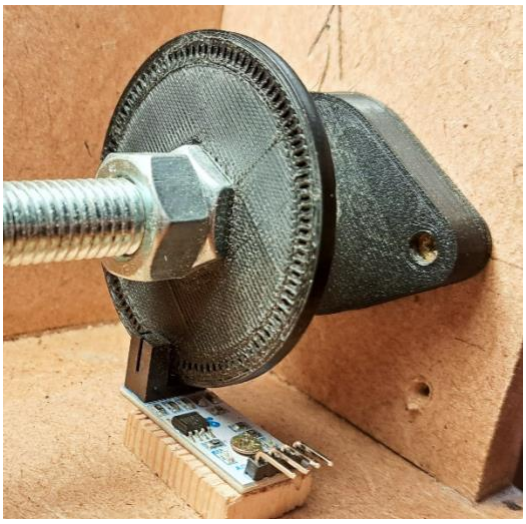


*Figura 5: Protótipo do encoder.*

### 3.3 Latência

Um problema comum a qualquer instrumento digital ou interface tangível destinada à performance musical é a latência: o atraso existente entre a ação do músico e a produção de som. Qualquer instrumentista está familiarizado com o tempo de ataque do seu instrumento acústico/eletroacústico; ou talvez se deva dizer *os tempos*, uma vez que para muitos instrumentos, este tempo varia de acordo com a nota tocada, sendo que as notas mais graves dos instrumentos mais graves (e.g. tuba, contrabaixo, voz baixo profundo) têm um tempo de ataque muito lento. Para um instrumentista treinado e em forma, isto não é um problema; com a prática, o tempo de ataque de cada nota passa a ser previsível e compensar torna-se instintivo e é uma parte fundamental da técnica do instrumento.

No que toca a instrumentos ou controladores digitais, as causas da latência são várias: a limitação de banda das ligações, drivers não otimizados para baixa latência, atrasos e variações no *scheduling* do software, bem como a latência inerente ao áudio digital, que está dependente do *buffer size* definido; todos estes elementos têm um efeito cumulativo que resulta na latência final do instrumento. Da mesma forma que um instrumentista é capaz de compensar o tempo de ataque, também é possível compensar auditivamente a latência até 55 ms (Dahl & Bresin, 2001). No entanto, é preciso ter em conta que isto requer um esforço adicional de concentração da parte do músico, principalmente para latências elevadas; esforço esse que seria mais bem empregue na qualidade musical da performance.

Em 2016, na conferência internacional NIME, McPherson, Jack e Moro colocaram a questão

"*How fast is "fast enough"?*"[1]. Segundo os autores, a referência de uma latência máxima de 10 ms, proposta por Wessel e Wright em 2002 é provavelmente a mais utilizada na comunidade de construtores de instrumentos musicais digitais. No entanto, (tal como tudo o que tem que ver com música) a tolerância à latência está altamente dependente do contexto musical. Interações percussivas são mais suscetíveis a serem influenciadas pela latência, uma vez que uma dessincronização tão pequena quanto 6 ms numa pulsação regular já é passível de ser detetada por um ouvinte. Por outro lado, sons com um transitório de ataque mais lento e cuja interação está mais dependente de um gesto contínuo permitem latências mais elevadas de 20 ms a 30 ms (Lago & Kon, 2004; Mäki-Patola & Hämäläinen, 2004). Finalmente, a distância do músico à fonte sonora — que no caso de som eletrónico pode ser um altifalante a uma distância considerável — também não deve ser menosprezada, uma vez que o tempo que o som demora a percorrer distâncias elevadas também contribui para a sensação de latência.

### 3.4 Bela e Bela mini

Uma configuração comum usada na criação de instrumentos musicais digitais e outras interfaces tangíveis envolve o uso de sensores de diversos tipos, ligados a um microcontrolador (e.g. Arduino), que por sua vez está ligado — com ou sem fios — a um computador. Através do uso de protocolos de comunicação por rede (e.g. Open Sound Control) a informação processada pelo microcontrolador é transmitida a um software gerador de som, sendo Max, Pd (Pure Data) e SuperCollider os mais usados para o efeito. Esta configuração comporta várias desvantagens e limitações, que a tornam desadequada para o desenvolvimento de instrumentos musicais digitais, de entre as quais:

- Em modo USB-serial (o protocolo de comunicação mais usado com o Arduino), mesmo nas configurações mais rápidas, é praticamente impossível obter uma latência fiável inferior ou igual a 10 ms (McPherson et al., 2016)[2];
- Segundo os autores, o desempenho é melhor quando se considera apenas eventos tipo *trigger*, em vez de um *stream* contínuo de dados. No entanto, isto impede a transmissão de diferentes tipos de informação pelo mesmo canal.
- A necessidade de um computador implica:

[1] Quão rápido é "suficientemente rápido"?

[2]Para minimizar a latência, os autores recomendam o uso dos protocolos USB-MIDI ou HID, cujos drivers estão otimizados para baixa latência (5,1 ms a 16,8 ms), mas estes comportam outro tipo de limitações.

a) A impossibilidade de construir um instrumento *standalone*, a não ser que se use um minicomputador (e.g. Raspberry Pi), requerendo maior investimento e passos adicionais de configuração;
b) No caso de uma instalação artística, o uso de um computador dedicado, que ficará indisponível para outras tarefas durante o decorrer da instalação.

*Figura 6: Bela Mini*

Foi exatamente para resolver estes problemas que a plataforma Bela foi desenvolvida: um sistema de hardware e software capaz de latências infra milissegundo, passível de funcionar em modo *standalone*. O hardware consiste numa placa de expansão acoplada a um computador *single-board* BeagleBone Black ou PocketBeagle (Bela Mini), com pelo menos oito entradas analógicas e dezasseis digitais, duas entradas e duas saídas áudio analógico. O sistema operativo Bela é baseado num kernel Linux Xenomai e permite *buffer sizes* de 2 samples a 44.1 kHz (McPherson et al., 2016). O software inclui um IDE acessível a partir de qualquer browser através de uma ligação de rede USB, permitindo programar e correr projetos em C++, Pd, SuperCollider e Csound (bem como Faust e Pyo, mas ainda em fase experimental).

O sistema Bela conta com vários acessórios de expansão (áudio multicanal e até 64 entradas e saídas analógicas) e sensores capacitivos de toque chamados Trill, desenvolvidos com visa a permitir uma interação orgânica com o sistema.

Optamos por desenvolver o projeto num sistema Bela, pois este permite uma interação muito mais natural e dispensa o uso de um computador para o instrumento funcionar.

### 3.5 Primeiro algoritmo, em Pure Data

De entre todas as linguagens de programação suportadas, o Pure Data é provavelmente a mais acessível e habitualmente usada pela Digitópia, juntamente com o software Max, da Cycling74. O Pure Data (ou Pd) é uma linguagem de programação visual voltada principalmente para criação de música eletrónica, design de som, arte interativa, performances audiovisuais e instalações multimédia. Foi desenvolvida por Miller Puckette nos anos 1990 e é um software de código aberto, disponível para várias plataformas (Windows, Mac e Linux).

Na primeira experiência que fizemos com o sensor, optámos por ligar o encoder à primeira

entrada digital do Bela, assumindo que, uma vez que o sinal era binário, seria a opção mais lógica. Infelizmente, verificámos que o fluxo de dados que entrava no Pd não era constante. A informação entrava em períodos irregulares e por vezes em blocos. Tornou-se claro que este método não serviria para calcular a velocidade de rotação com precisão. Ao pesquisar na documentação do Bela, reparámos que esta trazia um exemplo de código em C++ para um encoder, que utilizava não uma entrada digital, mas uma entrada analógica. E um sinal constante à taxa de amostragem áudio de 44.1 kHz já permitia fazer a leitura correta do movimento.

Resolvido o problema da entrada de sinal no Bela, foi-nos sugerido pelo coordenador da Digitópia Óscar Rodrigues o seguinte algoritmo:

i Registar o número de transições de 0 para 1 num *array*; será 0 se não houver transições e 1 se houver, ou seja, se houver movimento. Guardar os *n* últimos eventos;
ii Fazer duas médias volantes desse mesmo *array*: uma curta (e.g. 100 ms), outra longa (e.g. 5000 ms);
iii Registar o estado atual numa variável global, com valores de 0 a 5: 0 será silêncio, 1 será uma camada, 2 duas camadas e assim sucessivamente;
iv Para subir de estado (0 para 1, ou 1 para 2, ou...) usar a média volante curta;
v Para descer de estado, usar a média volante longa.

Levantou-se a questão de como calcular uma média volante em Pd — à qual nenhum membro da equipa sabia responder. Resolvi voluntariar-me para descobrir como o fazer e assim implementar os dois primeiros pontos do algoritmo proposto.

***Nota:*** *É importante referir que, embora eu tivesse já uma vasta experiência a programar algoritmos de DSP em Max — que por razões históricas tem muitas semelhanças ao Pd[3] — não estava de todo habituado a programar em Pd, e as diferenças entre os dois são suficientes para implicarem um desafio e um processo de aprendizagem. Como tal, o patch passou por três versões, à medida que eu me ia familiarizando com a linguagem e os objetos do programa.*

**Primeiro passo: calcular uma média volante**

i Comecei por gerar um fluxo de valores aleatórios;
ii Criei um *array* de tamanho arbitrário *n*;
iii Programei um pequeno algoritmo simples, que conta ciclicamente de 0 a *n* e servirá de

[33] Ambos foram criados por Miller Puckette.

índice para o *array*;

iv Utilizando o objeto [array set], os valores aleatórios são armazenados ciclicamente no *array*, de acordo com os índices gerados, sendo os valores mais antigos substituídos pelos mais recentes;

v De cada vez que é armazenado um novo valor, é realizado um somatório total do *array* através do objeto [array sum];

vi O resultado do somatório é dividido por *n* para obter a média volante.

Este método implica a conversão de um sinal áudio em mensagem e necessita de uma reamostragem com o objeto [snapshot~] a uma taxa muito inferior à do sinal áudio. Consequentemente, existe um elevado grau de imprecisão temporal e a possibilidade de perdas de informação entre amostras.

**Segundo passo: deteção de movimento**

i Percebi que poderia usar o objeto [tabsend~] para escrever blocos de áudio[4], de uma só vez, diretamente no *array*;

ii Utilizei uma onda quadrada para simular o sinal de entrada, uma vez que o sinal proveniente do encoder é binário.

iii O sinal é escrito diretamente no *array* através do objeto [tabsend~], substituindo sempre os samples mais antigos pelos mais recentes. O tamanho do *array* pode ser alterado de acordo com a necessidade.

iv A uma determinada taxa (e.g. a cada 20 ms), é feito o somatório de todo o *array*. Se o valor for maior que 0, significa que há movimento.

a) No caso de o sinal (binário) parar num 1, o *array* será totalmente preenchido por 1 e o somatório será igual ao tamanho da *array* (*n*), o que resultaria num falso positivo.

b) A solução passa por calcular o módulo *n* do somatório. Assim, para $\Sigma = n$ (*array* totalmente preenchido por 1), o valor será 0.

Este método dispensa a conversão do sinal áudio em mensagem, resolvendo os problemas levantados no passo anterior. No entanto, apenas regista um *boolean* que indica se há

[4]Por norma, os programas processam áudio em blocos de samples de tamanho pré-definido (64 samples é a predefinição do Pd). É possível redefinir o tamanho do bloco para o *patch* ou *subpatch* em questão com o objeto [block~].

movimento ou não.

**Terceiro passo: registar transições e calcular médias volantes**

i Como visto anteriormente, para maior precisão, a melhor opção é trabalhar sempre em sinal.

ii Para detetar transições $0 \rightarrow 1$ / $1 \rightarrow 0$, optei por subtrair cada 2 samples consecutivos:

a) Transição $0 \rightarrow 1 = -1$

b) Transição $1 \rightarrow 0 = 1$

c) $|-1| = |1| = 1$

***Nota:*** *Como estou a aprender Pd ao mesmo tempo que programo, foi necessário fazer alguma pesquisa para perceber como implementar aquilo que no Max seria o objeto [history] dentro de um patch [gen~]*[5].

iii A solução encontrada foi o objeto [fexpr~], que permite excrever expressões semelhantes a C, para serem aplicadas sample a sample [fexpr~ $x1[0] – $x1[–1]], onde:

a) $x1[0] indica o sample atual na primeira entrada do objeto;

b) $x1[–1] indica o sample imediatamente anterior.

iv O resultado é um sinal com as seguintes características:

a) Sinal estacionário: $0 \rightarrow 0 \parallel 1 \rightarrow 1 = 0$;

b) Sinal com movimento: $0 \rightarrow 1 \parallel 1 \rightarrow 0 = \pm 1$

v Resta calcular o valor absoluto do sinal, através do objeto [abs~], para completar o primeiro ponto do algoritmo proposto. Assim o sinal será:

a) 0 quando estacionário;

b) Picos de 1 quando em movimento.

vi Agora, é possível armazenar continuamente o sinal em dois *arrays* de tamanho diferente:

a) Um *array* de curta duração, com 4410 samples (100 ms @ 44,1 kHz);

b) Um *array* de longa duração, com 220500 samples (5000 ms @ 44,1 kHz).

vii Para calcular as médias volantes e concluir o segundo ponto do algoritmo proposto, basta fazer o somatório de cada *array* e (optei por) dividir pela duração de cada *array*

[5]O [gen~] é uma extensão do ambiente de programação Max, que permite operações diretas sample a sample. O objeto [history] consiste num delay de um único sample, que permite operações entre dois samples consecutivos.

em milissegundos. Assim obtendo as duas médias volantes em número de transições por milissegundo.

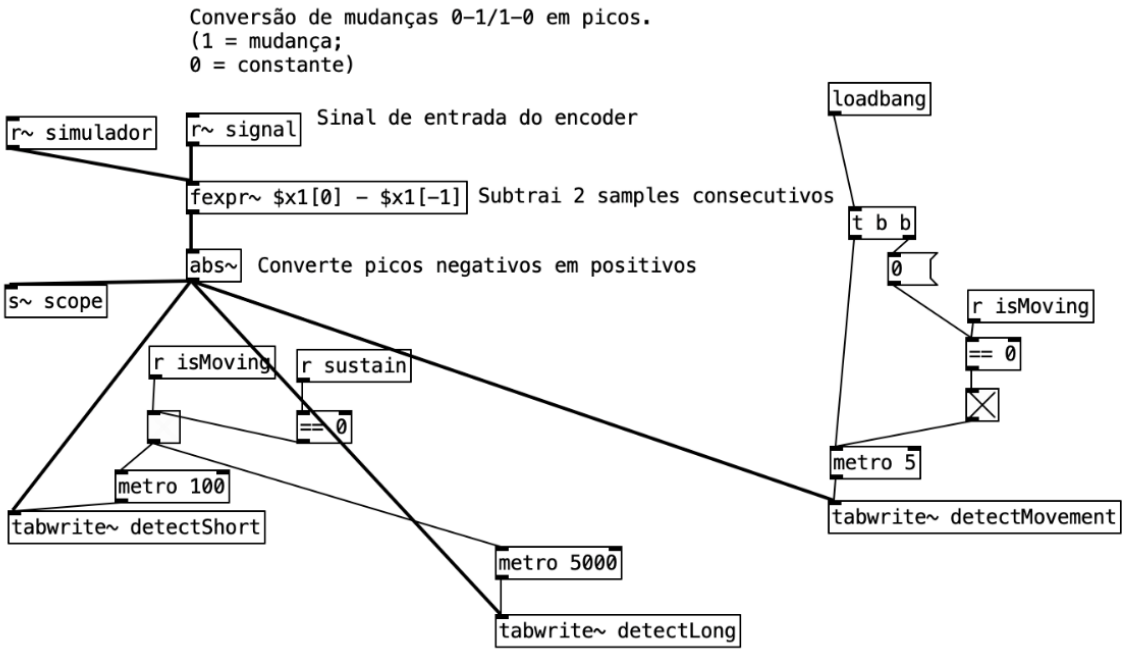



*Figura 7: Subpatch de conversão e escrita do sinal no buffer, em Pure Data.*

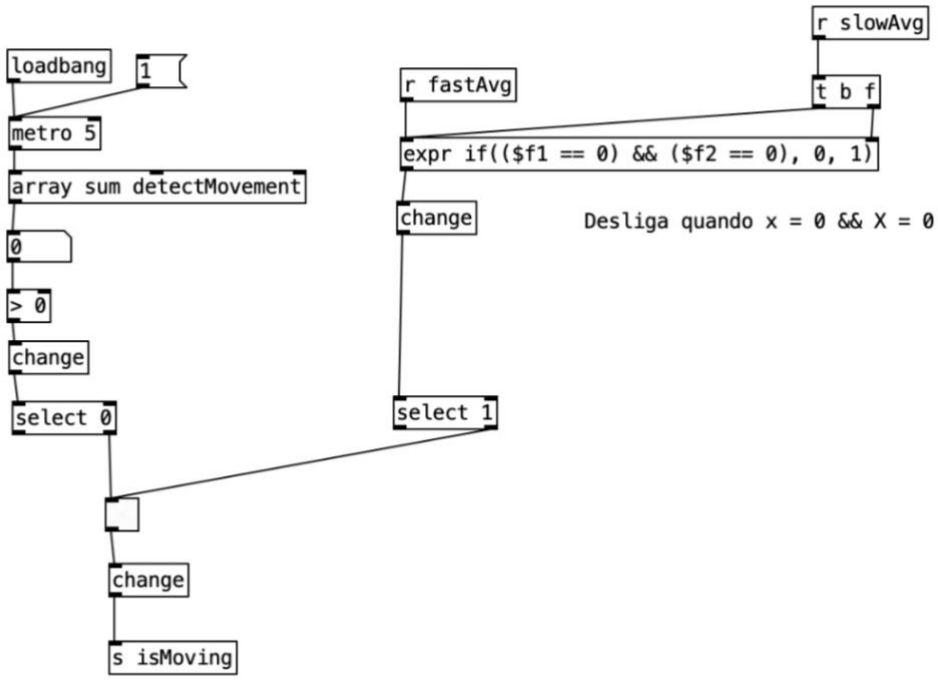



*Figura 8: Subpatch de deteção de movimento, em Pure Data.*

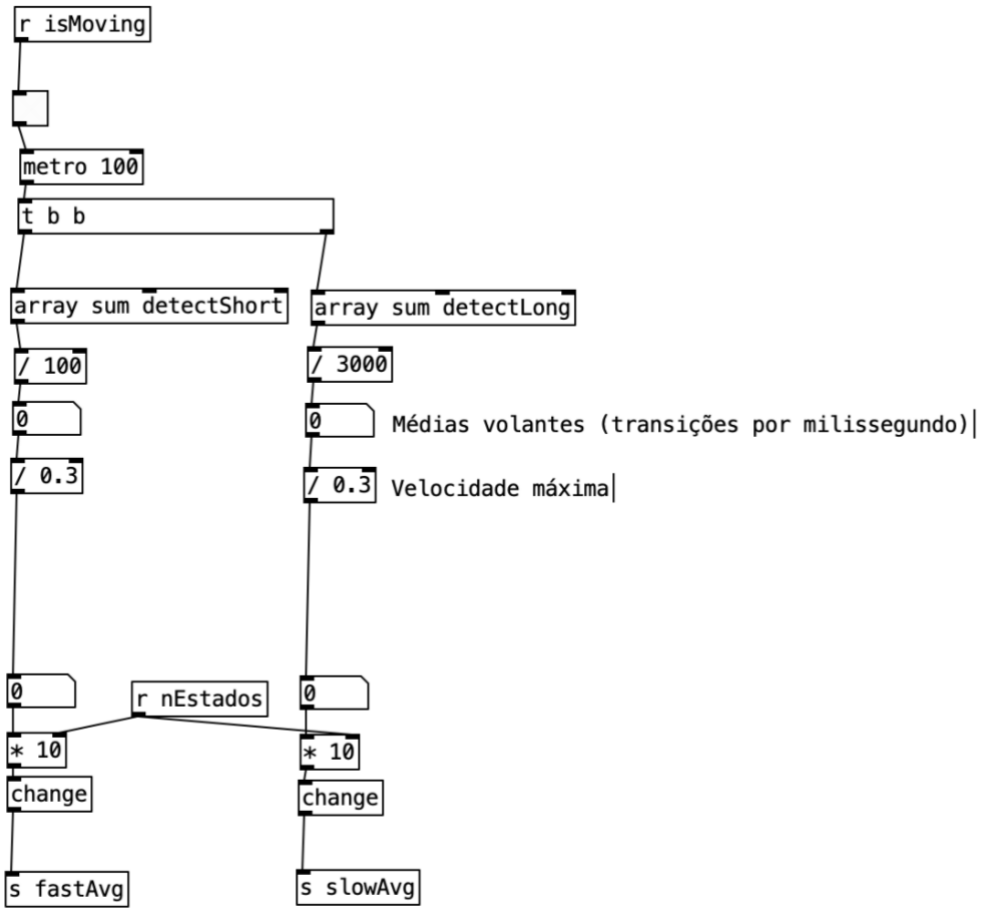



*Figura 9: Médias volantes.*

### 3.6 O primeiro protótipo

O primeiro protótipo consiste numa caixa de madeira trapezoide, com um plano inclinado face ao instrumentista, onde, numa versão futura, haveria de ser colocado um painel de controlos adicionais. De cada lado curto da caixa, existe uma manivela, ligada ao seu respetivo encoder no interior da caixa. No veio de cada encoder existe uma roda de borracha, que é pressionada e friccionada por um veio perpendicular, que serve de travão e permite regular a resistência ao movimento oferecida pela manivela.

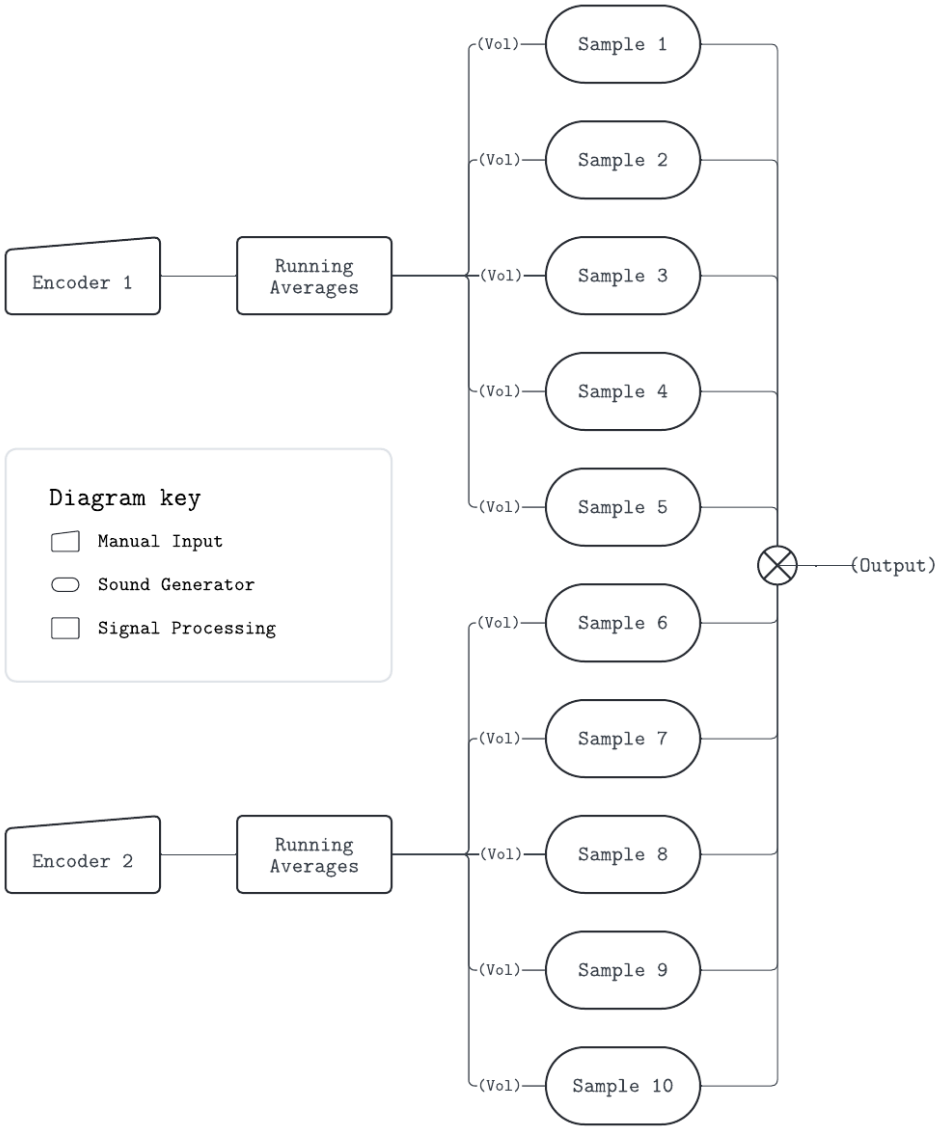



*Figura 10: Diagrama representativo do primeiro algoritmo.*

Ambos os encoders estão ligados a entradas de sinal digital do Bela. A saída de áudio digital do Bela está ligada a uma saída stereo TRS de 3.5 milímetros. O Bela é alimentado por um cabo USB ligado ao exterior do instrumento.

Quanto à produção sonora, para cada manivela foram definidos cinco níveis de intensidade, correspondentes à quantidade de camadas que são reproduzidas em simultâneo (nível 1: uma camada; nível 2: duas camadas, etc.). As camadas consistem em ficheiros de áudio pré-gerados, que passam por um filtro passa-baixo variável, cuja frequência de corte vai aumentando de acordo com a velocidade de rotação da manivela.

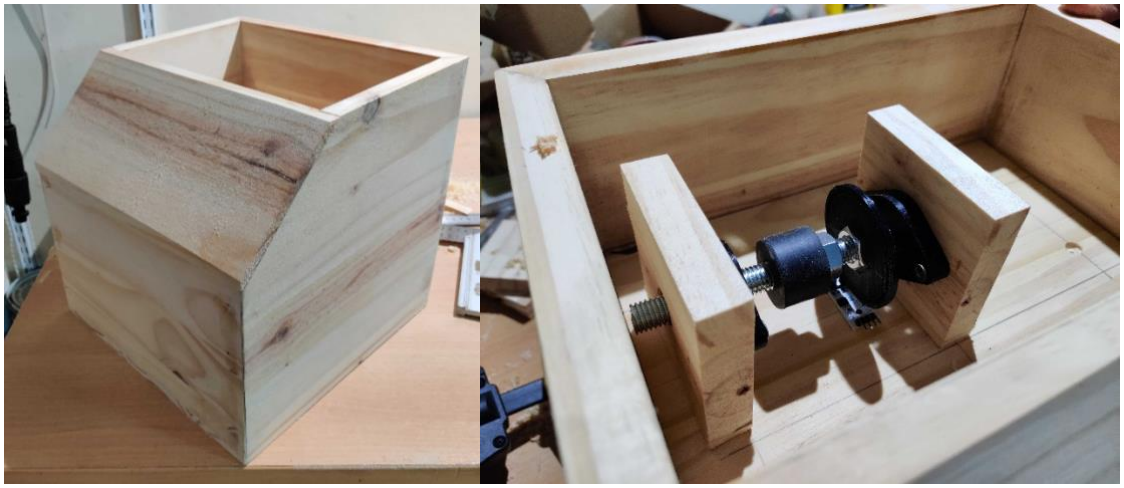


*Figura 11: Imagens do primeiro protótipo em construção.*

### 3.7 Consultoria de design de som, teste e conclusões

O meu papel no desenvolvimento do primeiro protótipo terminou com a consultoria de design de som para o instrumento. Embora o som do primeiro protótipo seja inteiramente da autoria de Ricardo Vieira, não me sendo possível descrever em detalhe o processo de decisão e explicar as escolhas envolvidas na criação dos sons, acompanhei em parte o processo, aconselhando sempre que necessário e procurando assegurar o equilíbrio e uma boa distribuição espectral entre as várias camadas. O meu papel foi mais relevante no final deste processo, tendo me sido pedido que realizasse a mistura final.

Assim que o primeiro protótipo ficou funcional, eu e o meu colega Filipe Fernandes fomos convidados participar num ensaio, na associação Somos Nós, que se dedica a apoiar "jovens adultos com dificuldades intelectuais e desenvolvimentais" (*Somos Nós*, sem data) e cujos membros iriam estrear os protótipos em concerto, sendo eles alguns dos seus destinatários finais. A sessão foi produtiva, inspiradora e cheia de surpresas. Para além de participar na sessão e assegurar que tudo funcionava corretamente, estávamos lá para observar as reações iniciais dos participantes e as interações que daí surgiriam, com o objetivo de perceber como é que o nosso público-alvo interagia intuitivamente com os instrumentos.

As reações ao tocar foram as mais variadas, desde excitação e alegria pura, a timidez e vergonha (por ser o centro das atenções naquele momento). Alguns dos participantes pareciam não se fartar de explorar e tocar os novos instrumentos, enquanto outros pareceram ficar totalmente

satisfeitos com o primeiro som produzido, passando imediatamente a vez ao próximo. Os momentos mais surpreendentes e inspiradores ocorreram durante a exploração da DigiBall.

A DigiBall consiste numa bola feita de espuma, com acelerómetro e giroscópio no seu interior, que permitem gerar som a partir do movimento da bola. Na sua versão atual, a bola em si não produz som, sendo os dados provenientes da leitura do movimento transmitidos sem fios para um computador, que os processa e a partir deles gera som. As reações à DigiBall foram desde as mais previsíveis: atirar ao ar, passar a bola de uns para os outros, rodar a bola na mão ou segurar com uma mão e bater com a outra, com as quais nós já contávamos e para as quais estávamos preparados; às mais imprevisíveis, com as quais não contávamos de todo: abanar a bola perto do ouvido (como se esta chocalhasse ou produzisse algum tipo de som), dançar passando a bola à volta do corpo, da cabeça e por entre as pernas e até mesmo dançar à John Travolta no filme *Saturday Night Fever*. Infelizmente, os sons gerados ao dançar com DigiBall não eram particularmente interessantes ou satisfatórios, o que nos levou a concluir imediatamente que teríamos de rever esse aspeto, pois aquele momento de dança tão especial, bonito e inspirador, teria de obter um resultado sonoro à sua altura.

Por outro lado, o protótipo da Caixa da Música levantou uma série de problemas à equipa da Digitópia. Reparamos que a interação com o instrumento era praticamente sempre a mesma: a pessoa que estivesse a tocar tinha, normalmente, a tendência para estabilizar numa velocidade de rotação mais ou menos constante, resultando num bordão maioritariamente estático e sempre presente. Não era este o resultado musical que estávamos à procura. O bordão sempre na mesma nota torna-se cansativo ao fim de algum tempo e tornou-se claro que o instrumento, naquela configuração, não era suficientemente estimulante para quem está a tocar. Em discussão interna, entre os membros da Digitópia, chegámos às seguintes conclusões:

i Apenas uma manivela, ou mesmo duas, torna-se demasiado limitativo e convida pouco à exploração e prática do instrumento. Traduz-se em controlar um instrumento com apenas uma ou duas variáveis que não interagem entre si. As manivelas tornam-se supérfluas, ao ter em conta que o mesmo trabalho pode ser realizado com mais controlo, utilizando somente um ou dois potenciómetros deslizantes.

ii A imutabilidade da nota do bordão também é uma limitação. Este problema poderia ser resolvido se a nota evoluísse autonomamente ao longo do tempo, ou então com a adição de um teclado que permitisse mudar a nota. Por outro lado, a adição do teclado iria contra o propósito "Universal" do projeto e uma evolução autónoma da nota

configuraria mais uma forma de retirar controlo a quem está a tocar.

iii Ao recorrer às médias volantes, o resultado sonoro ficou demasiado descorrelacionado com o movimento da manivela. Na esmagadora maioria dos instrumentos acústicos, eletroacústicos ou eletrónicos, uma alteração mecânica introduzida pelo instrumentista no instrumento resulta numa alteração imediata e proporcional no som. Não era esse o caso neste instrumento e, na minha opinião, é este o principal fator que leva a que o instrumento convide pouco à exploração musical.

iv A utilização exclusiva de samples pré-gerados e em *loop* agrava ainda mais esta descorrelação.

v Este algoritmo consome demasiados recursos para realizar a deteção de movimento, deixando pouca margem para gastar em geração e processamento de som. É necessária uma solução mais eficiente.

Foi nesta fase que o desenvolvedor principal da Caixa da Música, Ricardo Vieira, decidiu abandonar a Digitópia, cabendo-me a mim dar seguimento ao seu trabalho e desenvolver um segundo protótipo.

### 3.8 O segundo protótipo

Assumindo este projeto como meu, no meu novo papel de desenvolvedor principal, havia algumas ideias que eu precisava de clarificar para mim próprio, antes de prosseguir. Numa das reuniões em que discutimos a Caixa da Música, o coordenador Óscar Rodrigues referiu que, para ele, este primeiro protótipo da Caixa "não é um instrumento". Muito devido às razões suprarreferidas, de facto, a experiência de tocar o protótipo em quase nada se assemelha à experiência de tocar um instrumento musical. Mas, então, o que é que faz de algo um instrumento musical?

Ao trabalhar em projetos desta área, é frequente surgir a questão sobre o que é um instrumento, o que é um controlador e o que é que os distingue. Num instrumento acústico, eletroacústico ou mesmo num sintetizador exclusivamente analógico, uma resposta possível é: considera-se o instrumento como o conjunto de todos componentes mecânicos e eletrónicos que contribuem para a geração e caracterização do som. Isto pode incluir até o próprio corpo do intérprete e, em casos específicos, o espaço onde o campo sonoro se propaga. Num instrumento digital, ou pelo menos parcialmente digital, há outros aspetos a considerar: A distinção que se costuma fazer entre instrumento e controlador prende-se com a capacidade de produzir som. Um instrumento

é capaz de produzir som autonomamente (e.g. um sintetizador ou um sampler) e um controlador apenas deteta e transmite dados correspondentes a uma performance musical a um ou vários geradores de som independentes (e.g. teclado MIDI). Segundo esta definição, a DigiBall, por exemplo, não é um instrumento, mas sim um controlador, uma vez que precisa de um gerador de som externo. Mas esta definição de instrumento acaba por ser pouco útil para este propósito. Segundo esta definição, ou qualquer definição puramente objetiva, o protótipo é um instrumento musical. A minha intuição diz-me que a solução é mais profunda e mais subjetiva.

Não importa o que faz de algo um instrumento musical. Importa como conseguir que a experiência de tocar a Caixa da Música se assemelhe à de tocar os restantes instrumentos musicais.

***Nota:*** *Como já foi referido anteriormente, a baixa latência é um dos principais fatores que contribuem para a sensação de que se está, de facto, a tocar um instrumento real.*

Com esta ideia em mente, estabeleci alguns princípios orientadores para guiar as minhas decisões:

i. À semelhança de um instrumento acústico ou eletroacústico, tem de haver uma muito maior correlação entre mecânica e som.
ii. Uma vez que o objetivo continua a ser a geração de um bordão contínuo, a característica mais fácil e diretamente variável deverá ser o timbre.
iii. Ainda assim, é importante haver alguma forma de mudar e até mesmo afinar a nota do bordão, para uma maior conveniência ao tocar em conjunto com outros instrumentos.
iv. Deverá haver algum elemento com potencial melódico, para dotar o instrumento de maiores capacidades solísticas.
v. Seria importante haver alguma resposta tátil à rotação da manivela, para criar a sensação de que se está a acionar um qualquer mecanismo gerador de som (e. g. os melhores pianos digitais contêm um mecanismo de martelos, tal como um piano acústico. Estes martelos em nada contribuem para a geração de som, mas são essenciais para um toque mais realista).

#### Funcionalidade

- Uma manivela rotativa, que controla os geradores de som;
- Um sensor de toque com dois estados, que permite transpor o bordão uma quinta perfeita abaixo;

- Um sensor deslizante tátil que permite transpor continuamente uma das camadas do bordão (potencial melódico);
- Um sensor deslizante tátil que simula uma flauta de harmónicos, afinada com o bordão, e que pode ser tocada independentemente da manivela (potencial melódico);
- Quatro interruptores de alavanca que permitem ativar ou desativar as diversas camadas sonoras, potenciando a diversidade tímbrica e o seu desenvolvimento musical:
    1. Bordão Baixo (Synth 3);
    2. Bordão Voz 1 (Synth 1);
    3. Bordão Voz 2 (Synth 2);
    4. Textura de fundo: (Synth 4 e samples).
        a) Os samples da textura de fundo só são ativados se a Voz 1 também estiver ativa;
- Quatro interruptores, configurando quatro bits em binário, que permitem escolher a fundamental do bordão:

    0000: Mi ♭;
    0001: Mi;
    0010: Fá;
    0011: Fá ♯;
    0100: Sol;
    0101: Sol ♯;
    0110: Lá;
    etc.
- Um botão de *sustain*, que permite livrar a mão que aciona manivela;
- Um botão de corte do sinal de saída.

#### Mecanismo interno

Com o objetivo de obter uma resposta tátil ao movimento da manivela, decidimos criar um mecanismo interno que gerasse algum tipo de ruído e atrito irregular. Em conversa com os restantes membros da Digitópia, várias ideias surgiram para mecanismos baseados em rodas dentadas e outras possíveis melhorias à construção da Caixa. Infelizmente (como acabei por frisar aos meus colegas), os meus talentos não incluem qualquer particular habilidade para desenho, construção, operação de máquinas de corte, ou trabalhos manuais no geral. A probabilidade de ser eu a construir o segundo protótipo e ficar um trabalho bem feito era, no

mínimo, relativamente baixa. Acresce que o meu coordenador Óscar Rodrigues já tinha expressado a vontade de ter alguém especializado nesta área a colaborar com a Digitópia em projetos deste cariz. Foi assim que tive o privilégio de colaborar com Sérgio Coutinho, artista que integra coletivo portuense Berru, na elaboração do mecanismo.

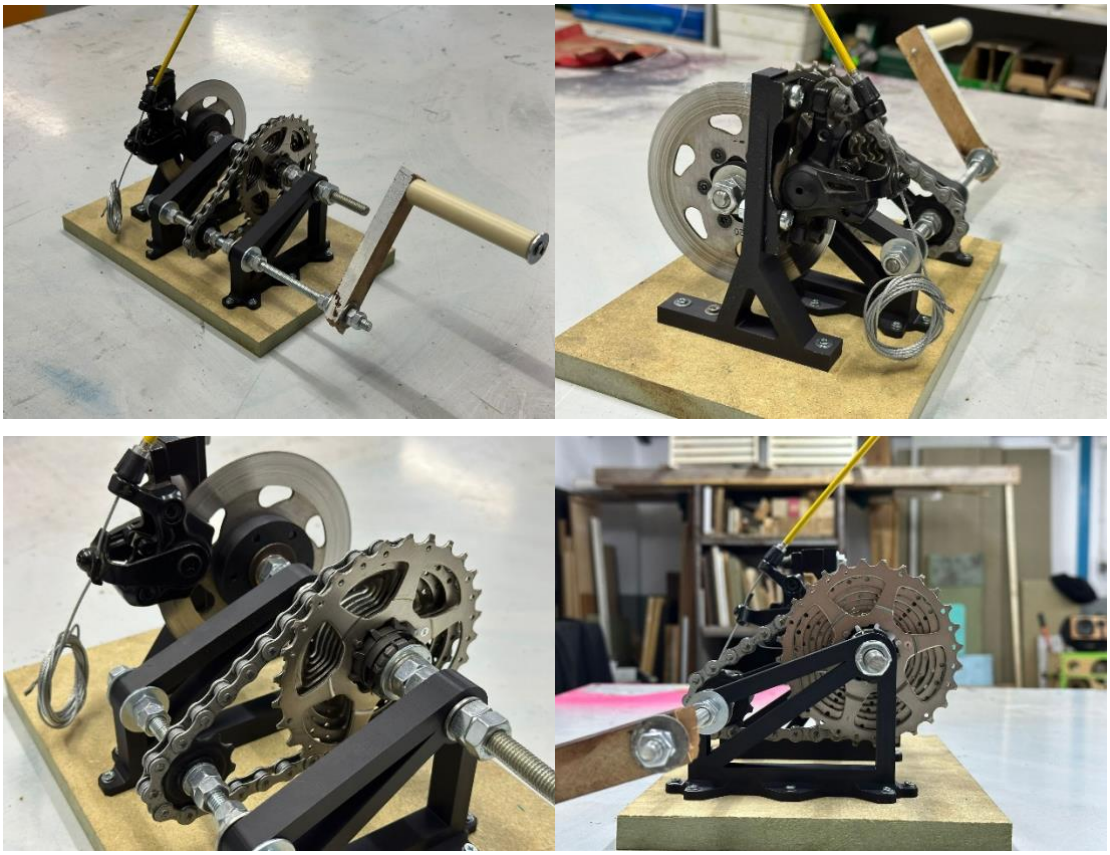


*Figura 12: Imagens do mecanismo construído por Sérgio Coutinho para o segundo protótipo.*

Por sugestão do artista Sérgio Coutinho, decidimos acrescentar um segundo eixo ao interior da caixa. Cada eixo (incluindo o da manivela) contém uma roda dentada, ambas unidas por uma corrente de bicicleta. Para além de gerar atrito irregular, isto permite utilizar o segundo eixo para acrescentar massa ao sistema e aumentar a sua inércia. Optamos também por repensar o mecanismo de travão, uma vez que o anterior era algo rudimentar (para além de inestético e deselegante). A fricção entre o veio e a borracha do travão fazia com que muitas partículas de borracha se soltassem. Para além de sujar o interior da Caixa, estas partículas acabavam por se alojar nas ranhuras do disco do encoder, interferindo com a leitura do movimento. O novo travão é inspirado pelo travão utilizado no Digi-Gurdy e consiste num simples travão de bicicleta que fricciona um disco acoplado a um dos eixos. O travão pode ser ajustado através de um cabo de aço ligado a uma cravelha no exterior do instrumento.

### 3.9 Segundo algoritmo, em SuperCollider

O SuperCollider é uma plataforma digital especializada em síntese sonora e composição algorítmica, utilizada por compositores, artistas sonoros e investigadores na área do som. Pode ser dividida em três componentes principais:

- scsynth: um servidor de áudio em tempo real, responsável por toda a geração sonora;
- sclang: uma linguagem de programação interpretada, dinâmica e orientada a objetos, com sintaxe semelhante a C, que comunica com o servidor através de OSC;
- scide: um ambiente de desenvolvimento multiplataforma que permite a interpretação e execução de excertos de código em tempo real, o que faz de sclang uma linguagem de programação ideal para *live-coding*.

Uma vez que o servidor recebe instruções via OSC, também é possível utilizar outros clientes, que não sclang, para comunicar com o servidor, bem como utilizar diversos clientes a correr simultaneamente em sistemas separados.

Optei por usar SuperCollider para desenvolver o segundo protótipo, uma vez que desconfiava que este me permitiria criar algoritmos de síntese mais complexos de forma mais eficiente (em tempo de trabalho e computação) que Pure Data, sendo que a capacidade de processamento do Bela é limitada.

***Nota:*** *Embora eu já tivesse algumas noções de SuperCollider, nunca tinha desenvolvido nada utilizando esta linguagem. Na verdade, nunca tinha desenvolvido nada em código, uma vez que estou mais habituado a ambientes de programação visuais, tal como Max e Pd.*

#### Deteção de movimento

O sinal na saída do encoder é binário e, consistindo numa onda retangular, cuja largura de pulso varia de acordo com a frequência a que as ranhuras do disco rotativo do encoder intercetam o feixe infravermelho do sensor. Consequentemente, para obter valores análogos à velocidade rotativa do movimento, basta determinar a frequência do sinal.

Isto é conseguido muito facilmente em SuperCollider com a classe ZeroCrossing.ar()[6], que mede o tempo entre cada intersecção do sinal com o eixo zero, consistindo num *pitch tracker*

[6] O método -ar significa que o código encapsulado na classe corre à taxa de amostragem da interface áudio. -ar é uma sigla para *audio rate*.

básico, mas suficiente para o propósito. Este valor é escalado de acordo com um valor de frequência máxima definido manualmente e normalizado para uma escala de zero a um, correspondendo zero à ausência total de movimento e um à velocidade máxima definida

Paralelamente, optei por definir uma variável isMoving, que traduz a ocorrência (ou não) de movimento num sinal binário, a partir de um limite de RMS (valor eficaz).

Estes dois sinais [velocidade, gate] são escritos em dois canais de barramento de controlo[7] previamente definido e armazenado na variável de ambiente ~speedBus, facilitando o seu acesso no futuro.

```
SynthDef(\encoder, {
    var sig, scale, isMoving, sustain, susSig, gate;
    sustain = ~sustainButton.kr();
    scale = ~maxSpeed.reciprocal;

    sig = DigitalIn.ar(0);
    sig = LeakDC.ar(sig);
    isMoving = RMS.ar(sig, 800) > 0.00001;

    sig = ZeroCrossing.ar(sig);
    sig = (sig.lag(0.5) * scale * isMoving).clip(0, 1);
    sig = [sig, isMoving];

    gate = isMoving + (1 - sustain);
    gate = gate > 0;

    susSig = Gate.kr(DelayN.ar(sig, 0.01), gate);

    sig = SelectX.kr(sustain.varlag(0.05), [sig, susSig]);

    Out.kr(~speedBus, sig);
}).add;
```

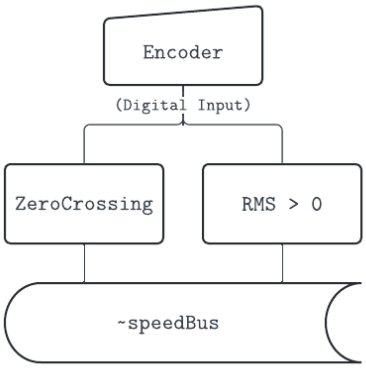



*Figura 13: SynthDef com algoritmo de deteção de movimento e diagrama representativo.*

**Simulação de fole**

Como o sistema de médias volantes anterior não era satisfatório, optei, desta vez, por seguir outra ideia. Talvez por ser instrumentista de sopros e sempre ter tido algum fascínio por órgãos de tubos e harmónios, a ideia que mais naturalmente me surgiu foi a de um fole que funcionaria exatamente como uma gaita-de-foles. Como se a manivela movimentasse um mecanismo gerador de pressão de ar, que vai enchendo um fole, e a pressão de ar dentro do fole excitasse um gerador de som. Tal como um fole real, este fole demoraria algum tempo a esvaziar e a deixar de produzir som.

[7] Para além da taxa de amostragem áudio, o servidor SC tem um segundo clock que corresponde à taxa de atualização de cada bloco de áudio (o buffer de áudio interno do SC). É explicitado pelo método -kr.

Para implementar esta ideia, comecei por definir duas variáveis posPressure e negPressure, correspondendo à pressão positiva (enche o fole) e à pressão negativa (esvazia do fole). Optei por definir um limite de velocidade, a partir do qual a pressão negativa já não tem efeito, permitindo mais facilmente encher o fole até ao máximo. Abaixo desse limite, é possível encontrar velocidades em que as pressões positivas e negativas se igualam em valor absoluto, estabilizando o som gerado.

A pressão positiva é definida como a multiplicação do valor normalizado da velocidade por uma fração da capacidade máxima do fole, definida manualmente. Quanto maior a velocidade de movimento, maior a pressão positiva e mais depressa o fole enche. Quanto maior a capacidade máxima do fole, mais tempo demora a encher totalmente.

A pressão negativa é definida como a relação entre o valor atual do fole e o seu valor máximo, multiplicada por uma constante. Como consequência, a pressão negativa é máxima quando o fole está totalmente cheio e mínima quando está praticamente vazio.

Neste caso, todos os valores são atualizados à taxa de controlo, sendo que, a cada ciclo, o novo valor do fole é o resultado da soma do seu valor anterior com a pressão positiva e a pressão negativa (evidentemente, com sinal inverso). Se a pressão positiva for superior à pressão negativa, o fole enche. Se for inferior, o fole esvazia. Se forem aproximadamente iguais, e a pressão positiva estiver abaixo do limite suprarreferido, é possível encontrar pontos de estabilidade. Acima do limite, a pressão negativa passa a ser zero e o fole enche até ao máximo.

```
SynthDef(\bellows, {                // Bellows mechanics.
    var speed, bellows, posPressure, negPressure, hasNegPressure, sustain, switch;
    speed = ~speedBus.kr(1);
    bellows = ~bellowsBus.kr();
    sustain = ~sustainButton.kr();
    switch = ~switchBus.kr(1, 3);

    posPressure = speed * ~maxBellows.reciprocal;
    hasNegPressure = (speed < ~pressureThresh);
    negPressure = bellows * ~maxBellows.reciprocal * 1.5 * hasNegPressure;

    bellows = (bellows + posPressure - negPressure).clip(0, 1);
    bellows = bellows * switch;

    bellows = Select.kr(sustain.varlag(0.05), [bellows, ~bellowsBus.kr()]);

    Out.kr(~bellowsBus, bellows);
}).add;
```

*Figura 14: SynthDef com algoritmo de simulação de fole.*

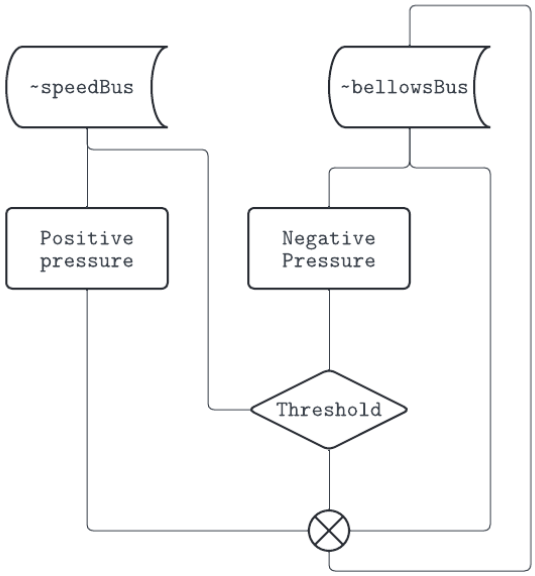



*Figura 15: Diagrama representativo do algoritmo de simulação de fole.*

***Nota:** Em sclang, os algoritmos de processamento de sinal são transmitidos ao servidor através da classe SynthDef (synth definition). O servidor contém barramentos internos (classe Bus) onde os sinais podem ser escritos, ficando acessíveis a qualquer Synth (um nó de geração/processamento de sinal no servidor, que executa o algoritmo definido por uma SynthDef). Estes barramentos podem correr à taxa de amostragem áudio (a cada sample) ou à taxa de controlo (a cada bloco de samples).*

### 3.10 Síntese digital por guias de onda

A síntese digital por guias de onda é uma extensão do algoritmo Karplus-Strong e foi desenvolvida, em grande parte, pelo professor Julius Orion Smith III, da universidade de Stanford, na década de 1980. A técnica procura simular com grande eficiência computacional a forma como as ondas sonoras se propagam nas fontes sonoras de diferentes instrumentos musicais tais como cordas, lâminas, membranas ou tubos de ar.

Nos instrumentos acústicos, o som é gerado a partir de perturbações — sinal excitador — que se propagam ao longo do material vibrante (e.g. corda ou tubo), refletindo nas extremidades e interferindo entre si de forma construtiva ou destrutiva, dependendo da frequência. Chama-se guia de onda a este meio delimitado no qual a onda se propaga, reflete e interfere consigo própria. Esta geometria pode ser representada digitalmente por linhas de *delay* recursivas (ressoadores / *comb filters*), sendo que perdas frequenciodependentes de energia podem ser representadas por filtros digitais. Também é comum recorrer a elementos não-lineares,

dependendo do instrumento que se procura simular (Smith, 2010).

O meu interesse nesta técnica não se prende com a procura por uma simulação realista de instrumentos acústicos, mas sim com o seu potencial criativo para a procura por sons não existentes.

### 3.11 Design de som

#### Synth 1: corpo

Comecei por procurar o timbre principal para o registo médio do bordão. Procurei criar alguma ligação à sanfona ao escolher um timbre de corda friccionada de onde pudesse extrair isoladamente a componente de ruído. Após algumas experiências, escolhi como ponto de partida um sample de uma nota longa de contrabaixo. O sistema corda-arco do contrabaixo é particularmente interessante para o efeito, uma vez que tem um timbre muito rico em ruído e parciais agudos. Para extrair a componente de ruído do sample, recorri à técnica demonstrada por Eric Bowman (Bowman, 2023) e utilizei o software *PaulXStretch*, que permite uma manipulação profunda do som através de *time-stretching* extremo. Ajustei os parâmetros de forma a gerar uma textura contínua, contendo apenas a componente de ruído do som original e, assim, obtive o sample que serviria de excitador para o algoritmo de síntese.

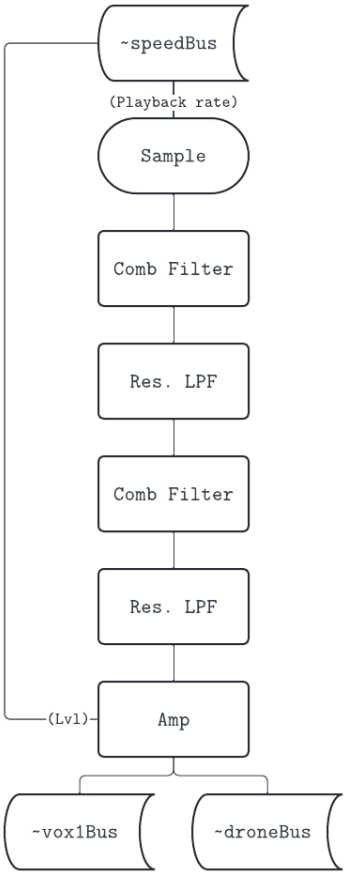



*Figura 16: Diagrama representativo do algoritmo de síntese do Synth 1.*

À medida que fui fazendo experiências com filtros e ressoadores, cheguei à conclusão que o timbre deveria variar de forma contínua e estar intimamente ligado à velocidade do movimento. Achei que seria interessante ter um som mais magro (com parciais médios-graves e graves menos intensos) quando a velocidade de rotação é lenta e a textura total é menos densa, fazendo sobressair principalmente os harmónicos agudos e assim obtendo um timbre que tem algumas características em comum com som produzido por cristal humedecido e friccionado. Quando a velocidade de rotação aumenta, o timbre fica mais encorpado, com maior intensidade de harmónicos médios-graves e menor

intensidade de harmónicos agudos. Esta modulação é conseguida através do mapeamento inverso da velocidade de rotação à velocidade de reprodução do sample, resultando num som mais agudo quando a rotação é mais lenta e, como consequência, numa maior excitação dos harmónicos agudos dos ressoadores.

Seguiram-se experiências com diversas configurações de *comb filters* e diferentes tipos de filtros ressonantes (junções de dispersão), até obter o resultado desejado. A configuração final acabou por consistir em dois conjuntos de *comb filter* e filtro passa-baixo ressonante, ligados em série, com os parâmetros ajustados a gosto.

```
SynthDef(\droneVox1, {
    var speed, freq, sig, gate, amp, formant, lambda, switch, grainGate;

    speed = ~speedBus.kr(1);
    gate = ~speedBus.kr(1, 1);
    switch = ~switchBus.kr(1, 1).lag(1);
    amp = PeakFollower.kr(speed, 0.999);

    freq = \freq.kr(110);
    freq = freq + SinOsc.kr(0.05, rrand(0,2pi), 0.002 * freq);
    lambda = freq.reciprocal;

    formant = (1 - amp * 25).clip(0.5, 30);

    sig = PlayBuf.ar(2, ~cbass.bufnum, formant, loop: 1);

    sig = sig * Linen.kr(gate, 0.05, 1, 0.05);
    sig = CombL.ar(
        in: sig,
        maxdelaytime: 1,
        delaytime: lambda,
        decaytime: 3,
        mul: 0.05
    );
    sig = BLowPass.ar(sig, 1760.clip(20,18000), 3.reciprocal);
    sig = CombL.ar(
        in: sig,
        maxdelaytime: 1,
        delaytime: lambda,
        decaytime: 4,
        mul: 0.05
    );
    sig = BLowPass.ar(sig, 880, 4.reciprocal);

    sig = sig * amp.clip(0.2, 1) * switch;
    Out.ar(~vox1Bus, sig);

    grainGate = ~bellowsBus.kr() < 1;
    sig = sig * grainGate.varlag(5);
    Out.ar(~droneBus, sig);
}).add;
```

*Figura 17: SynthDef com algoritmo de síntese do Synth 1.*

**Synth 2: presença e brilho**

O segundo componente do bordão é composto por duas camadas distintas, ambas com o objetivo de preencher a zona superior do espectro quando a velocidade de rotação é máxima e o Synth 1 ocupa a zona média-grave.

A primeira camada utiliza o mesmo sample excitador e funciona de forma semelhante ao Synth 1, sendo que a velocidade de reprodução do sample é inversamente mapeada à velocidade de rotação. No entanto, o algoritmo recorre apenas a um ressoador. O sinal começa por passar por um filtro passa-baixo muito gradual e com uma frequência de corte variável, antes de entrar no ressoador. De seguida, passa por um filtro passa-banda na zona de presença (2 kHz) e, finalmente, é novamente filtrado por um novo filtro passa-baixo, semelhante ao primeiro.

A segunda camada utiliza um sample diferente, obtido com recurso à mesma técnica que o primeiro e ao software *PaulXStretch*. O som que lhe deu origem pertence a uma Tambura: um instrumento de cordas beliscadas tradicionalmente utilizado na música clássica indiana para gerar uma textura harmónica contínua, que serve de acompanhamento a uma melodia. Possui quatro ou cinco cordas e é dedilhada, no entanto não toca material melódico e o seu papel consiste em repetir o mesmo padrão rítmico e harmónico ao longo de uma performance, criando o contexto e ambiente necessários e característicos da música clássica indiana. O instrumento tem um corpo oco, geralmente feito de madeira ou cabaça, bem como um braço, à semelhança do alaúde. As cordas são afinadas de acordo com o modo que está a ser tocado, variando ligeiramente a prática entre diferentes tradições ou escolas. A proximidade de funções entre a Tambura e a Caixa da Música (ambas geradoras de bordões), bem como o seu brilho e riqueza em parciais agudos despertaram o meu interesse neste instrumento para complementar o timbre da Caixa.

O sinal do sample é reproduzido ao dobro da velocidade para evidenciar o conteúdo agudo do espectro e passa apenas por um único ressoador, seguindo-se um filtro passa-alto com frequência de corte de 1000 Hz e, em série, um filtro passa-baixo muito gradual, com uma frequência de corte variável. Se o primeiro sensor deslizante for acionado, os ressoadores são transpostos de acordo com a posição do dedo no sensor.

Finalmente, é fundamental que o Synth 2 não se sobreponha ao Synth 1 quando a velocidade é lenta e ambos ocupam a zona média-aguda. Consequentemente, o Synth 2 é praticamente inaudível quando a velocidade de rotação é baixa, fazendo-se ouvir somente quando já não interfere com o Synth 1. Para este efeito, a amplitude e a frequência de corte dos filtros passa-

baixo são mapeadas diretamente à velocidade de rotação. Quanto menor a velocidade, menor a amplitude e mais baixa a frequência de corte; e vice-versa.

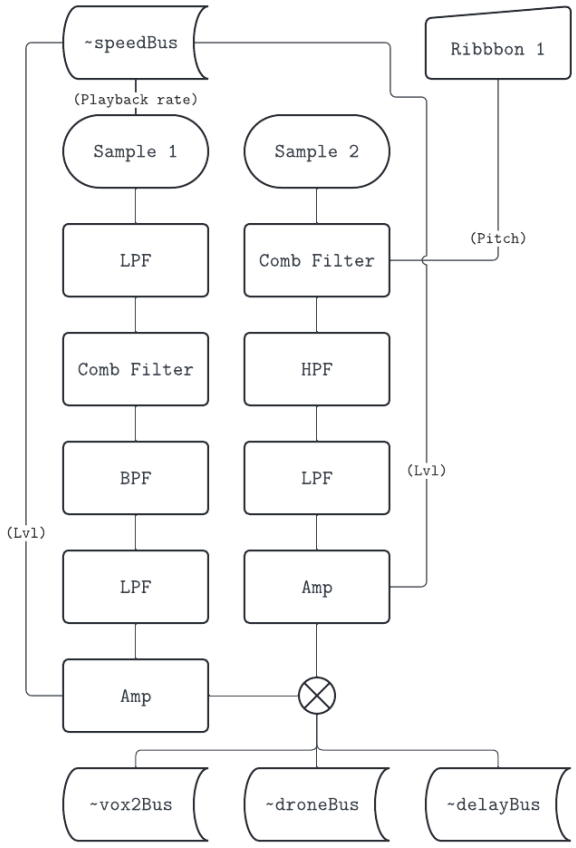



*Figura 18: Diagrama representativo do algoritmo de síntese do Synth 2.*

**Synth 3: baixo**

Assim que o caráter do bordão ficou definido, senti necessidade de adicionar um registo grave, alargando a tessitura e assim expandido as possibilidades expressivas do instrumento, principalmente no papel de acompanhador.

O algoritmo é semelhante ao do Synth 1, fazendo inclusivamente uso do mesmo sample, mas aqui é reproduzido a um quarto da velocidade, para evidenciar o registo grave. Os restantes parâmetros foram ajustados a gosto, de modo a equilibrar bem com as restantes camadas.

```
SynthDef(\droneVox2, {
    var speed, freq, sig, sig1, sig2, gate, amp, formant, lambda, switch, env, ramp, rampGate,

    speed = ~speedBus.kr(1);
    gate = ~speedBus.kr(1, 1);
    switch = ~switchBus.kr(1, 2).lag(1);
    amp = speed.lag(0.3);

    rampGate = ~ribbonBus1.kr(1);
    ramp = Env([0, 1, 0], [0.5, 2], releaseNode: 1).kr(gate: rampGate);
    ramp = ramp * 2 - 1;
    freqGate = ramp > 0;
    freq = (~ribbonBus1.kr(1, 1).linexp(0.0, 1.0, 50.midicps, 64.midicps) * freqGate)
    + (\freq.kr(110) * (1- freqGate));

    env = ramp.abs;

    freq = freq + SinOsc.kr(0.05, rrand(0,2pi), 0.007 * freq);
    lambda = freq.reciprocal;

    formant = (1 - amp * 24.5).clip(0.5, 1);

    sig1 = PlayBuf.ar(2, ~cbass.bufnum, formant, loop: 1);
    sig2 = PlayBuf.ar(2, ~tanpura.bufnum, 2, loop: 1);

    sig1 = BLowPass.ar(sig1, amp * 10000, 0.5.reciprocal);
    sig1 = CombL.ar(
        in: sig1,
        maxdelaytime: 1,
        delaytime: lambda,
        decaytime: 20,
        mul: 0.7);
    sig1 = BBandPass.ar(sig1, 2000, 1);
    sig1 = BLowPass.ar(sig1, amp * 10000, 0.1.reciprocal);

    sig2 = sig2 * Linen.kr(gate, releaseTime: 0.001);
    sig2 = CombL.ar(
        in: sig2,
        maxdelaytime: 1,
        delaytime: lambda,
        decaytime: 1,
        mul: 0.7
    );
    sig2 = BHiPass4.ar(sig2, 1000);
    sig2 = BLowPass.ar(sig2, amp * 15000, 0.5.reciprocal);

    sig1 = sig1 * 0.25;
    sig2 = sig2 * 0.2;
    sig = sig1 + sig2 * env * switch;
    Out.ar(~vox2Bus, sig);

    sig1 = sig1 * amp;
    sig2 = sig2 * amp.exprange(0.001, 1);
    sig = sig1 + sig2 * env * switch;

    grainGate = ~bellowsBus.kr() < 1;
    sig = sig * grainGate.varlag(5) * Linen.kr(gate, 0.05, 1, 0.05);
    Out.ar(~delayBus, sig * 0.3);
    Out.ar(~droneBus, sig);
}).add;
```

*Figura 19: SynthDef com algoritmo de síntese do Synth 2.*

**Synth 4**

Como o Synth 4 faz parte da textura de fundo, em vez de estar diretamente dependente da velocidade, é controlado pelo nível de pressão dentro do fole. Achei que a ideia do fole, a existir, teria de ser claramente percetível. Teria de haver uma camada com particular ênfase no registo agudo, com energia suficiente na zona de presença para se percecionar bem, sem nunca tomar o primeiro plano. Seria importante conferir a esta camada um caráter com um brilho característico.

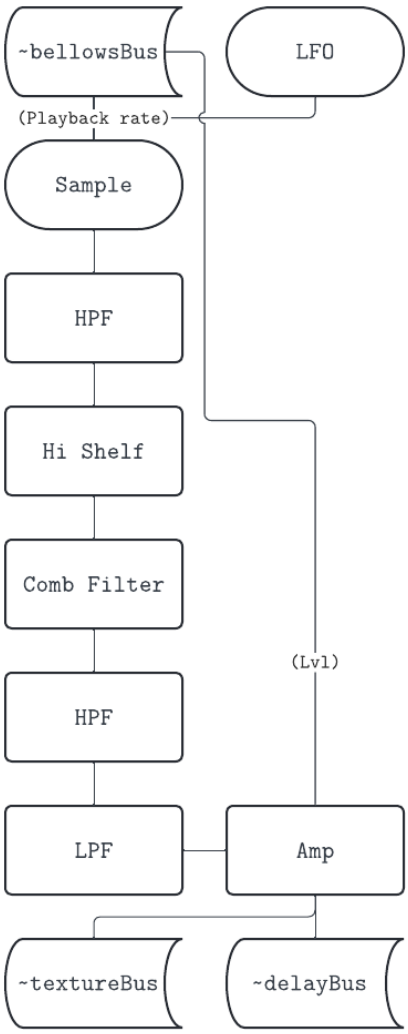



*Figura 20: Diagrama representativo do algoritmo de síntese do Synth 4.*

Comecei por imaginar uma taça tibetana friccionada, mas não conseguir encontrar nenhum sample com a rugosidade textural que procurava, tampouco tinha acesso rápido a uma para fazer experiências. É aqui que se revela o potencial da síntese digital por guias de onda: se até agora estava à procura de timbres próximos de sons reais, o som que eu estava a imaginar agora era algo como o som de rolamento metálico em movimento, com as ressonâncias e o brilho de uma taça tibetana.

Procurei por samples de sons produzidos por rolamentos e bolas metálicas em movimento, até encontrar um que fosse ao encontro das características texturais que estava à procura e, de seguida, apliquei a mesma técnica com o software *PaulXStretch*. No entanto, ao contrário do que fiz com os samples anteriores, em vez de gerar um ruído contínuo, com determinadas características tímbricas, mas estático, optei por preservar algum do movimento e dinâmica do som original.

Seguiu-se a escolha do parâmetro do algoritmo de síntese a ser modulado pelo fole. Não poderia ser a frequência, pois resultaria num som parecido com uma sirene e isso desviaria demasiado a atenção do ouvinte. A frequência deve ser estática e em relação harmónica com a fundamental.

Para que a principal característica desta camada seja o seu brilho, faz sentido que o timbre seja o principal elemento de condução musical; isto é: o elemento que mais capta a atenção do

ouvinte por ser o que mais percetivelmente varia ao longo do tempo. Novamente, a síntese por guias de onda demonstra a sua versatilidade, pois permite a transformação do timbre através da modulação da velocidade de reprodução do sample excitador, para além do tradicional recurso a filtros passa-alto, -baixo e -banda. Ao utilizar o fole para modular este parâmetro, torna-se possível seguir auditivamente a evolução do fole ao longo do tempo sem grande esforço e sem desviar demasiado a atenção.

O sinal do sample passa primeiro por um filtro passa-alto muito suave, apenas para limpar o registo grave; segue-se um filtro *high shelf*, em série, que atua acima dos 2kHz, para evidenciar presença e brilho; e só depois o sinal passa pelo ressoador. O resultado sonoro de um filtro antes ou depois do ressoador, na cadeia de sinal, é consideravelmente diferente – o efeito do filtro é mais subtil quando colocado antes do ressoador. De seguida, o sinal passa por um novo filtro passa-alto menos gradual que o anterior e, finalmente, por um filtro passa-baixo, novamente muito suave, que atua acima dos 10 kHz, uma vez que o excesso de brilho pode fazer com que o som passe para o primeiro plano.

```
SynthDef(\layer0, {
    var sig, freq, formant, amp1, amp2, switch;
    freq = \freq.kr(110);
    freq = freq.cpsmidi + 31;
    freq = freq.midicps;

    switch = ~switchBus.kr(1, 3);
    formant = ~bellowsBus.kr();
    formant = LinExp.kr(formant, 0.0, 1.0, 0.3, 30);
    formant = formant + SinOsc.kr(0.3, mul: formant * 0.5);

    // Rate increases as bellows fills up
    sig = PlayBuf.ar(1, ~ball, formant, startPos: [0.0, 0.5 * BufFrames.ir(~ball)], loop: 1);

    sig = BHiPass.ar(sig, freq/4, 0.1.reciprocal);
    sig = BHiShelf.ar(sig, 2000, 1.reciprocal, 12);
    sig = CombL.ar(sig, 1, freq.reciprocal, 0.1);
    sig = BHiPass.ar(sig, freq*2, 0.4.reciprocal);
    sig = BLowPass.ar(sig, 10000, 0.2.reciprocal);

    amp1 = LinLin.kr(formant, 0.3, 1, 10, 1).clip(1, 10);
    amp2 = ~bellowsBus.kr();

    sig = sig * 0.05 * amp1 * amp2 * switch;
    Out.ar(~delayBus, sig);
    Out.ar(~textureBus, sig);

}).add;
```

*Figura 21: SynthDef com algoritmo de síntese do Synth 4.*

**Textura de fundo (samples)**

Para as restantes camadas da textura de fundo, optei por utilizar três dos samples pré-gerados por Ricardo Vieira para o primeiro protótipo, pelo menos por agora. Os dois samples excluídos correspondiam ao som principal do bordão e à componente grave, tendo sido substituídos pelos sintetizadores Synth 1, 2 e 3. Uma vez que os valores do fole são normalizados entre zero e um, basta mapear estes valores, como pretendido, à amplitude de cada sample e às frequências de corte dos filtros passa-baixo a eles associados. A seguinte tabela mostra como estes valores foram mapeados:

| | Fole | Amplitude | Freq. corte (Hz) |
|---|---|---|---|
| Sample 1 | 0.0 – 0.5 | 0.0 – 1.0 | 100 – 20k |
| Sample 2 | 0.25 – 0.75 | 0.0 – 1.0 | 100 – 20k |
| Sample 3 | 0.5 – 1.0 | 0.0 – 1.0 | 100 – 20k |

Desta forma, à medida que o fole vai enchendo, a amplitude e frequência de corte de cada sample vai aumentando independentemente dos restantes, aumentando a densidade de camadas, bem como o conteúdo espectral de cada camada.

Esta implementação difere daquela programada por Ricardo Vieira em Pd, na medida em que a transformação do som ocorre de forma contínua, de acordo com o fole. No *patch* Pd, a evolução das camadas era feita de forma discreta, de acordo com cinco "estados" (um por cada camada), definidos a partir das médias volantes. Isto significa que a evolução de uma camada tinha de estar completa para que a seguinte fosse ativada, quando o estado seguinte fosse atingido. Nesta nova versão, o Sample 2 é ativado quando o Sample 1 ainda vai a meio da sua evolução; sendo o Sample 3 ativado quando a evolução do Sample 1 é concluída e a do Sample 2 vai a meio.

```
SynthDef(\layers, {
    var sig, l1, l2, l3, amp1, amp2, amp3, f1, f2, f3, bellows, switch1, switch2;

    bellows = ~bellowsBus.kr().lag(10);
    switch1 = ~switchBus.kr(1, 1);
    switch2 = ~switchBus.kr(1, 3);

    l1 = PlayBuf.ar(2, ~layer1.bufnum, loop: 1);
    l2 = PlayBuf.ar(2, ~layer2.bufnum, loop: 1);
    l3 = PlayBuf.ar(2, ~layer3.bufnum, loop: 1);

    amp1 = bellows.clip(0.0, 0.5).linlin(0.0, 0.5, 0.0, 1.0);
    amp2 = bellows.clip(0.25, 0.75).linlin(0.25, 0.75, 0.0, 1.0);
    amp3 = bellows.clip(0.5, 1.0).linlin(0.5, 1.0, 0.0, 1.0);

    f1 = amp1.linexp(0.0, 1.0, 100, 20000);
    f2 = amp2.linexp(0.0, 1.0, 100, 20000);
    f3 = amp3.linexp(0.0, 1.0, 100, 20000);

    l1 = LPF.ar(l1, f1);
    l2 = LPF.ar(l2, f2);
    l3 = LPF.ar(l3, f3);

    l1 = l1 * amp1;
    l2 = l2 * amp2 * 0.5;
    l3 = l3 * amp3 * 0.5;

    sig = l1 + l2 + l3 * 0.1 * switch1 * switch2;

    Out.ar(~textureBus, sig);

}).add;
```

*Figura 22: SynthDef de reprodução e mapeamento dos samples pré-gerados ao fole.*

**Synth 5: flauta de harmónicos**

Para além da manivela correspondente ao bordão, o primeiro protótipo tinha ainda uma segunda manivela cujo funcionamento era em tudo semelhante à primeira, fazendo apenas uso de samples diferentes. A ideia do meu colega Ricardo Vieira para esta segunda manivela era acionar um gerador de melodias; no entanto, como ele havia feito a opção de trabalhar exclusivamente com samples pré-gerados, isto implicaria que a melodia fosse sempre a mesma, repetida em *loop ad aeternum;* isto para mim configurava um grande problema. Para além da quase total descorrelação entre movimento e som, a ideia de um gerador de melodias semiautónomo implicaria retirar controlo ao instrumentista, o que para mim está fora de questão. Por outro lado, entendo a necessidade de ampliar o potencial melódico do instrumento.

Assim que todos os geradores de som correspondentes à primeira manivela ficaram funcionais, eu não consegui resistir à tentação de fazer uma pequena sessão de improvisação, para testar a versatilidade musical do instrumento. Para além de ficar bastante satisfeito com o resultado sonoro e musical, fiquei particularmente interessado nos harmónicos produzidos pelo Synth 1,

quando excitado a baixas velocidades. Desta vez, o som lembrou-me uma flauta de harmónicos e eu achei que este seria o complemento ideal para o instrumento.

Uma flauta de harmónicos consiste num simples tubo, normalmente feito de madeira, com uma embocadura em forma de apito. Ao contrário da generalidade das flautas, este tipo de flauta não contém orifícios para os dedos, recorrendo à variação da pressão de ar para fazer disparar diferentes harmónicos. No entanto, o orifício oposto à embocadura, na extremidade do tubo, pode ser tapado com a mão para mudar a configuração de tubo aberto para tubo fechado, permitindo ao instrumentista alternar entre duas séries de harmónicos diferentes. A flauta que escolhi como referência foi a Koncovka (Kovac, sem data), um instrumento tradicional da Eslováquia, tipicamente tocada por pastores, cuja escala se aproxima do modo Lídio.

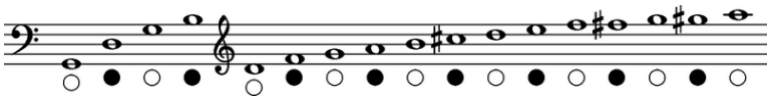


*Figura 23: Notas disponíveis numa Koncovka em sol. As notas tocadas com o tubo aberto estão marcadas com "○" e com o tubo fechado com "●".*

Decidi então que a interface tangível que controlaria este gerador de som seria um segundo sensor tátil deslizante, pois este proporcionaria uma interação muito intuitiva, sendo que uma das extremidades do sensor corresponderia ao parcial mais grave e a outra ao parcial mais agudo. Uma vez que optei por utilizar um sensor Trill Bar (*Trill touch sensors*, sem data) para prototipar, que possui funcionalidades *multitouch*, fiz corresponder a alternância entre tubo aberto e tubo fechado ao toque com um ou dois dedos, respetivamente.

Primeiro, tentei modelar a flauta recorrendo à mesma técnica utilizada anteriormente, mas os resultados não foram satisfatórios, pois os harmónicos não eram evidentes o suficiente. Uma vez que o resultado que eu procurava era tão específico e direto: duas séries de harmónicos alternantes com um número limitado de parciais, optei por recorrer à simples combinação de síntese aditiva com síntese subtrativa.

Comecei por criar dois *arrays* com as frequências correspondentes às duas séries e um parâmetro que seleciona uma ou outra de acordo com a informação *multitouch* gerada pelo Trill Bar e um banco de osciladores sinusoidais é afinado de acordo com a série selecionada através de expansão multicanal (uma funcionalidade específica do SuperCollider). Os sinais gerados são somados num único sinal que passa por uma banda de equalizador de pico altamente ressonante, afinada de acordo com a posição do dedo no sensor, o que evidencia um dos parciais do sinal mais do que todos os outros. O sinal passa finalmente por um filtro passa-alto, para

limpar as frequências abaixo da nota desejada.

```
SynthDef(\overtones, {
    var sig, env, gate, freq, pressure, closed, hs1, hs2, hs;
    freq = \freq.kr(110) * 2;

    gate = ~ribbonBus2.kr(1);

    closed = ~ribbonBus2.kr(1).clip(1,2) - 1;
    hs1 = (1..10) * freq;
    hs1 = [hs1, hs1.size -1];
    hs2 = Array.series(9, 6, 4) * freq/4;
    hs2 = [hs2, hs2.size -1];
    hs = Select.kr(closed, [hs1, hs2]).varlag(0.05);

    pressure = (~ribbonBus2.kr(1,1) * hs[1]);
    pressure = Select.kr(pressure, hs[0]).varlag(0.05);


    env = Env.adsr(0.9, 0.5, 0.5, 0.5, 1, 2).kr(gate: gate);
    sig = (SinOsc.ar(hs[0], mul: 0.3)).sum;
    sig = BPeakEQ.ar(sig, pressure, 3.reciprocal, 30, 0.02);
    sig = BHiPass4.ar(sig, pressure, 2.reciprocal,);
    //          sig = BLowPass.ar(sig, (pressure * 8).clip(2000, 20000), 2);
    sig = sig * env;
    sig = sig!2;

    Out.ar(~delayBus, sig);
    Out.ar(~overtoneBus, sig);

}).add;
```

*Figura 24: SynthDef com algoritmo de síntese do Synth 5.*

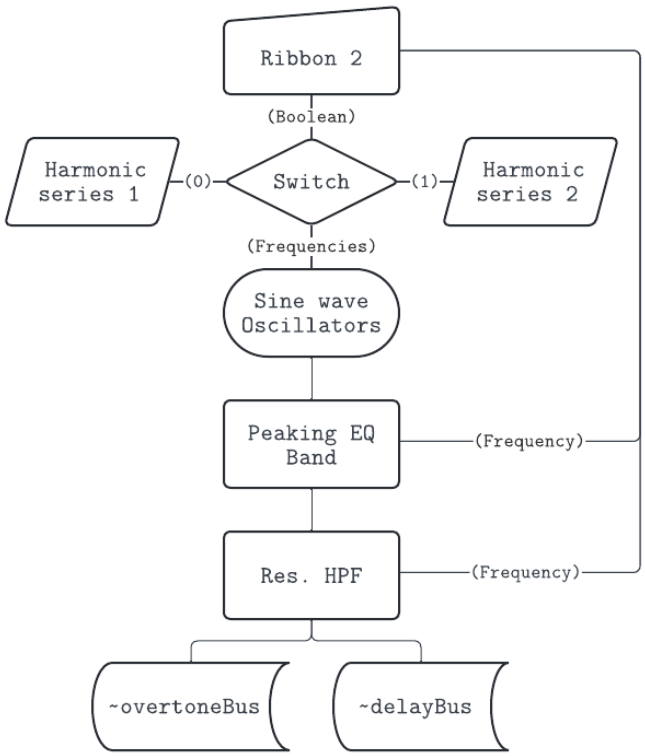



*Figura 25: Diagrama representativo do algoritmo de síntese do Synth 5.*

**Granulação em tempo real**

Atualmente, a Caixa da Música contém apenas dois efeitos de processamento de sinal: um *delay* com *feedback* — tão simples e banal que não merece explicação — e um granulador em tempo real.

O granulador em tempo real tem três razões de ser:

i De um modo geral, senti que faltava algum ritmo e textura nos sons gerados pelo instrumento;

ii Achei que o fole deveria ser mais consequente e, portanto, deveria alterar o bordão principal de alguma forma;

iii Em tempo real, porque seria importante que as alterações tímbricas causadas pelas mudanças momentâneas de velocidade continuassem a ser percetíveis.

A ideia era simples: quando o fole chega ao seu valor máximo, os sinais provenientes dos sintetizadores Synth 1 e Synth 2 vão sendo gradualmente granulados de forma independente, até não restar nada do sinal original. À medida que o fole vai esvaziando, o sinal original torna a fazer-se ouvir.

Felizmente, ao contrário do Max e do Pd, o SuperCollider tem diversas classes dedicadas à síntese granular, incluindo a classe GrainIn, dedicada à granulação em tempo real. Os grãos são disparados pela classe Dust, geradora de impulsos aleatórios, e a panorâmica é modulada por um ruído de baixa frequência.

```
SynthDef(\droneGrains, {
    var sig, grainGate;

    GrainIn.ar(
        numChannels: 2,
        trigger: Dust.kr(\grainDens.kr(40)),
        dur: \grainDur.kr(0.05),
        in: Mix.ar(In.ar(\input.kr(~vox1Bus))),
        pan: LFNoise1.kr(40),
        mul: \amp.kr(0.1),
    );

    grainGate = 1 - (~bellowsBus.kr() < 1);

    sig = sig * grainGate.varlag(5);

    Out.ar(~droneBus, sig);
}).add;
```

*Figura 26: SynthDef de granulação em tempo real, com a classe GrainIn.*

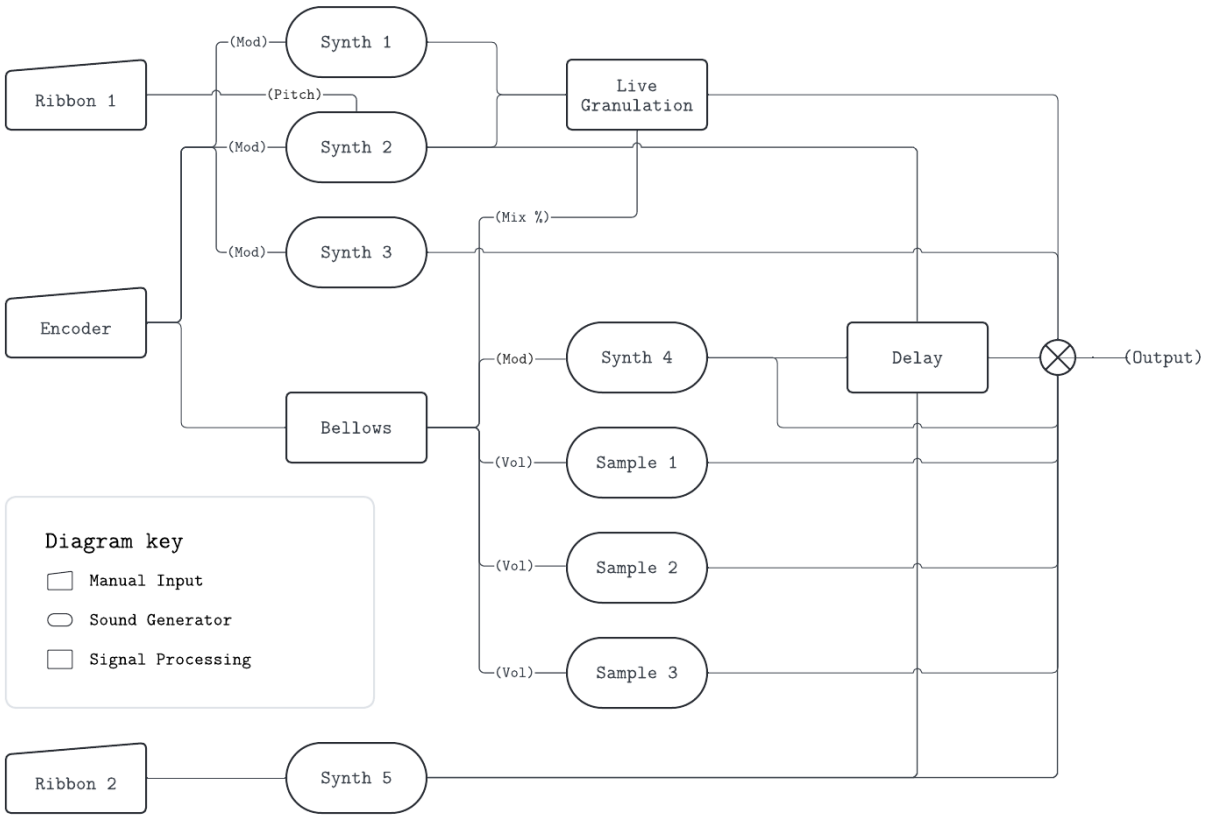



*Figura 27: Diagrama representativo do algoritmo do segundo protótipo.*

### 3.12 Conclusões e continuidade

Ainda que este seja um trabalho muito técnico, é sempre difícil (para não dizer impossível) retirar conclusões objetivas daquilo que acaba por ser um processo criativo. Talvez a única conclusão geral verdadeiramente objetiva seja que o SuperCollider é mais eficiente do que o Pure Data para este tipo de trabalho, mas esta não é uma conclusão muito interessante. As restantes conclusões têm um elevado grau de subjetividade e, em último caso, valem apenas para mim e para quem lhes quiser dar valor. No entanto, ao longo deste processo foi-me possível retirar algumas conclusões que considero importantes:

i Ao construir um instrumento musical digital, principalmente um cuja interface é invulgar, seja ele acessível, universal ou não, é fundamental testá-lo com aqueles que o vão tocar. Como vimos, os jovens da Somos Nós interagiram com a DigiBall de uma forma que foi para nós completamente inesperada e também foi através deles que percebemos as limitações da Caixa da Música.

ii A relação entre gesto e resultado sonoro não é tudo, mas é quase tudo. Para que o instrumento seja convincente e a interação natural, a energia que o músico nele injeta tem de ter uma consequência sonora claramente percetível e proporcional. Não tem necessariamente de ser óbvia, mas se a interação é física, a fisicalidade do instrumentista tem de ser ouvida.

iii A síntese digital por guias de onda é uma técnica particularmente útil quando o objetivo é dotar um som sintetizado da fisicalidade característica dos sons naturais, principalmente através do uso de um som natural como sinal excitador.

iv A existência de uma resposta tátil ao movimento ajuda a uma interação mais convincente e a uma experiência mais satisfatória para o instrumentista.

Ao momento da escrita deste relatório, o segundo protótipo ainda não está concluído. O código está plenamente funcional e a eletrónica já foi quase toda prototipada e testada. Por enquanto, os sensores táteis que estamos a utilizar são Trill Bars (capacitativos), mas há planos para serem substituídos por sensores resistivos semelhantes ao controlador *ribbon* presente no famoso sintetizador Yamaha CS-80, o que irá permitir que o sensor seja mais longo e utilize menos recursos. Falta ainda construir a caixa que alojará os componentes e montar o protótipo.

Assim que o segundo protótipo estiver concluído, haverá discussões sobre alterações a fazer para o instrumento final, havendo planos para construir cerca de 30 unidades para serem oferecidas a escolas e associações que trabalham com pessoas com deficiência.

# 4 Outras Atividades

### Ferrofone

O primeiro trabalho que realizei no estágio foi acompanhar a fase final de desenvolvimento do instrumento/instalação chamado Ferrofone: uma instalação interativa inspirada nas esculturas sonoras dos irmãos Baschet, feitas de metal e plástico industriais, que são simultaneamente esculturas, instrumentos musicais e obras de engenharia. Consiste num instrumento que permite explorar a relação entre o som, o meio em redor e o modo como a interação entre ambos afeta o nosso sentido de direção e perceção sonora (*GitHub - Digitopia/Ferrofone: An interactive percussion sound installation*, sem data).

O Ferrofone foi desenvolvido por Filipe Fernantes, sendo que a minha contribuição consistiu na programação e uma pequena parte do *patch* Pd que serve apenas para detetar quando um gesto rápido é tocado.

### Du cristal

*Du cristal* é uma peça para orquestra da compositora Kaija Saariaho, que tem uma parte escrita para o sintetizador Yamaha SY77. Assegurar a parte eletrónica dos concertos da Casa da Música faz parte das funções da Digitópia e, neste caso, era necessário instalar os sons criados pela compositora no sintetizador que iria ser utilizado no concerto. A instalação é feita a partir de mensagens MIDI *System Exclusive*, algo que era muito comum nos sintetizadores dos anos 1980 e 1990, mas que praticamente não é utilizado atualmente. Por coincidência, eu costumo trabalhar com a versão *rack* do mesmo sintetizador: o Yamaha TG77, cuja operação é praticamente igual ao SY77. Coube-me então a mim instalar os sons e acompanhar os ensaios da orquestra, para certificar que tudo estava a funcionar corretamente.

### Orelhudo!

O Orelhudo! É um projeto do Serviço Educativo da Casa da Música com o objetivo possibilitar gratuitamente uma audição ampla e plural aos alunos do primeiro ciclo do ensino básico. Consiste numa aplicação que pode ser acedida através da internet por qualquer pessoa, onde todos os dias é publicado um excerto musical relacionado com a data, juntamente com alguma informação relevante e algumas atividades sugeridas, relacionadas com o excerto. Desde janeiro de 2024 que a minha função é editar e fazer o upload dos ficheiros de áudio para o website.

### A Madrugada

*A Madrugada* é uma peça do compositor Daniel Moreira para coro, orquestra e eletrónica encomendada pela Fundação Casa da Música no contexto das celebrações dos cinquenta anos do 25 de abril. Fiquei encarregue de auxiliar o compositor com o que fosse necessário para criar a parte eletrónica da peça.

O compositor havia realizado algumas gravações de diversas pessoas a ler excertos de poemas de Sophia de Mello Breyner Andresen relacionados com a data, os quais pretendia que soassem a rádios antigos. Para isso, utilizamos dois rádios: um Blaupunkt Virgina dos anos 1950 a válvulas e um Telefunken transístor, cujo modelo era desconhecido. Ambos os rádios utilizavam uma entrada de áudio DIN de cinco pinos (atualmente caída em desuso), pelo que foi necessário procurar a especificação e soldar um adaptador XLR–DIN. Depois, bastou reproduzir os ficheiros áudio através de cada um dos rádios e gravar o resultado com um microfone Schoeps CMIT 5.

Outro dos desejos do compositor para a peça era recriar o famoso som dos passos que introduzem a *Grândola, vila morena* de José Afonso. Para isso, gravamos várias camadas de passos sobre gravilha, com diferentes tipos de calçado, para maximizar a diversidade tímbrica e criar a ilusão de um maior número de pessoas.

Posteriormente, já após o final do estágio, tive o privilégio de aconselhar o compositor sobre a espacialização e mistura multicanal, bem como acompanhar e auxiliar a operação da eletrónica nos ensaios e em concerto, juntamente com o meu coordenador Óscar Rodrigues.

## 5 Considerações finais

O trabalho realizado pela Digitópia requer um perfil polivalente com conhecimentos técnicos e musicais avançados. Ao longo do estágio, tive de cumprir vários papéis desde músico, a programador, a técnico de som. Considero que fiz um bom trabalho, tendo cumprido todos os objetivos que me foram apresentados, ainda que o trabalho na Caixa da Música não esteja concluído. O DigiUniversal é um projeto com futuro e há planos para a criação de novos instrumentos. Assim que o estágio ficou concluído, fui convidado a continuar o meu trabalho na Digitópia, tendo atualmente a cargo a direção artística de vários concertos planeados.

## 6 Referências e Bibliografia

*Agenda 2024 - Casa da Música*. (sem data). Obtido 4 de Janeiro de 2024, de https://casadamusica.com/2023/11/22/agenda-2024/

Bowman, E. (2023, Junho 12). *How To: Wooden/Overblown Flute/Shakuhachi in Vital & PaulXStretch - Synthesis Sound Design Tutorial*. https://www.youtube.com/watch?v=9a8HvBWgzDo

Dahl, S., & Bresin, R. (2001). Is the player more influenced by the auditory than the tactile feedback from the instrument? *Proceedings of the COST G-6 Conference on Digital Audio Effects*.

*Digi-Gurdy*. (sem data). Obtido 15 de Agosto de 2024, de https://digigurdy.com/

*Digitópia*. (sem data). Obtido 4 de Janeiro de 2024, de https://digitopia.casadamusica.com/

*GitHub - Digitopia/Ferrofone: An interactive percussion sound installation*. (sem data). Obtido 4 de Janeiro de 2024, de https://github.com/Digitopia/Ferrofone

Henrique, L. (2004). *Instrumentos Musicais*. Fundação Calouste Gulbenkian.

Kovac, T. (sem data). *FAQ about: Koncovka (overtone flute)*. Obtido 29 de Setembro de 2024, de https://www.fujara.sk/faq/koncovka.htm

Lago, N. P., & Kon, F. (2004). The Quest for Low Latency. *Proc. International Conference on Mathematics and Computing*.

Mäki-Patola, T., & Hämäläinen, P. (2004). Latency Tolerance for Gesture Controlled Continuous Sound Instrument without Tactile Feedback. *Proc. International Computer Music Conference*.

McPherson, A., Jack, R., & Moro, G. (2016). Action-Sound Latency: Are Our Tools Fast Enough? *Proceedings of the International Conference on New Interfaces for Musical Expression*, 20–25. https://doi.org/10.5281/zenodo.3964611

*MidiGurdy*. (sem data). Obtido 15 de Agosto de 2024, de https://midigurdy.com/

*Rotary Encoder - Northwestern Mechatronics Wiki*. (sem data). Obtido 16 de Agosto de 2024, de https://hades.mech.northwestern.edu/index.php/Rotary_Encoder

*Rotary Encoders | What is a Rotary Encoder? Rotary Encoders Explained*. (sem data). Obtido 16 de Agosto de 2024, de https://www.encoder.com/rotary-encoders

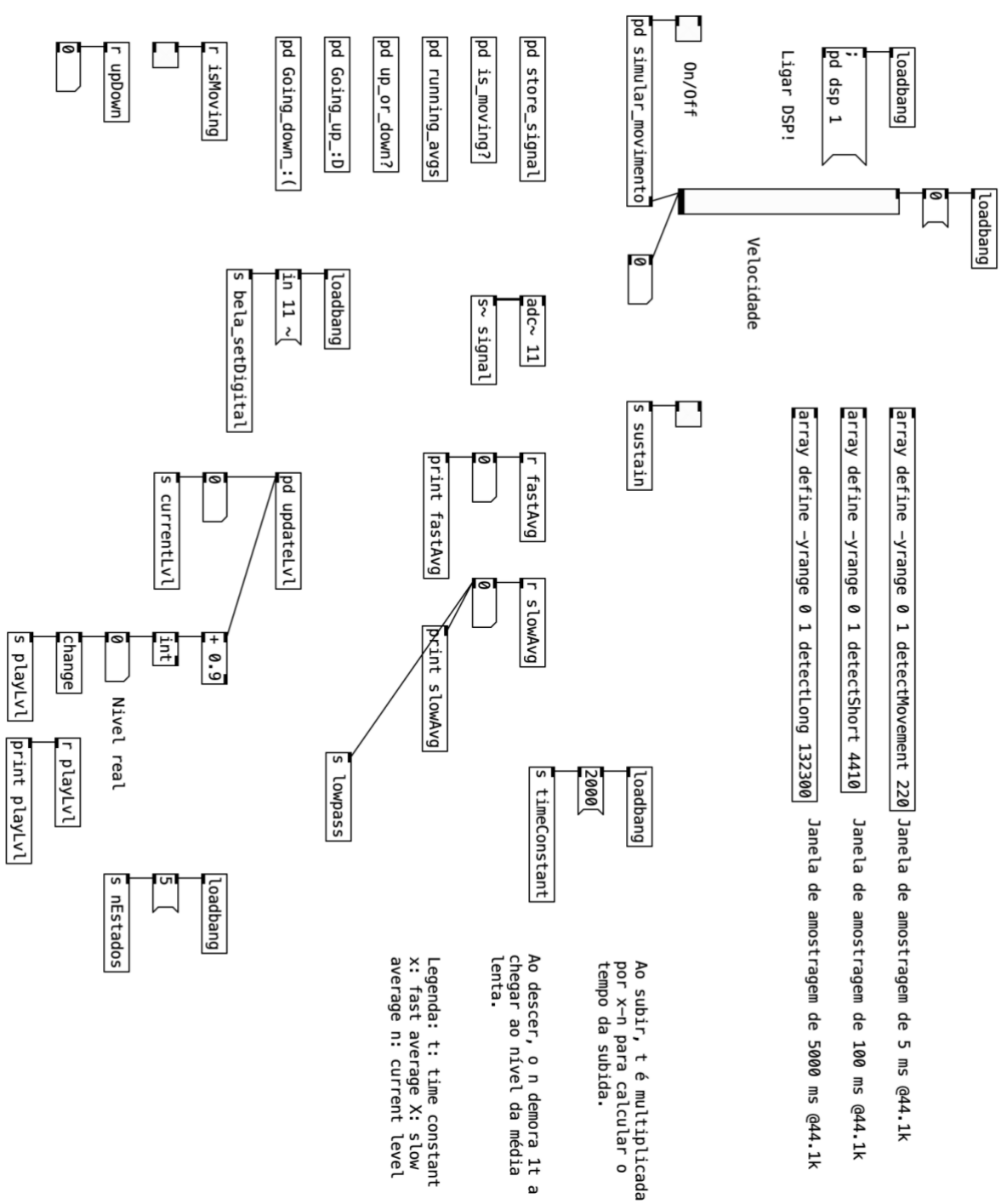
Smith, J. O. (2010). *Physical Audio Signal Processing*. W3K Publishing.

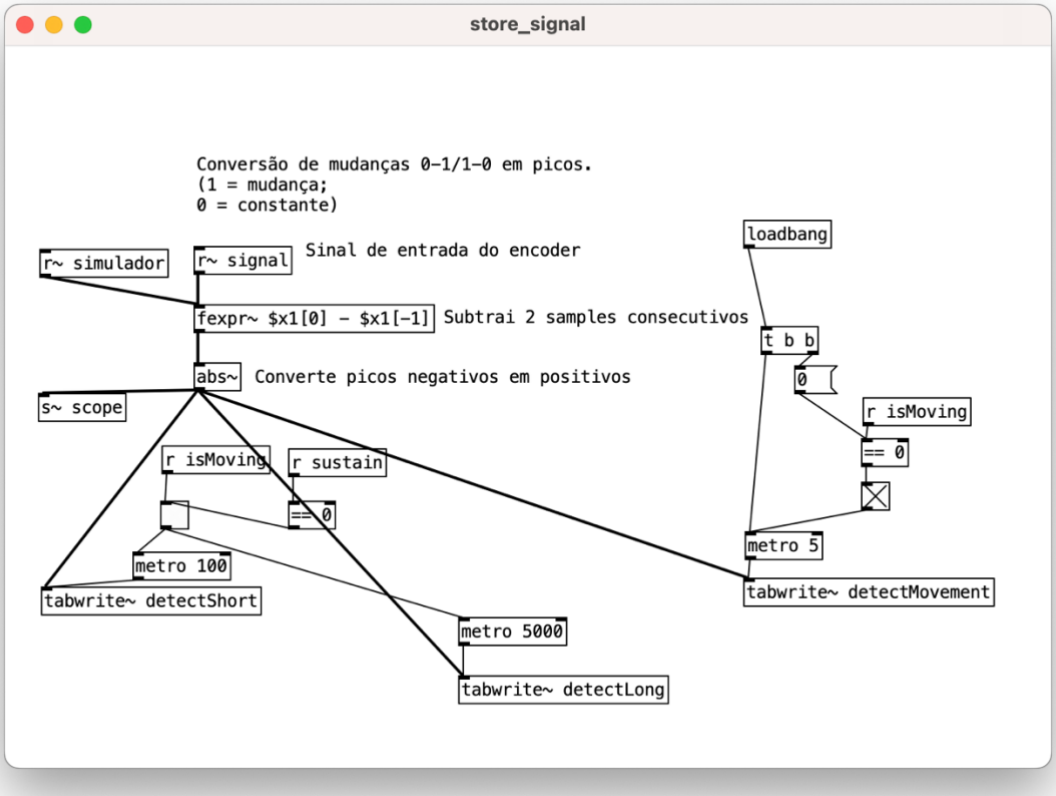
*Somos Nós*. (sem data). Obtido 4 de Janeiro de 2024, de https://somosnos.pt/

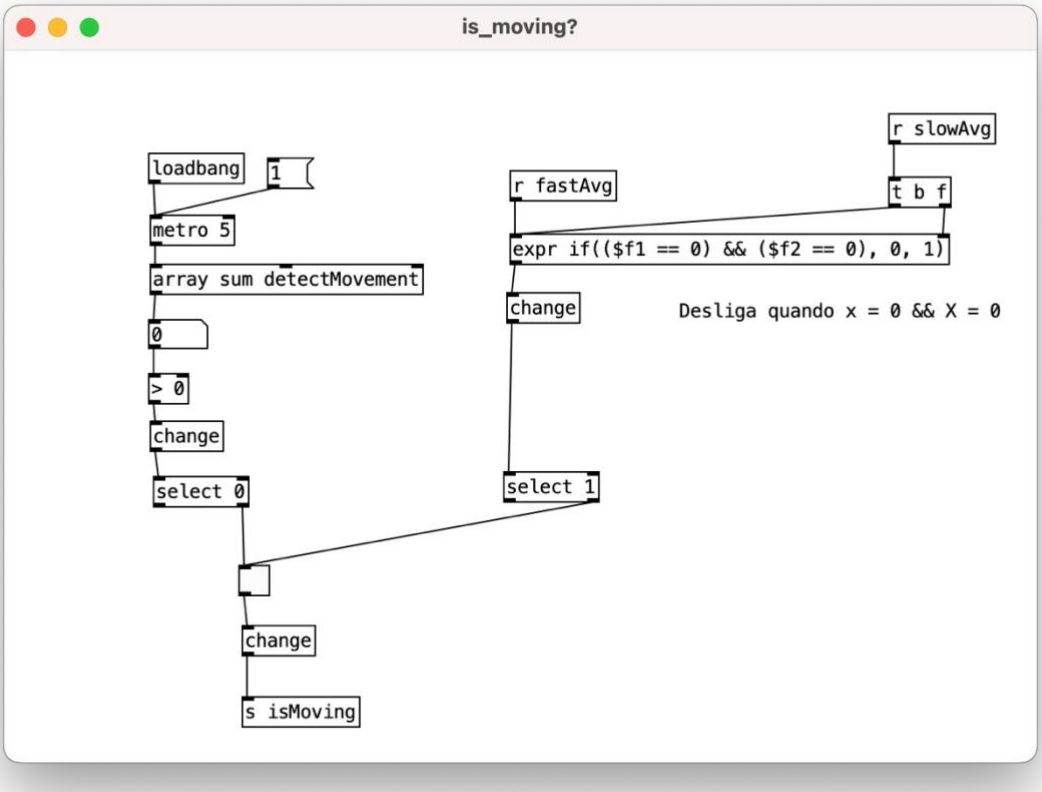
*Trill touch sensors*. (sem data). Obtido 29 de Setembro de 2024, de https://bela.io/products/trill/#types-of-trill-sensors

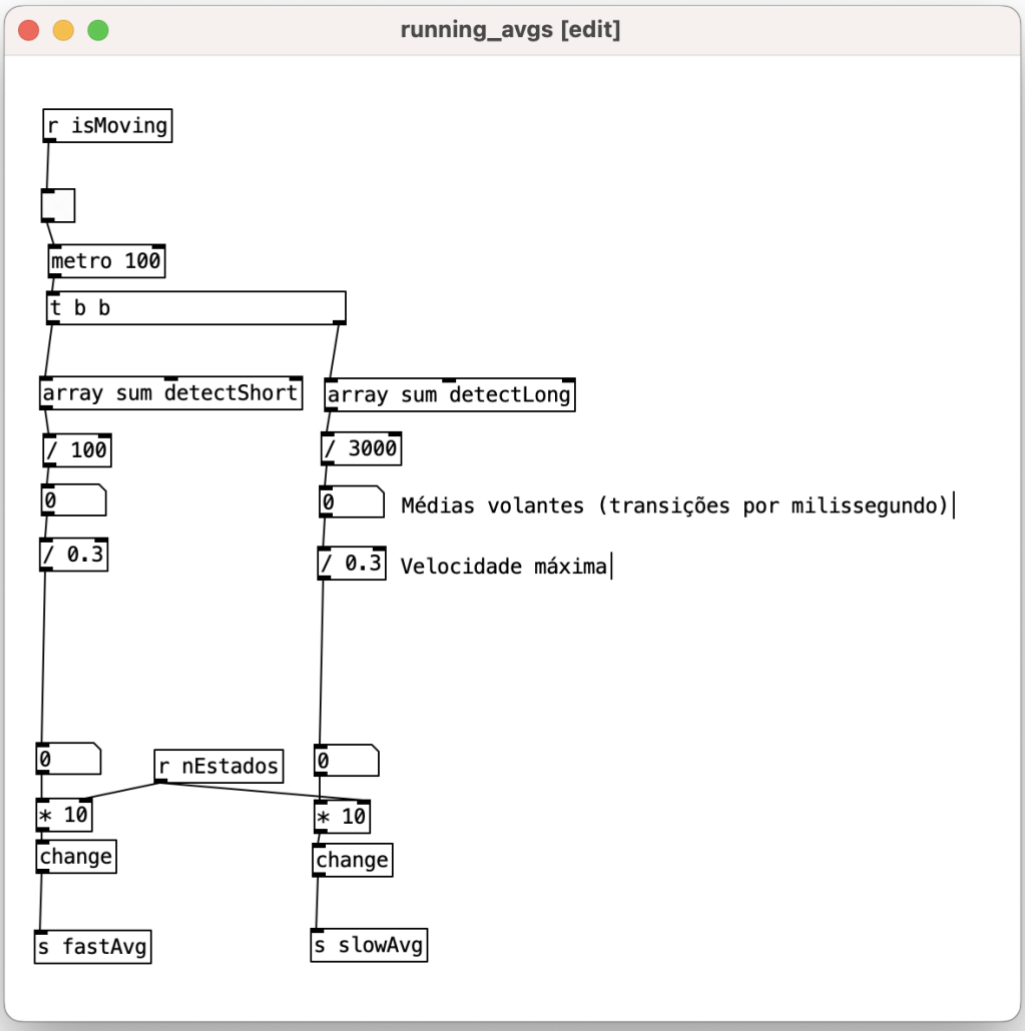
Wessel, D., & Wright, M. (2002). Problems and Prospects for Intimate Musical Control of Computers. *Computer Music Journal*, *26*(3), 11–22. http://cnmat.berkeley.edu/publications/problems-and-prospects-intimate-musical-control-computers-0

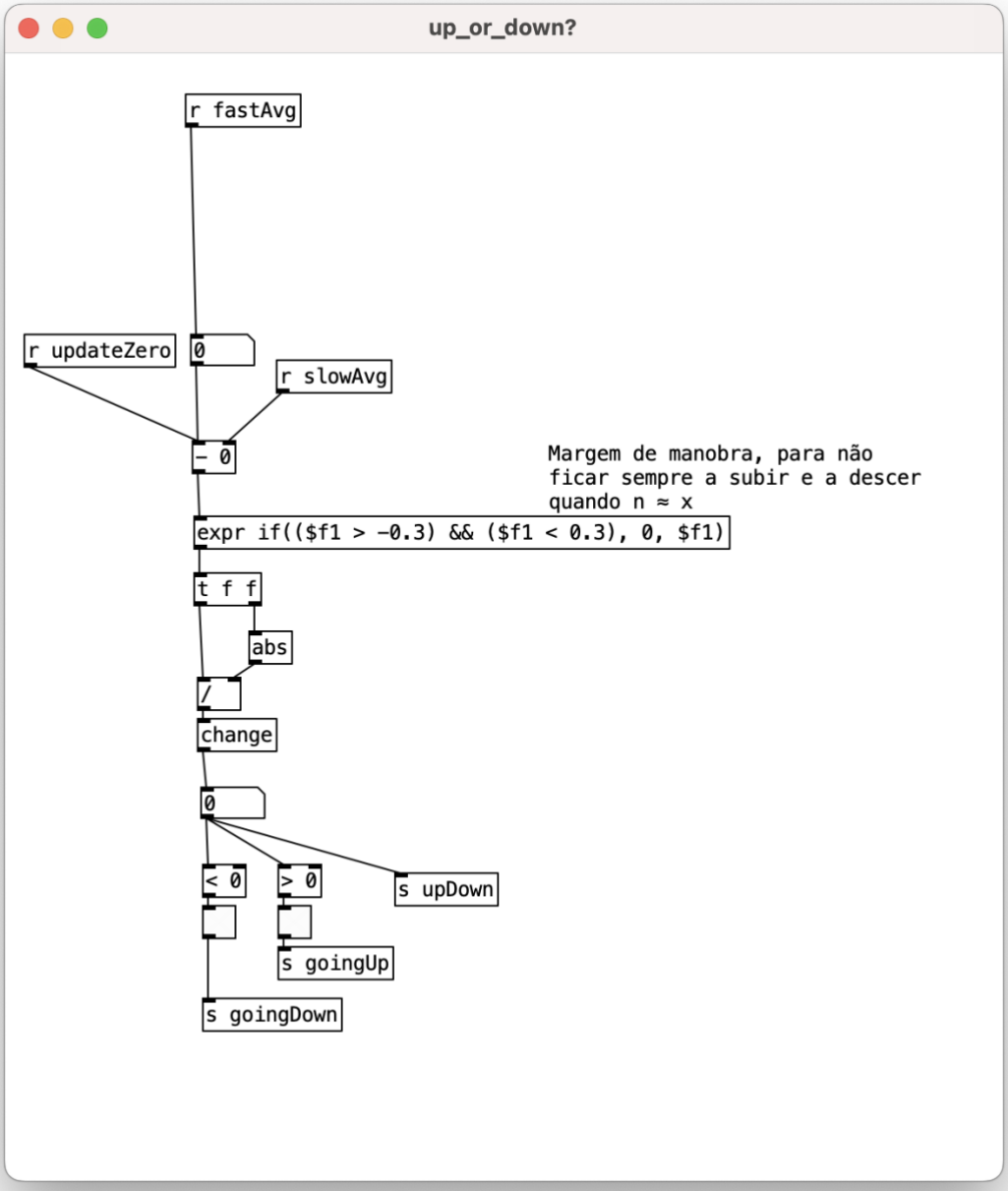
# 7 APÊNDICE A: Patch Pure Data

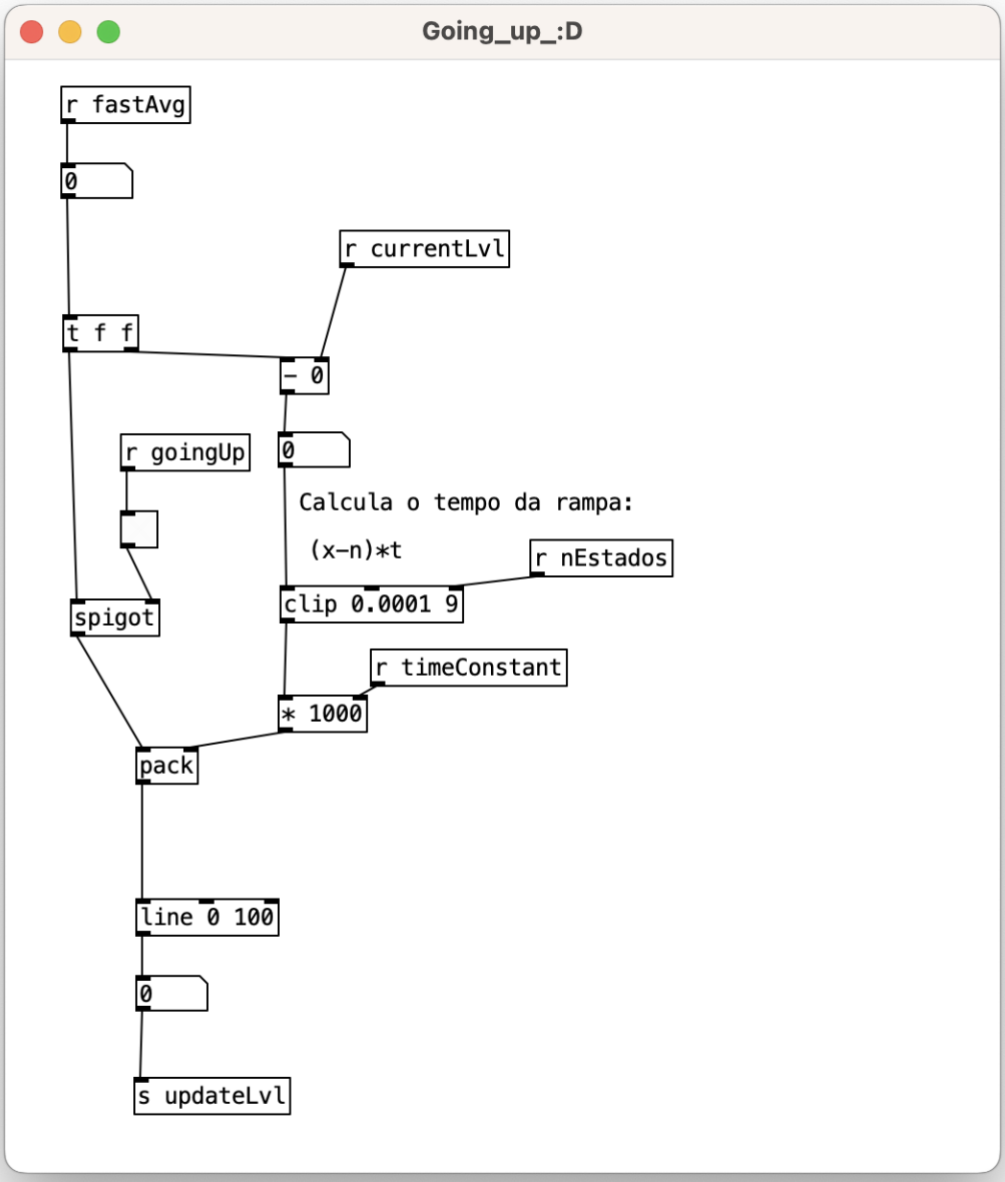


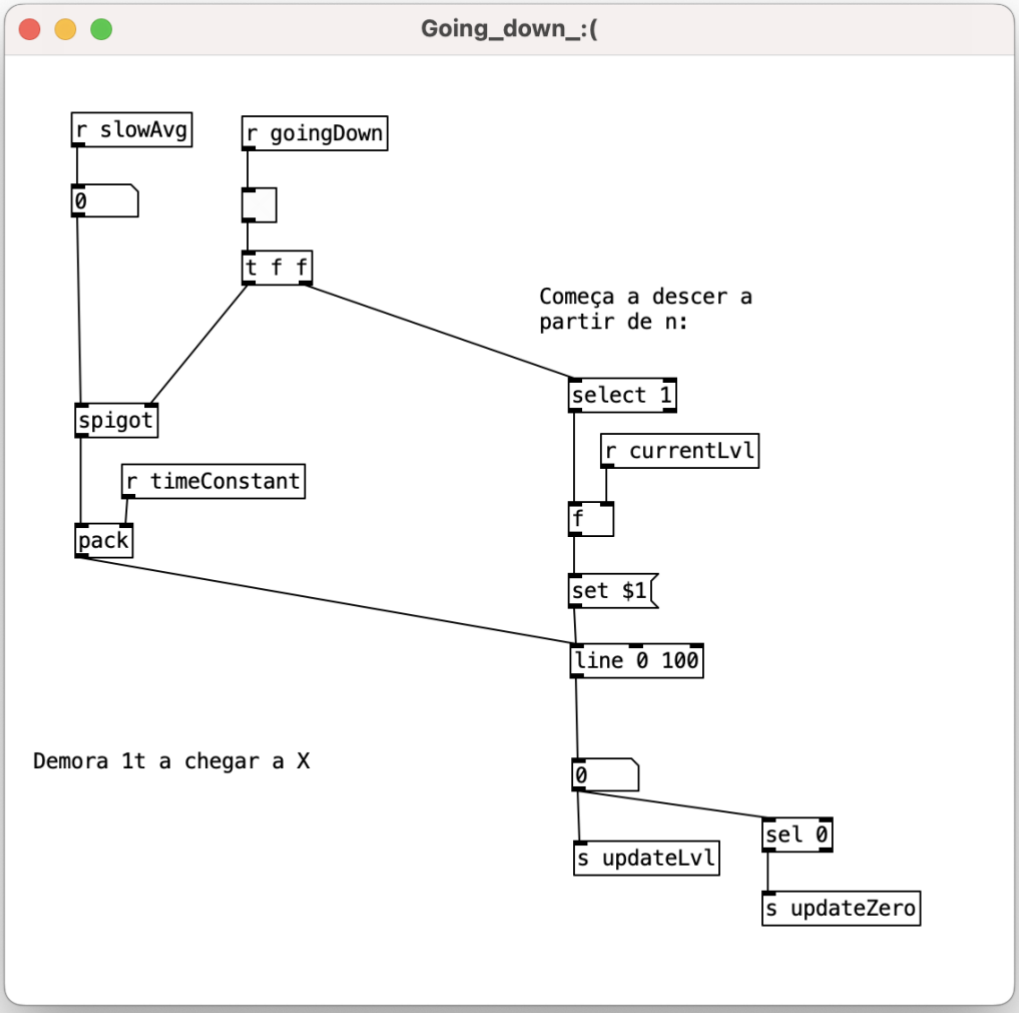

store_signal
Conversão de mudanças 0-1/1-0 em picos.
(1 = mudança;
0 = constante)
r~ simulador
r~ signal
Sinal de entrada do encoder
fexpr~ $x1[0] - $x1[-1]
Subtrai 2 samples consecutivos
abs~
Converte picos negativos em positivos
s~ scope
r isMoving
r sustain
== 0
metro 100
tabwrite~ detectShort
metro 5000
tabwrite~ detectLong
loadbang
t b b
0
r isMoving
== 0
metro 5
tabwrite~ detectMovement


is_moving?
loadbang
1
metro 5
array sum detectMovement
0
> 0
change
select 0
change
s isMoving
r fastAvg
r slowAvg
t b f
expr if(($f1 == 0) && ($f2 == 0), 0, 1)
change
Desliga quando x = 0 && X = 0
select 1


running_avgs [edit]
r isMoving
metro 100
t b b
array sum detectShort
array sum detectLong
/ 100
/ 3000
0
0
Médias volantes (transições por milissegundo)
/ 0.3
/ 0.3
Velocidade máxima
0
r nEstados
0
* 10
* 10
change
change
s fastAvg
s slowAvg


up_or_down?
r fastAvg
r updateZero
0
r slowAvg
- 0
Margem de manobra, para não
ficar sempre a subir e a descer
quando n ≈ x
expr if(($f1 > -0.3) && ($f1 < 0.3), 0, $f1)
t f f
abs
/
change
0
< 0
> 0
s upDown
s goingUp
s goingDown


Going_up_:D
r fastAvg
0
r currentLvl
t f f
- 0
r goingUp
0
Calcula o tempo da rampa:
(x-n)*t
r nEstados
spigot
clip 0.0001 9
r timeConstant
* 1000
pack
line 0 100
0
s updateLvl


Going_down_:(
r slowAvg
r goingDown
0
t f f
Começa a descer a
partir de n:
select 1
spigot
r currentLvl
r timeConstant
f
pack
set $1
line 0 100
Demora 1t a chegar a X
0
sel 0
s updateLvl
s updateZero

# 8 APÊNDICE B: Código SuperCollider

```
(
// GENERAL VARIABLES
~maxSpeed = 100;
~maxBellows = 500;
~pressureThresh = 0.3;
~root = 39;
~ribbon1Addr = 32;
~ribbon2Addr = 32;

~setup = {

	// GROUPS
	~inputGroup = Group.head(s);
	~controlGroup = Group.after(~inputGroup);
	~sourceGroup = Group.after(~controlGroup);
	~returnGroup = Group.after(~sourceGroup);
	~mixGroup = Group.tail(s);

	// AUDIO BUSSES
	s.newBusAllocators;
	~vox1Bus = Bus.audio(s, 2);
	~vox2Bus = Bus.audio(s, 2);
	~droneBus = Bus.audio(s, 2);
	~textureBus = Bus.audio(s, 2);
	~delayBus = Bus.audio(s, 2);
	~fxBus = Bus.audio(s, 2);
	~overtoneBus = Bus.audio(s, 2);

	// CONTROL BUSSES
	~speedBus = Bus.control(s, 2);
	~bellowsBus = Bus.control(s, 1);
	~switchBus = Bus.control(s, 4);
	~ribbonBus1 = Bus.control(s, 3);
	~ribbonBus2 = Bus.control(s, 3);
```

```
~sustainButton = Bus.control(s, 1);
~tuningBus = Bus.control(s, 4);

~bellowsBus.value_(0);


// DEFINITIONS: INPUTS

SynthDef(\droneSwitches, {
	var sig;
	sig = DigitalIn.kr((1..4));
	Out.kr(~switchBus, sig);
}).add;

SynthDef(\sustain, {
	var sig;
	sig = DigitalIn.kr(14);
	Out.kr(~sustainButton, sig);
}).add;

SynthDef(\ribbon, {
	var sig, sig1, gate;
	sig = TrillCentroids.kr(1, \addr.kr(32), 0.5).at([0,1,2]);

	gate = sig[0].clip(0, 2);

	sig1 = sig[1].linlin(0.1, 0.9, 0.0, 1.0).clip(0,1);
	sig1 = Gate.kr(sig1, gate);

	sig = [gate, sig1, sig[2]];
	Out.kr(\bus.kr(~ribbonBus1), sig);
}).add;

SynthDef(\tuning, {
	var tuning, drop;
```

```
	tuning = DigitalIn.kr((6..9));
	drop = DigitalIn.kr(12) * -7;

	tuning = [
		tuning[0] * 2.pow(3),
		tuning[1] * 2.pow(2),
		tuning[2] * 2.pow(1),
		tuning[3],
	].sum + 39 + drop;
	tuning.poll;
	tuning = tuning.midicps;
	SendReply.kr(Changed.kr(tuning), '/tuning', tuning);
}).add;

OSCFunc({ |msg|
	s.sendMsg('n_set', ~sourceGroup.nodeID, \freq, msg[3]);
}, '/tuning', s.addr);


// DEFINITIONS: CONTROLS

SynthDef(\encoder, {
	var sig, scale, isMoving, sustain, susSig, gate;
	sustain = ~sustainButton.kr();
	scale = ~maxSpeed.reciprocal;

	sig = DigitalIn.ar(0);
	sig = LeakDC.ar(sig);
	isMoving = RMS.ar(sig, 800) > 0.00001;

	sig = ZeroCrossing.ar(sig);
	sig = (sig.lag(0.5) * scale * isMoving).clip(0, 1);
	sig = [sig, isMoving];
```

```
            gate = isMoving + (1 - sustain);
            gate = gate > 0;

            susSig = Gate.kr(DelayN.ar(sig, 0.01), gate);

            sig = SelectX.kr(sustain.varlag(0.05), [sig, susSig]);

            Out.kr(~speedBus, sig);
      }).add;

      SynthDef(\bellows, {                        // Bellows mechanics.
            var speed, bellows, posPressure, negPressure, hasNegPressure,
sustain, switch;
            speed = ~speedBus.kr(1);
            bellows = ~bellowsBus.kr();
            sustain = ~sustainButton.kr();
            switch = ~switchBus.kr(1, 3);

            posPressure = speed * ~maxBellows.reciprocal;
            hasNegPressure = (speed < ~pressureThresh);
            negPressure = bellows * ~maxBellows.reciprocal * 1.5 *
hasNegPressure;

            bellows = (bellows + posPressure - negPressure).clip(0, 1);
            bellows = bellows * switch;

            bellows = Select.kr(sustain.varlag(0.05), [bellows,
~bellowsBus.kr()]);

            Out.kr(~bellowsBus, bellows);
      }).add;


      // DEFINITIONS: SOUND GENERATORS
```

```
	SynthDef(\droneVox1, {
		var speed, freq, sig, gate, amp, formant, lambda, switch,
grainGate;

		speed = ~speedBus.kr(1);
		gate = ~speedBus.kr(1, 1);
		switch = ~switchBus.kr(1, 1).lag(1);
		amp = PeakFollower.kr(speed, 0.999);

		freq = \freq.kr(110);
		freq = freq + SinOsc.kr(0.05, rrand(0,2pi), 0.002 * freq);
		lambda = freq.reciprocal;

		formant = (1 - amp * 25).clip(0.5, 30);

		sig = PlayBuf.ar(2, ~cbass.bufnum, formant, loop: 1);

		sig = sig * Linen.kr(gate, 0.05, 1, 0.05);
		sig = CombL.ar(
			in: sig,
			maxdelaytime: 1,
			delaytime: lambda,
			decaytime: 3,
			mul: 0.05
		);
		sig = BLowPass.ar(sig, 1760.clip(20,18000), 3.reciprocal);
		sig = CombL.ar(
			in: sig,
			maxdelaytime: 1,
			delaytime: lambda,
			decaytime: 4,
			mul: 0.05
		);
		sig = BLowPass.ar(sig, 880, 4.reciprocal);
```

```
            sig = sig * amp.clip(0.2, 1) * switch;
            Out.ar(~vox1Bus, sig);

            grainGate = ~bellowsBus.kr() < 1;
            sig = sig * grainGate.varlag(5);
            Out.ar(~droneBus, sig);
      }).add;

      SynthDef(\droneVox2, {
            var speed, freq, sig, sig1, sig2, gate, amp, formant, lambda,
switch, env, ramp, rampGate, freqGate, grainGate;

            speed = ~speedBus.kr(1);
            gate = ~speedBus.kr(1, 1);
            switch = ~switchBus.kr(1, 2).lag(1);
            amp = speed.lag(0.3);

            rampGate = ~ribbonBus1.kr(1);
            ramp = Env([0, 1, 0], [0.5, 2], releaseNode: 1).kr(gate:
rampGate);
            ramp = ramp * 2 - 1;
            freqGate = ramp > 0;
            freq = (~ribbonBus1.kr(1, 1).linexp(0.0, 1.0, 50.midicps,
64.midicps) * freqGate)
            + (\freq.kr(110) * (1- freqGate));

            env = ramp.abs;

            freq = freq + SinOsc.kr(0.05, rrand(0,2pi), 0.007 * freq);
            lambda = freq.reciprocal;

            formant = (1 - amp * 24.5).clip(0.5, 1);

            sig1 = PlayBuf.ar(2, ~cbass.bufnum, formant, loop: 1);
            sig2 = PlayBuf.ar(2, ~tanpura.bufnum, 2, loop: 1);
```

```
            sig1 = BLowPass.ar(sig1, amp * 10000, 0.5.reciprocal);
            sig1 = CombL.ar(
                  in: sig1,
                  maxdelaytime: 1,
                  delaytime: lambda,
                  decaytime: 20,
                  mul: 0.7);
            sig1 = BBandPass.ar(sig1, 2000, 1);
            sig1 = BLowPass.ar(sig1, amp * 10000, 0.1.reciprocal);

            sig2 = sig2 * Linen.kr(gate, releaseTime: 0.001);
            sig2 = CombL.ar(
                  in: sig2,
                  maxdelaytime: 1,
                  delaytime: lambda,
                  decaytime: 1,
                  mul: 0.7
            );
            sig2 = BHiPass4.ar(sig2, 1000);
            sig2 = BLowPass.ar(sig2, amp * 15000, 0.5.reciprocal);

            sig1 = sig1 * 0.25;
            sig2 = sig2 * 0.2;
            sig = sig1 + sig2 * env * switch;
            Out.ar(~vox2Bus, sig);

            sig1 = sig1 * amp;
            sig2 = sig2 * amp.exprange(0.001, 1);
            sig = sig1 + sig2 * env * switch;

            grainGate = ~bellowsBus.kr() < 1;
            sig = sig * grainGate.varlag(5) * Linen.kr(gate, 0.05, 1,
0.05);
            Out.ar(~delayBus, sig * 0.3);
```

```
            Out.ar(~droneBus, sig);
      }).add;


      SynthDef(\bass, {
            var speed, gate, amp, freq, lambda, sig, switch;

            speed = ~speedBus.kr(1);
            gate = ~speedBus.kr(1, 1);
            switch = ~switchBus.kr(1).lag(1);
            amp = speed.lag(0.5);
            freq = \freq.kr(110);

            lambda = freq.reciprocal;

            sig = PlayBuf.ar(2, ~cbass.bufnum, 0.25, loop: 1);
            sig = sig * Linen.kr(gate, 0.7, 1, 0.01);

            sig = CombL.ar(
                  in: sig,
                  maxdelaytime: 1,
                  delaytime: lambda,
                  decaytime: -5,
                  mul: 0.5
            );
            sig = BLowPass.ar(sig, freq * 32, 6.reciprocal);
            sig = CombL.ar(
                  in: sig,
                  maxdelaytime: 1,
                  delaytime: lambda,
                  decaytime: 4,
                  mul: 0.1
            );
            sig = BLowPass.ar(sig, freq * 8 + (amp * freq * 8),
3.reciprocal);
```

```
            sig = sig * amp.clip(0.1, 1) * 2.5 * switch;

            Out.ar(~droneBus, sig);
            Out.ar(~delayBus, sig);
      }).add;

      SynthDef(\layer0, {
            var sig, freq, formant, amp1, amp2, switch;
            freq = \freq.kr(110);
            freq = freq.cpsmidi + 31;
            freq = freq.midicps;

            switch = ~switchBus.kr(1, 3);
            formant = ~bellowsBus.kr();
            formant = LinExp.kr(formant, 0.0, 1.0, 0.3, 30);
            formant = formant + SinOsc.kr(0.3, mul: formant * 0.5);

            // Rate increases as bellows fills up
            sig = PlayBuf.ar(1, ~ball, formant, startPos: [0.0, 0.5 *
BufFrames.ir(~ball)], loop: 1);

            sig = BHiPass.ar(sig, freq/4, 0.1.reciprocal);
            sig = BHiShelf.ar(sig, 2000, 1.reciprocal, 12);
            sig = CombL.ar(sig, 1, freq.reciprocal, 0.1);
            sig = BHiPass.ar(sig, freq*2, 0.4.reciprocal);
            sig = BLowPass.ar(sig, 10000, 0.2.reciprocal);

            amp1 = LinLin.kr(formant, 0.3, 1, 10, 1).clip(1, 10);
            amp2 = ~bellowsBus.kr();

            sig = sig * 0.05 * amp1 * amp2 * switch;
            Out.ar(~delayBus, sig);
            Out.ar(~textureBus, sig);
```

```
	}).add;

	SynthDef(\layers, {
		var sig, l1, l2, l3, amp1, amp2, amp3, f1, f2, f3, bellows,
switch1, switch2;

		bellows = ~bellowsBus.kr().lag(10);
		switch1 = ~switchBus.kr(1, 1);
		switch2 = ~switchBus.kr(1, 3);

		l1 = PlayBuf.ar(2, ~layer1.bufnum, loop: 1);
		l2 = PlayBuf.ar(2, ~layer2.bufnum, loop: 1);
		l3 = PlayBuf.ar(2, ~layer3.bufnum, loop: 1);

		amp1 = bellows.clip(0.0, 0.5).linlin(0.0, 0.5, 0.0, 1.0);
		amp2 = bellows.clip(0.25, 0.75).linlin(0.25, 0.75, 0.0, 1.0);
		amp3 = bellows.clip(0.5, 1.0).linlin(0.5, 1.0, 0.0, 1.0);

		f1 = amp1.linexp(0.0, 1.0, 100, 20000);
		f2 = amp2.linexp(0.0, 1.0, 100, 20000);
		f3 = amp3.linexp(0.0, 1.0, 100, 20000);

		l1 = LPF.ar(l1, f1);
		l2 = LPF.ar(l2, f2);
		l3 = LPF.ar(l3, f3);

		l1 = l1 * amp1;
		l2 = l2 * amp2 * 0.5;
		l3 = l3 * amp3 * 0.5;

		sig = l1 + l2 + l3 * 0.1 * switch1 * switch2;

		Out.ar(~textureBus, sig);

	}).add;
```

```
SynthDef(\overtones, {
	var sig, env, gate, freq, pressure, closed, hs1, hs2, hs;
	freq = \freq.kr(110) * 2;

	gate = ~ribbonBus2.kr(1);

	closed = ~ribbonBus2.kr(1).clip(1,2) - 1;
	hs1 = (1..10) * freq;
	hs1 = [hs1, hs1.size -1];
	hs2 = Array.series(9, 6, 4) * freq/4;
	hs2 = [hs2, hs2.size -1];
	hs = Select.kr(closed, [hs1, hs2]).varlag(0.05);

	pressure = (~ribbonBus2.kr(1,1) * hs[1]);
	pressure = Select.kr(pressure, hs[0]).varlag(0.05);



	env = Env.adsr(0.9, 0.5, 0.5, 0.5, 1, 2).kr(gate: gate);
	sig = (SinOsc.ar(hs[0], mul: 0.3)).sum;
	sig = BPeakEQ.ar(sig, pressure, 3.reciprocal, 30, 0.02);
	sig = BHiPass4.ar(sig, pressure, 2.reciprocal,);
	sig = sig * env;
	sig = sig!2;

	Out.ar(~delayBus, sig);
	Out.ar(~overtoneBus, sig);

}).add;

// DEFINITIONS: FX RETURNS

SynthDef(\droneGrains, {
	var sig, grainGate;
```

```
	GrainIn.ar(
		numChannels: 2,
		trigger: Dust.kr(\grainDens.kr(40)),
		dur: \grainDur.kr(0.05),
		in: Mix.ar(In.ar(\input.kr(~vox1Bus))),
		pan: LFNoise1.kr(40),
		mul: \amp.kr(0.1),
	);

	grainGate = 1 - (~bellowsBus.kr() < 1);

	sig = sig * grainGate.varlag(5);

	Out.ar(~droneBus, sig);
}).add;

SynthDef(\fbDelay, {
	var sig, fb;
	sig = In.ar(~delayBus, 2);
	fb = LocalIn.ar(2);
	sig = sig + fb;
	sig = DelayN.ar(sig, 0.5, 0.5, -6.dbamp);
	sig = SineShaper.ar(sig, 0.2);
	sig = BBandPass.ar(sig, 630, 6);
	LocalOut.ar(sig);
	Out.ar(~fxBus,sig)
}).add;

// DEFINITIONS: BUS PROCESSING
SynthDef(\droneMix, {
	var sig, cut;
	cut = 1 - DigitalIn.kr(15);

	sig = Mix.ar([
		~droneBus.ar() * 0.3,
```

```
                ~textureBus.ar(),
                ~fxBus.ar(),
                ~overtoneBus.ar()
            ]);
            sig = BHiPass.ar(sig, 100);
            sig = Compander.ar(sig, sig, -16.dbamp, 1, 1.5.reciprocal,);

            sig = sig * cut;

            Out.ar(0, sig)
        }).add;

};

// BUFFERS
s.waitForBoot({
        ServerTree.removeAll;
        Buffer.freeAll;
        Server.freeAll(evenRemote: true);

        s.sync;
        ~cbass = Buffer.read(s,
"/root/Bela/projects/CaixadaMusica/cbass_noise.wav");
        ~tanpura = Buffer.read(s,
"/root/Bela/projects/CaixadaMusica/tanpura_noise.wav");
        ~flute = Buffer.read(s,
"/root/Bela/projects/CaixadaMusica/flute_noise.wav");
        ~vox1Buf = Buffer.alloc(s, s.sampleRate * 3);
        ~vox2Buf = Buffer.alloc(s, s.sampleRate * 3);
        ~ball = Buffer.read(s,
"/root/Bela/projects/CaixadaMusica/rolling_ball.wav");
        ~layer1 = Buffer.read(s,
"/root/Bela/projects/CaixadaMusica/Layer_1.wav");
        ~layer2 = Buffer.read(s,
"/root/Bela/projects/CaixadaMusica/Layer_2.wav");
```

```
	~layer3 = Buffer.read(s,
"/root/Bela/projects/CaixadaMusica/Layer_3.wav");

	s.sync;
	ServerTree.add(~setup);
	~setup.value();
});
)

(
s.waitForBoot({
	s.sync;

	// INPUTS
	~tuning = Synth.head(~inputGroup, \tuning);

	~droneSwitches = Synth.head(~inputGroup, \droneSwitches);
	~sustain = Synth.head(~inputGroup, \sustain);
	~ribbon1 = Synth.head(~inputGroup, \ribbon, [
	\addr, ~ribbon1Addr,
	\bus, ~ribbonBus1,
	]);
	~ribbon2 = Synth.head(~inputGroup,  \ribbon, [
		\addr, ~ribbon2Addr,
		\bus, ~ribbonBus2,
	]);

	// CONTROL PROCESSING
	~encoder = Synth.head(~controlGroup, \encoder);
	~bellows = Synth.after(~encoder, \bellows);

	s.sync;

	// SOUND SOURCES
	~droneVox1 = Synth.head(~sourceGroup, \droneVox1);
```

```
        ~droneVox2 = Synth.head(~sourceGroup, \droneVox2);
        ~bass = Synth.head(~sourceGroup, \bass);
        ~layer0 = Synth.head(~sourceGroup, \layer0);
        ~textureLayers = Synth.head(~sourceGroup, \layers);
        ~overtones = Synth.head(~sourceGroup, \overtones);

        s.sync;

        // FX RETURNS
        ~vox1Grain = Synth.head(~returnGroup, \droneGrains, [
              \input, ~vox1Bus,
              \grainDur, 0.1,
              \grainDens, 30,
              \amp, 1.5,
        ]);
        ~vox2Grain = Synth.head(~returnGroup, \droneGrains, [
              \input, ~vox2Bus,
              \grainDur, 0.05,
              \grainDens, 60,
              \amp, 1.5,
        ]);
        ~delay = Synth.head(~returnGroup, \fbDelay);

        s.sync;

        // SIGNAL PROCESSING
        ~droneMix = Synth.head(~mixGroup, \droneMix);
});
)
```