INSTITUTO POLITÉCNICO DE BRAGANÇA Escola Superior de Tecnologia e Gestão

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Mapeamento de movimento para controle de projetor holográfico laser

Alexandre Colauto Neto

Dissertação apresentada à Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança para obtenção do Grau de Mestre em Engenharia Industrial, ramo Engenharia Eletrotécnica, no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná.

Trabalho realizado sob a orientação de:

Professor João Rocha Silva

Professor Lucas Ricken Garcia

Bragança

Outubro 2018

Resumo

O ser humano possui a incrível capacidade de contagiar-se com emoções de outros indivíduos.

Este projeto tem o intuito de auxiliar esta transferência de felicidade e animação. Com a utilização de um acelerômetro preso a mão do utilizador, realiza o mapeamento dos movimentos e os transmite via bluetooth para um microcomputador, onde é classificado os movimentos e os transforma em mensagens DMX transmitidas via USB para um projetor holográfico, gerando assim padrões luminosos, sendo possível dessa maneira expressar a agitação do utilizador. Para isto utilizou-se a técnica de Aprendizagem de maquina para a classificação dos movimentos, mais precisamente o algoritmo de Regressão Logistica, que ainda foi comparado com outro algoritmo, o Support Vector Machine. Por fim os resultados foram satisfatórios alcançando uma precisão de aproximadamente 80%.

Palavras chave: DMX, OLA, Raspberrypi, acelerômetro, Machine Learning, bluetooth.

Abstract

The human being possesses the incredible capacity to be infected with the emotions of other individuals.

This project is intended to help this transfer of happiness and animation. With the use of an accelerometer attached to the user's hand, it maps the movements and transmits them via bluetooth to a microcomputer, where the movements are classified and transformed into DMX messages transmitted via USB to a holographic projector, thus generating luminous patterns, being able in this way, to express the agitation of the user. For this purpose, the Machine Learning technique was used to classify movements, more precisely the Logistic Regression algorithm, which was still compared to another algorithm, the Support Vector Machine. Finally the results were satisfactory reaching an accuracy of approximately 80%.

Key Words: DMX, OLA, Raspberrypi, acelerometer, Machine Learning, bluetooth.

Sumário

Lista de figuras

Lista de Abreviaturas

EUA Estados Unidos da America

USITT United States Institute for Theatre Technology

DMX Digital Multiplex

EIA Electronic Industries Association

TIA Telecommunications Industry Association

OLA Open Light Architecture

RDM Remote Device Management

RGB Red Green Blue

IP Internet Protocol

API Application Programming Interface

USB Universal Serial Bus

HDMI High Definition Multimedia Interface

DSI Display Serial Interface

GPIO General Purpose Input Output

SVM Support Vector Machine

OVO One-Vs-One

OVR One-Vs-Rest

1. Introdução

O uso de iluminação, nas artes e no entretenimento, pela humanidade vem desde o controle da luz natural do sol, do fogo, da vela e, com o desenvolvimento de novas tecnologias, a lâmpada. Atualmente são utilizados projetores holográficos capazes de gerar espetáculos inimagináveis. Entretanto, não importa quão avançado seja um projetor holográfico ele ainda precisa ser controlado para que se haja um espetáculo de qualidade. Para que tal controle possa ser realizado de forma mais artística este projeto foi desenvolvido.

1.1. Objetivo Geral

Desenvolver um sistema interativo para controle de um projetor holográfico, utilizado em iluminação de eventos baseado nos movimentos do utilizador.

1.2. Objetivos Específicos

Utilizar um acelerômetro *bluetooth* atrelado ao braço para realizar o mapeamento do movimento do utilizador.

A partir do mapeamento ativar padrões de iluminação em um projetor holográfico, através de um *Raspberry Pi*.

1.3. Estrutura da Tese

O presente trabalho é composto de sete capítulos. O primeiro apresenta os objetivos pretendidos a serem alcançados bem como a estrutura da tese.

O segundo capítulo apresenta uma revisão teórica a respeito da tecnologia utilizada em controle de iluminação.

O terceiro capítulo apresenta uma fundamentação sobre o hardware, isto é a parte física utilizada no projeto.

O quarto capítulo apresenta o estado da arte dos algoritmos utilizados.

O quinto capítulo apresenta o desenvolvimento, os passos tomados no desenvolvimento do protótipo.

O sexto capítulo apresenta a discussão dos resultados obtido.

Por fim, o sétimo capitulo apresenta a conclusão, bem como propostas para futuros melhoramentos.

2. Tecnologias e Controle de Iluminação

Será descorrido aqui o estado da arte das tecnologias utilizadas no projeto

2.1. Introdução

Muitas vezes, a primeira coisa que vem à cabeça quando se ouve o termo luzes de entretenimento são apenas lasers e luzes se movendo, porém, pode-se estar perdendo um ponto de vista interessante. Não se trata apenas de uma automatização no movimento de uma luminária, mas sim da necessidade de compreender o universo e experienciá-lo. Se trata da esquina onde ciência encontra a arte e uma influencia a outra [1]. A luz é um dos principais meios com ao qual o ser humano observa e vivencia o mundo ao seu redor. Ela pode mostrar, esconder ou modificar a percepção de forma de um objeto, sugerir movimento, distorcer ou ampliar texturas assim como criar ou ampliar sentimentos, e esta é apenas alguma das várias maneiras que a luz pode manipular a percepção do mundo [2].

Na Grécia antiga os anfiteatros já eram construídos em céu aberto, e virados do este para o oeste, a fim de se aproveitar a luz natural que atingiria apenas os atores. Para efeitos mais dramáticos de cores, toldos eram posicionados sobre a orquestra, deixando passar apenas alguns comprimentos de onda, criando uma sensação visual agradável. Esta forma de iluminação perdurou por séculos, e foi somente por volta do século 16 que a iluminação em espaço fechado começou a ser implementada na Itália, sendo utilizada velas em candelabros sobre o palco, no vão frontal do palco, assim como na escada principal, dessa forma a iluminação era focada nos atores, ajudando o público a permanecer imerso à peça e se preocupar menos com circundantes [2]

O uso de velas saiu de cena quando as primeiras luminárias a gás e óleo surgiram em meados de 1800. Estas eram colocadas na frente dos palcos, e ainda era

necessário um operador para colocar o óleo, acender e apagar a luminária quando se mudava a cena. Nessa mesma época o *Chestnut Tree Theatre* na Philadelphia, EUA, já introduziu um conceito de automação na iluminação do teatro, por meio de uma mesa com vários botões, era possível controlar o fluxo de gás em diferentes luminárias, conseguindo controlar assim a intensidade da luz, e dessa forma criar momentos muito mais dramáticos [3].

2.2. Controladores de luz

Já no século 20, a eletricidade começou a desempenhar um papel mais ativo no ramo da iluminação, pois esquentava menos que os sistemas a gás, eram mais seguros e possuíam um maior controle sobre as luminárias. Sistemas com *dimmers* (Figura 1) começaram a ser usados, porém eram extremamente grandes, caros, e totalmente analógicos. Utilizava-se sinais de 0 a 10 V para efetuar o controle do sistema de iluminação, sendo necessário cabos individuais com uma bitola elevada passando por todo o palco. Por si só já eram muito suscetíveis a ruídos, além de apresentarem um controle não-linear para algumas lâmpadas. Além disso, os primeiros modelos ainda efetuavam controle com o próprio sinal que a lâmpada utilizava, chaveando algumas centenas, quando não milhares de volts [2]

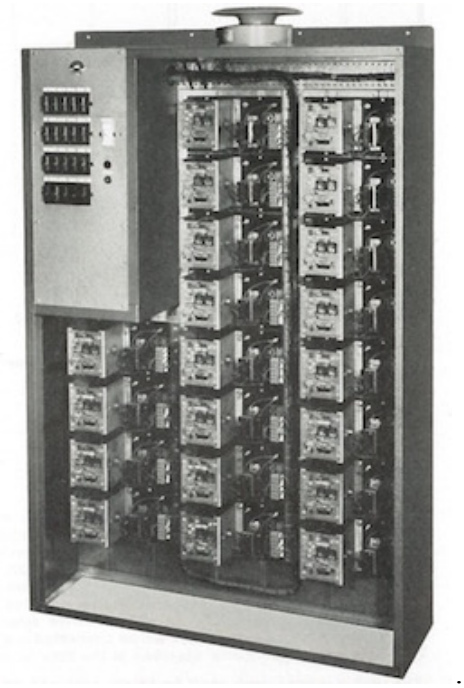

Figura 1: *Dimmer* de 20 canais JTM (1975) [4].

2.3. DMX 512

Os 60 canais que eram usualmente implantados nos controles analógicos e a baixa proteção contra ruídos não atendiam a demanda da indústria da iluminação, que estava em um crescimento acelerado. Cada fabricante, por sua vez, tinha o próprio protocolo de comunicação com os *dimmers*, que eram incompatíveis com outros equipamentos da época.

Assim sendo, uma padronização no controle fez-se necessária. A comissão de engenharia do United States Institute for Theatre Technology (USITT) se reuniu na conferência anual de 1986 em Oakland na Califórnia, e desta conferência nasceu o padrão que seria o menor denominador comum na indústria de iluminação de entretenimento, o DMX512 (*Digital Multiplex* com 512 pacotes de informação). DMX512, ou apenas DMX, é o padrão que descreve a comunicação digital entre um controlador e o equipamento de luz a ser controlado. Ele abrange as características elétricas, que são baseadas no padrão EIA/TIA–485, formato e protocolo de dado e os tipos de conectores [5].

Com o avanço na complexidade dos equipamentos, fez-se necessário a utilização de mais canais para poder realizar um controle apropriado dos equipamentos, podendo ser utilizado quantos canais o fabricante julgar necessário. Mais canais por dispositivos representa menos dispositivos que podem ser controlado através de uma única controladora. Assim os universos DMX foram introduzidos, que são basicamente diversos dispositivo ligados em um mesmo barramento. Cada dispositivo de um universo deve ser ligado em cascata, e desta forma escutam apenas as mensagens de seu universo. Os universos podem possuir ainda até 512 canais de controle DMX, e as mesas mais recentes já possuem saída para diversos universos [1].

O barramento de comunicação é baseado em um diferencial RS-485, que será tratado mais profundamente adiante. Teoricamente o RS-485 suporta 32 componentes no barramento, mas este número pode ser incrementado utilizando repetidores de sinal [3].

2.4. Camada física

Em um universo DMX, o controlador possui uma saída e os dispositivos possuem uma saída e uma entrada, ficando organizados de forma serial, onde o controlador está no início do universo, como é chamado cada cadeia de dispositivos, e cada componente está conectado com outro em forma de cascata, podendo haver quantos dispositivos forem possíveis endereçar em 512 canais, sendo que o número de canais varia de acordo com a complexidade do dispositivo.

Por um período, a entrada de um dispositivo era ligada à saída de outro através de cabos XLR-3 (Figura 2), porém, devido ao fato de serem idênticos aos cabos diferenciais utilizados em aplicações de áudio, que necessitam geralmente de uma fonte CC de 48V, ocorriam eventuais conexões de um dispositivo de iluminação em um dispositivo de áudio, que acabavam por vir a danificar o equipamento [2].

Figura 2: Cabo XLR-5 fêmea (esquerda) para XLR-3 Macho (direita) [6].

Para sanar este problema, alterou-se o padrão de forma a permitir apenas conexões XLR-5, que continuam possuindo as mesmas funções do XLR-3, porém com 2 pinos a mais que não são utilizados, e existem apenas para prevenir conexões acidentais com equipamentos de áudio. Mas como os dois pinos adicionais são apenas para prevenir conexões errôneas, muitos fabricantes continuam produzindo seus equipamentos com entrada de 3 pinos, quebrando assim o padrão, o que acaba por tornar comum encontrar cabos no mercado com saída XLR-5 dos controladores para XLR-3 dos dispositivos (Figura 2) [1].

Como os dispositivos estão ligados em série (Figura 3), todos irão "escutar" todas as instruções, isto é, um determinado sinal emitido pelo controlador passa por todos os dispositivos de um universo, assim como ocorre analogamente em um barramento de dados, mas os dispositivos irão processar apenas aqueles que estão endereçado a eles. Podendo haver também situações onde dois ou mais dispositivos conectados ao mesmo universo e com o mesmo endereço, neste caso ambos irão efetuar o mesmo movimento, desde que o hardware de ambos sejam idênticos [2].

No final de cada universo há um terminal, um resistor de 120 Ω, segundo o padrão RS-485 para prevenção de reflexão de sinal. A maioria dos fabricantes já embutem um *dip switch* com esse valor dentro dos dispositivos [2].

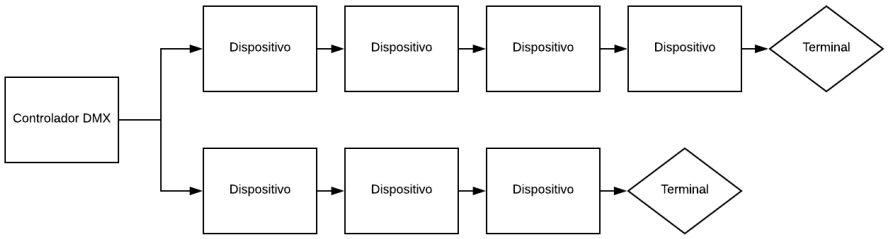

Figura 3: Fluxograma de dois universos DMX.(fonte: o autor)

2.4.1. RS-485 (EIA/TIA - 485)

RS-485 é um padrão de comunicação serial, desenvolvida entre duas entidades em cooperação: *Electronic Industries Association* (EIA) e *Telecommunications Industry Association* (TIA). Nomeadamente a princípio com o prefixo “RS” (RS-485) , que significa *Recommended Standard* (Padrão Recomendado), mas que depois veio a mudar para EIA/TIA (EIA/TIA-485) para ajudar a identificar a origem do padrão, entretanto é muito comum encontrar ainda a nomenclatura de RS-485 [7].

É baseado em uma transmissão diferencial através de pares trançados, onde a informação é transmitida de forma complementar através de dois fios de polaridades opostas, geralmente trançados, e a informação é a diferença desses dois sinais. Dessa forma qualquer ruído ocorrerá em ambos os cabos, e será eliminado através do diferenciador (Imagem 4). Os circuitos receptores identificam a diferença de tensão entre os sinais dos condutores, se o condutor “+” for maior que o condutor “-” tem-se o sinal lógico 1, caso o condutor “-” seja maior que o “+” tem-se um nível lógico 0 [8].

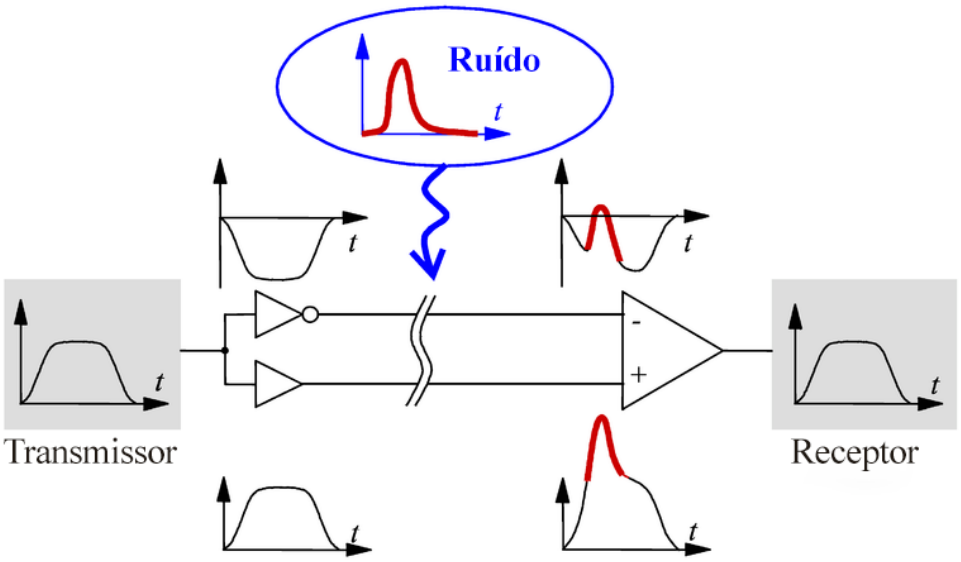

Figura 4: Esquemático de transmissão diferencial (Modificado) [9].

Dessa forma é possível atingir altas velocidades em curtas distâncias (aproximadamente 10 Mbps com 10 m de cabo), ou longas distâncias com baixa velocidade (aproximadamente 1000 m de cabo, mas com uma transmissão de apenas 100 kbps), como mostra a *Figura 5* [8].

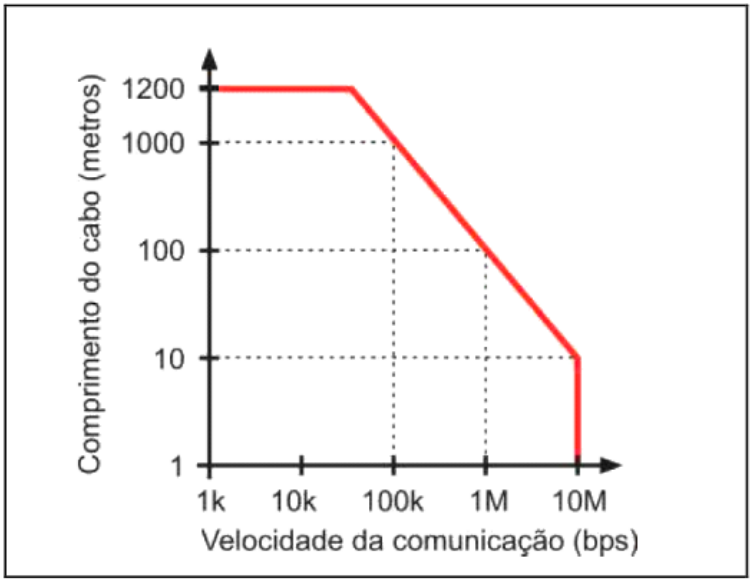

Figura 5: Comprimento do cabo vs velocidade de comunicação[8].

2.4.2. Protocolo DMX

O protocolo DMX determina uma transmissão assíncrona a uma velocidade de 250 kbaud. Como a maioria das transmissões assíncronas, a comunicação é feita através de pacotes. Cada pacote representa um canal DMX, podendo haver até 512 em um único universo. Os pacotes são constituídos de 8 bits, variando seu valor de 0 a 255 ($2^8 = 256$) esses pacotes controlam determinada característica do dispositivo, como por exemplo rotação, inclinação, cor, entre outros parâmetros determinados pelo fabricante. Todos os pacotes configurados são enviados em ordem, dessa forma os dispositivos só precisam estar configurados para reconhecer o endereço inicial de seus canais [10].

Por exemplo um projetor laser com 8 canais de controle e endereço inicial 32, terá que esperar até o 32º pacote de dados para começar a efetuar a escuta, e irá ler até o 40º pacote de dados, cada pacote corresponde a um parâmetro de controle do projetor [10].

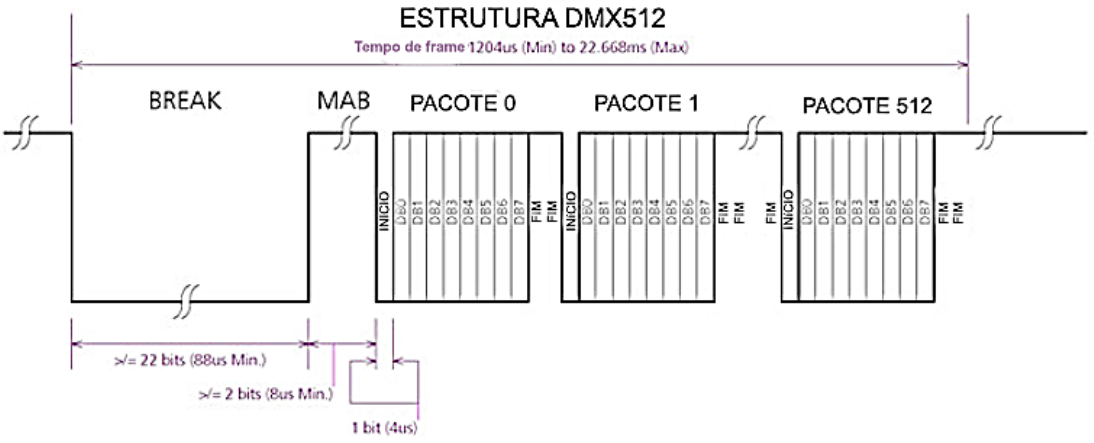

Figura 6: Estrutura do protocolo DMX512 [10] (Adaptado)

O protocolo é iniciado com um cabeçalho, sendo um *break* (pausa) de pelo menos 88 µs ou 22 bits de nível lógico baixo, seguido por um *Mark After Break* (Marca depois da pausa), de dois bits de nível lógico alto, ou 8 µs, sinalizando o início dos dados.

É importante frisar que o pacote 0, é o início da mensagem e nunca deve ser utilizado por nenhum dispositivo, sendo sempre composto de zeros. Após isto cada pacote é enviado em sequência, sendo o primeiro bit sinalizando o início do pacote, com o valor lógico 0, seguido por 8 bits de dados, que são os valores que o dispositivo irá efetivamente ler (0 a 255), e mais dois bits de final de mensagem, com nível lógico 1 (Figura 6).

Após o último pacote ser transmitido um novo *break* é enviado, sinalizando a retransmissão da mensagem. É possível ter até 512 pacotes sendo transmitidos, não sendo necessário transmitir os que não estão sendo utilizados, economizando, dessa forma, tempo de transmissão. Por exemplo, 512 pacotes transmitidos mais o *break* levam aproximadamente 0,022 segundos, o que resulta uma transmissão de um pouco mais de 45 Hz, já um universo com 128 canais alocados (isto é o último canal alocado é o 128) tem como tempo de transmissão de apenas 0,006 segundos (166 Hz). Como o tamanho da mensagem é determinado pelo último canal alocado, os dispositivos sempre tentam procurar o menor canal livre disponível, levando em conta o número de canais necessários para o dispositivo.

Como a maioria dos controladores DMX possuem múltiplos universos, é possível realizar um bom balanceamento de universos para garantir uma taxa de transmissão mais elevada quanto possível. Vale salientar também que como o controlador é o único dispositivo transmissor, não há risco de colisões, e nem a necessidade de algoritmos de correção de erro, uma vez que os pacotes são transmitidos continuamente [10].

2.4.3. Controladores

Como o próprio nome já diz, controle ou console, é onde o controle das luminárias é realizado, é onde ocorre a configuração, programação e computação dos dados. Variando desde dos mais simples como um *fader* (potenciômetro) por canal, ou seja um *fader* por cada parâmetro, e uma memória para salvar alguns *presets,* até os consoles mais complexos que controlam dezenas de universos (Figura 7).

Com estes controladores já é possível criar banco de memórias em comum, para lidar com as variáveis dos dispositivos simultaneamente, e dessa forma controlar parâmetros diretamente (ao contrário de *faders*). Podendo assim escolher uma cor diretamente em uma paleta de cores no próprio controlador e alterar as cores do ambiente todo de uma só vez, já que é o próprio controlador que irá realizar a programação, efetuar os cálculos e gerar os pacotes de dados para cada dispositivo independente [10].

Figura 7: Comparação entre dois controladores DMX, DMX disco 384 (à esquerda) [11] e Avolites Sapphire Touch (à direita) [12].

O custo destes equipamentos pode variar de algumas centenas de dólares, para os modelos mais simples, até milhares de dólares para os que possuem maior complexidade. Para tornar a iluminação de entretenimento mais acessível, tanto para hobby, até para eventos que não possuem um orçamento elevado, surgiu o *Open Light Architecture* (OLA).

2.5. Open Light Architecture (OLA)

O OLA é uma das faces do projeto *Open Light Project*, que visa através de um esforço diversificado, acelerar a adoção de novos protocolos padronizados, enquanto fornece um *software open source* de qualidade e confiável para a indústria de

iluminação. Além do OLA, o *Open Light Project* conta com RDM *Responder Tests, libartnet, Logic RDM Sniffer, Arduino RGB Mixer*, entre outros projetos, mas será abordado apenas o OLA neste trabalho [8].

O OLA provém aplicações para envio e recebimento de mensagens DMX usando dispositivos físicos ou transmitindo via *Internet Protocol* (IP), permitindo realizar o controle de diferentes universos e ainda trabalhar como ponte entre dispositivos de diferentes fabricantes que não são compatíveis entre si. Tudo isso através de *Application Programming Interface* (API), que são rotinas e padrões de programação, que traduzem as instruções da linguagem de programação para instruções que o programa consegue identificar [13], isto permite linguagens como C++, Java e Python, Linhas de Comando e interface *WEB*.

Através das API's é possível criar *triggers*, rotinas, *loops* e qualquer tipo de manipulação da saída DMX a partir de uma entrada. Pode-se, por exemplo, definir uma rotina para os dispositivos executarem cada vez que uma tecla for pressionada, oscilar as luzes de acordo com a frequência da música, ou ainda através de algum mapeador de movimento.

O OLA pode ser instalado nos principais sistemas operacionais disponíveis hoje, Windows, Mac OS X, Free BSD e Linux. No caso, este último será utilizado neste projeto, mais especificamente a distribuição Raspbian, que é uma versão do Debian criada especificamente para rodar nas placas de desenvolvimento Raspberry Pi.

3. Hardware

Camada física do projeto, isto é, os componentes aqui utilizados.

3.1. Raspberry Pi

Raspberry Pi (Figura 8) é um microcomputador do tamanho de um cartão de crédito, desenvolvido no Reino Unido pela fundação Raspberry Pi, com o intuito de estimular o aprendizado de computação básica nas escolas. O sistema operacional também é fornecido pela fundação, é disponibilizado uma arquitetura em Arch Linux e outra em Debian, o Raspbian. O Raspberry Pi ainda vem com o Python como sua linguagem principal, mas ainda tem suporte a BBC BASIC, C e Perl. Possui ainda 4 portas USB, uma porta ETHERNET, saída de áudio e vídeo (P2 de quatro polos), HDMI, micro USB, conector DSI (Display) e uma GPIO de 40 pinos [14].

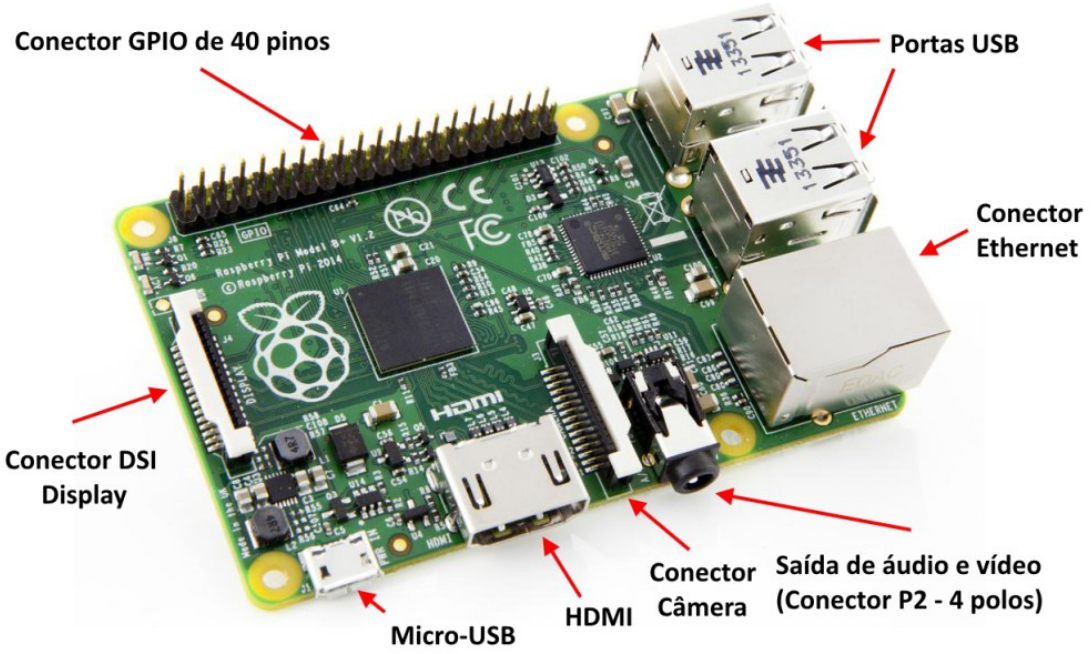

Figura 8: Raspberry PI [15].

O modelo Raspberry PI 3, conta ainda com um processador de 1.2GHz, 1Gb de memória RAM, Wi-Fi e *Bluetooth* integrados.

Por meio dos periféricos já inclusos nesta placa de desenvolvimento, é possível conectar diversos dispositivos e facilitar o processo de prototipação de um projeto. Por

exemplo, é possível conectar uma infinidade de dispositivos através do *bluetooth* integrado, tanto dispositivos de saída (atuadores), quanto de entrada (sensores). Neste trabalho, será uma ferramenta importante para realizar a aquisição dos dados provenientes de um acelerômetro, processá-los e atuar de forma a controlar a movimentação de luzes de entretenimento.

3.2. Acelerômetro

São sensores que, apesar do nome, não medem quanto a velocidade varia ao longo do tempo, ao invés disso é calculado a força. Como possuem uma massa fixa em seu interior, é possível medir a força que essa massa aplica sobre uma das placas de um capacitor, e com a variação da capacitância é possível mensurar a aceleração que está sendo aplicada no sistema.(Figura 9).

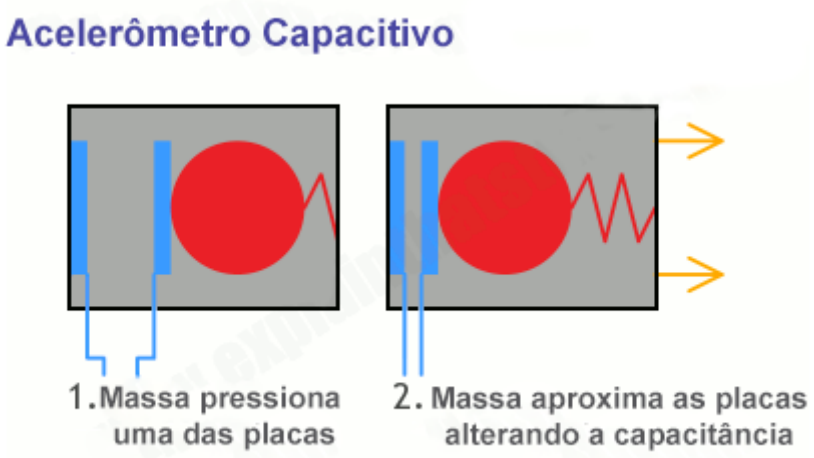

Figura 9: Acelerômetro capacitivo (adaptado de [16])

Há ainda outros tipos de acelerômetros como os mecânicos, onde a massa move uma caneta sobre um papel, acelerômetros que utilizam cristais piezoelétricos, que geram uma pequena tensão elétrica de acordo com a força aplicada sobre eles, e ainda outros que medem a variação em um campo magnético.

O modelo de acelerômetro que será empregado neste artigo é o JY-61, um dispositivo que combina um acelerômetro capacitivo de 3 eixos junto com um giroscópio de 3 eixos (MPU 6050), e tem integrado o *Digital Motion Processor*™ (DMP) que é responsável por realizar a filtragem do sinal, o processamento do movimento e a comunicação serial I²C.

Para realizar a comunicação com o Raspberry Pi, um módulo Bluetooth HC-06 está integrado com o módulo JY-61, lendo os dados de forma serial que o acelerômetro produz e transmitindo estes via comunicação sem fio(Figura 10).

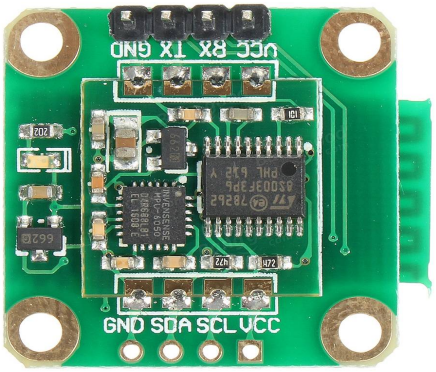

Figura 10 : Módulo JY-61 + HC-06 (parte posterior), acelerômetro acoplado com modulo *bluetooth* [17].

O acelerômetro é responsável por traduzir o movimento do utilizador em um vetor com os valores de aceleração, mas o vetor por si só não é capaz de ativar os gatilhos dentro do OLA, para que isso seja possível é necessário realizar uma classificação dos movimentos para se obter um padrão referente a cada movimento.

4. Machine Learning

Uma descrição do funcionamento dos algoritmos de classificação.

4.1. Introdução

Uma boa abordagem para classificação e reconhecimento de padrões é a utilização de técnicas de *machine learning,* ou em português aprendizado de máquina. Segundo Alpaydin [18] *machine learning* é definido como a programação de um computador para otimizar uma performance com base em exemplos ou experiências passadas [18].

Pode-se fazer uma analogia do *machine larning* com o aprendizado humano: a partir de referências (experiência própria, leitura ou mesmo observando uma pessoa mais habilidosa) construímos modelos, e a partir deste criamos expectativas para o resultado de algumas situações.

4.2. Support Vector Machine (SVM)

Support Vector Machine (SVM) é um algoritmo de aprendizado de máquina, um dos que mais vem crescendo em popularidade graças ao seu sucesso prático. Sucesso este geralmente atribuído à sólida fundamentação teórica baseada na teoria de Vapnik–Chervonenkis que busca explicar o *machine learning* a partir de um ponto de vista teórico [19].

O SVM é útil em casos de classificação e de problemas de regressão. Seu comportamento é mais facilmente visualizado em uma interpretação geométrica onde os *data points* são representados como pontos em um espaço *n*-dimensional, mapeados de forma que cada classe tenha um espaço tão amplo quanto possível entre elas. Isto é feito gerando um hiperplano de dimensão *n*-1, de forma que possua a maior distância possível entre os vetores de suporte, ou seja, pontos que estão mais próximos de uma classe vizinha (Figura 11), sendo este os mais complexos de se classificar [19].

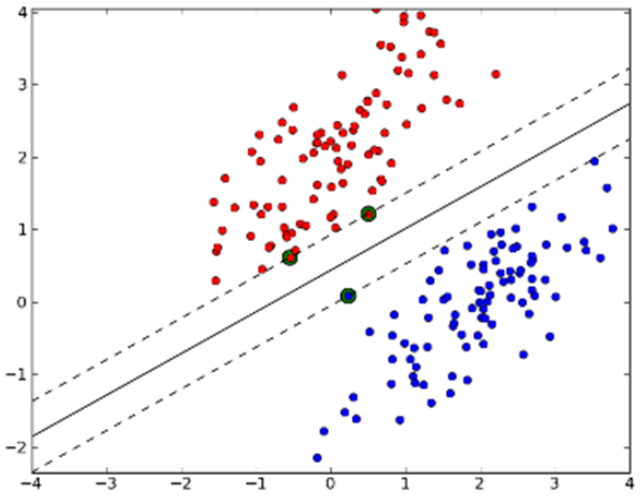

Figura 11: Exemplo de SVM, com duas classes, *support vectors* e o hiperplano *[20]*.

A formulação matemática responsável pela geração do hiperplano varia de acordo com o problema a ser classificado, podendo adotar uma função, dentre outras formas, linear, polinomial e *Radial Basis Function* [21].

Em situações que mais de uma classe é necessária podem ser adotadas duas abordagens distintas *One-Vs-One* (OVO) ou *One-Vs-Rest* (OVR). Na abordagem OVO, cada classe é categorizada individualmente contra todas as outras classes, gerando um hiperplano para cada categorização (Figura 12):

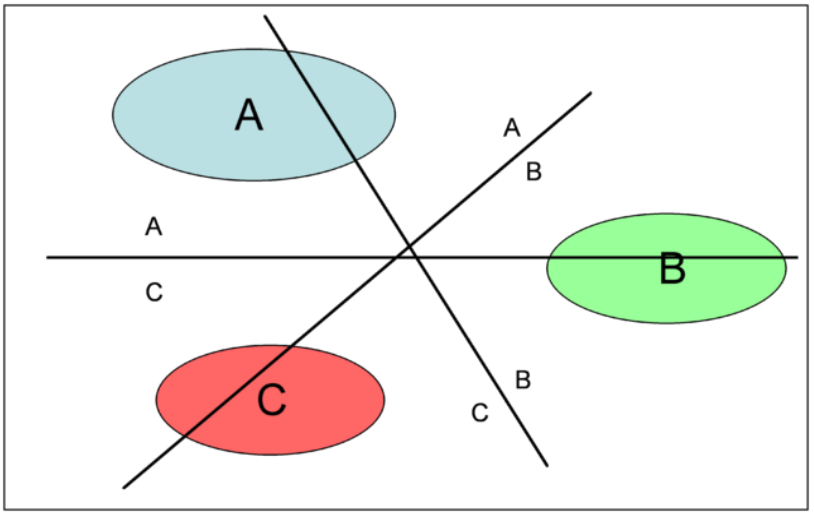

Figura 12: Exemplo de uma abordagem *One-Vs-One* [22]

Já na abordagem OVR, a classificação é realizada tomando uma classe como referência e considerando todas as outras como apenas uma classe, ao invés de classificar uma por uma como na abordagem OVO (Figura 13) [23].

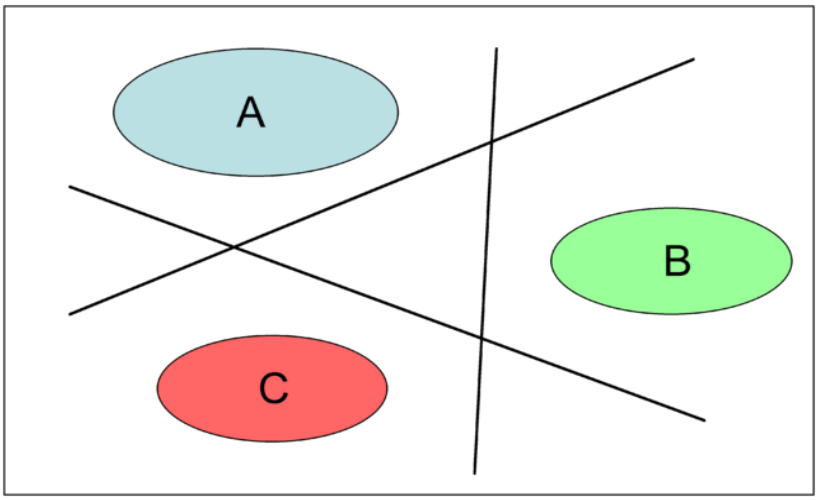

Figura 13: Abordagem One-Vs-Rest [22].

4.3. Regressão Logística

A regressão logística estima a probabilidade da ocorrência de um determinado evento em face de um conjunto de variáveis explanatórias. De forma simplificada, com base nos dados de aprendizagem é gerado uma função probabilística que retornará valores entre 0 e 1, sendo 0 a classe negativa e 1 a classe positiva. Esta função então é utilizada para predizer a probabilidade de uma entrada futura pertencer a uma das duas classes. A função probabilística adota um formato em "S" (Figura 14), forma característica deste tipo de função. Seus coeficientes são determinados a partir do método da máxima verossimilhança, que encontra uma combinação que maximiza a probabilidade da amostra ter sido observada pertencente a determinada classe [24].

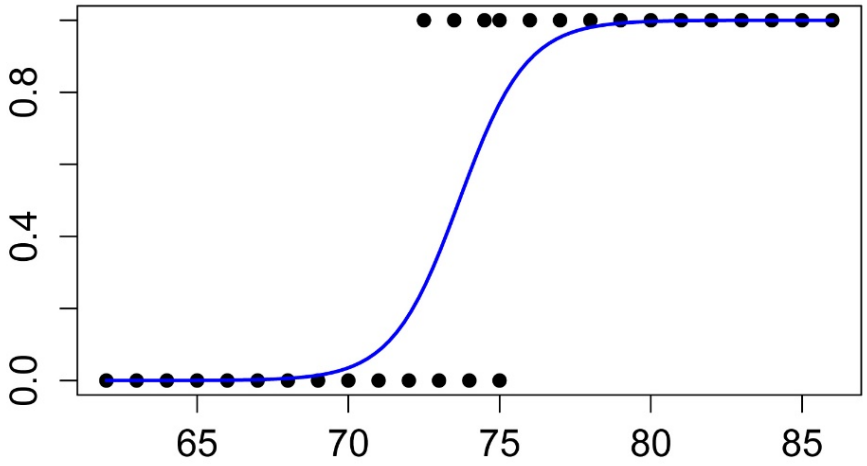

Figura 14: Exemplo de uma regressão logística, onde os dados (pontos preto) fornecem os coeficientes para a curva probabilística (linha azul) [25].

5. Desenvolvimento

5.1. Raspbian

Para a instalação do sistema operacional que rodará no *RaspberryPi* a imagem do Raspbian foi baixada diretamente do site da *Raspberry* [14], e com o auxílio de um *software* de criação de dispositivos *bootáveis* chamado *Etcher* [26] . Em seguida deu-se início ao processo de instalação e configuração do *Raspbian*. Vale ressaltar que a imagem já vem, em sua grande maioria, configurada, ficando a critério do utilizador definir apenas o *layout* do teclado, a região geográfica do dispositivo e alguns outros detalhes que não influenciam no desenvolvimento do projeto.

5.2. Open Light Architecture

Com o *Raspbian* instalado e funcionando corretamente deu-se início ao processo de instalação do OLA, para que possa ser gerado os dados que controlarão o dispositivo de iluminação. Por um problema de compatibilidade a versão do OLA criada especialmente para o *Raspberry Pi* não possui compatibilidade com o *firmware* do *RaspberryPi 3*. Por fim a versão do OLA destinada ao Linux funcionou corretamente e acabou sendo esta utilizada na versão final. Para a instalação foi utilizado os passos recomendados no tutorial do site do OLA [27] .

5.3. Primeiros testes

Para poder verificar o bom funcionamento do OLA, foi necessário realizar alguns testes preliminares, isto é, criar um universo DMX dentro do OLA, enviar pacotes de dados e verificar se a saída estava coerente. No processo de criação do universo DMX o OLA designa o dispositivo que irá realizar a interface com o sistema de iluminação. Atualmente há uma infinidade de dispositivos USB-DMX disponíveis, mas geralmente possuem um custo elevado e possuem muitas funções que não seriam utilizadas neste projeto. Por fim acabou-se por escolher um simples módulo USB-RS485 (Figura 15), pois possui um custo muito menor que os módulos USB-DMX e ainda assim compartilham das mesmas propriedades elétricas do protocolo DMX [10].

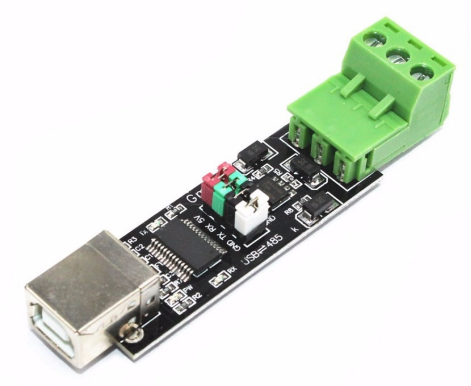

Figura 15: Módulo USB para RS485 [28].

O sistema OLA identifica este dispositivo como *DummyDevice*, ou em português um dispositivo burro, pois não possui qualquer tipo de *feedback* característico de outros dispositivos USB-DMX, apenas envia o pacote de dados gerado pelo OLA.

O OLA possui um arquivo para testes localizado no diretório *ola/python/examples/,* chamado ola_send_dmx.py. Escrito em *python*, este programa gera e envia dados para os 3 primeiros canais do universo, sendo os valores respectivamente: 10 no primeiro canal, 50 no segundo e 255 no terceiro. A saída então foi verificada com o auxílio de um osciloscópio ligado à saída do módulo. Pelo fato do padrão RS-485 possuir um barramento positivo, um negativo e a referência (terra), o osciloscópio foi ligado à parte positiva e à referência tornando possível plotar a saída e verificar assim o bom funcionamento do sistema até então.

5.4. Acelerômetro

O próximo passo no desenvolvimento do projeto foi realizar a comunicação do módulo JY-61+HC-06 com o *RaspberryPi*. Para esta etapa foi utilizado a biblioteca desenvolvida especialmente para esta conexão: Pymotion Tracker [29]. Desenvolvida especialmente para a leitura dos dados do módulo JY-61+HC-06, esta biblioteca fornece de forma clara métodos para a leitura de cada uma das informações fornecidas pelo acelerômetro, isto é, a aceleração, o ângulo e a velocidade angular em cada um dos 3 eixos (X, Y, Z).Importante lembrar que a função já fornece os valores escalonados, ou seja aceleração até 16g, os ângulos variando de -180 a 180 assim como a velocidade angular variando até 2000 rad/s. Além de fornecer uma função que pode ser chamada em outros programas para a utilização destes dados.

A única configuração necessária para esta biblioteca é o endereço *bluetooth* do módulo. O endereço pode ser facilmente descoberto utilizando o próprio *bluetooth* do *RaspberryPi*, para tal é necessário no terminal do *RaspberryPi* utilizar o comando: *sudo bluetoothctl,* para que seja ativado o *Bluetooth*. Em seguida utilizar o comando *agent on* para que o módulo se torne visível para os dispositivos próximos e assim possa negociar o código de pareamento. Com o *agent* definido como ligado, o próximo passo é iniciar a busca, através do comando *scan on.* Uma lista com todos os dispositivos deve aparecer (Figura 16).

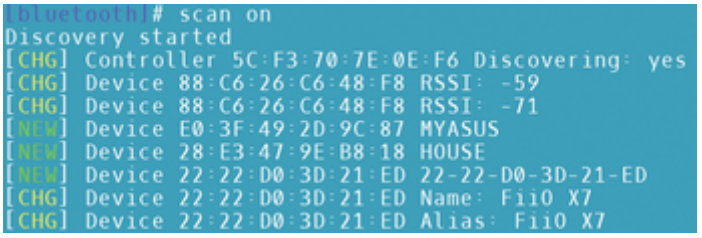

Figura 16: Exemplo da lista de dispositivos *Bluetooth* [30].

Com o endereço correto do módulo já é possível iniciar a leitura de dados.

Porém nesta etapa do projeto houve um contratempo: o módulo era alimentado por uma pilha de Lithium CR2032 (Figura 16), mas esta não era capaz de alimentar o circuito pois o módulo opera em uma tensão entre 3,3 V a 6 V para seu perfeito funcionamento. Tal empecilho foi resolvido utilizando uma bateria originalmente desenvolvida para *action cameras*, como por exemplo a GoPro®. Esta bateria possui uma tensão nominal de 3.7 V e fornece uma corrente de 1050mAh (Figura 17).

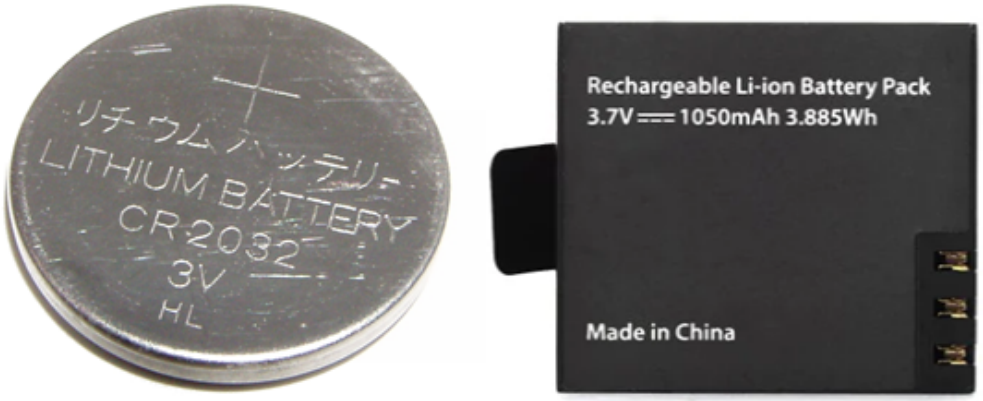

Figura 17: Bateria de Lithium CR2032 de 3V (esquerda) [31] e Bateria de Lithium-ion de 3.7V e 1050mAh (direita) [32].

Após este pequeno contratempo, foi possível realizar a leitura dos dados e verificou-se a coerência simplesmente exibindo os valores do acelerômetro e movendo o módulo enquanto observava se os valores exibidos na tela correspondiam ao movimento que estava sendo efetuado.

5.5. Mapeamento do movimento

A grande maioria dos projetores são capazes de rotacionar nos 3 eixos, sendo capaz de projetar sua luz virtualmente em qualquer direção. Com base nisso, foram utilizados os valores do acelerômetro para que o projetor acompanhasse o movimento do utilizador.

Considerando que a aceleração é a derivada da velocidade, isto é a taxa de variação da velocidade, e a velocidade é a derivada do espaço, é possível obter o deslocamento integrando os valores do acelerômetro duas vezes.

Na prática isso gera um problema, pois nenhum dispositivo real é totalmente preciso e livre de ruídos, e estes erros são amplificados quando são realizadas as integrações. Algumas técnicas de filtragem ajudam a reduzir o impacto deste desvio, como por exemplo o filtro de Kalman (nativo nos módulos JY-61) [33].

O maior dos problemas na utilização de apenas valores de aceleração para calcular o deslocamento é o *Drifting* (derrapagem) do zero, ou referência, que acaba por distorcer todo o resto do sistema. Assim sendo, alguma forma de definir o zero é necessária para que tal função seja viável sua aplicação no projeto. Esta característica acabou impossibilitando a implementação no estado atual, mas pode ser uma oportunidade para aprimoramentos futuros.

5.6. Machine Learning

Durante a pesquisa de trabalhos de referência foi encontrado um projeto similar a este aqui proposto: *Gesture Keyboard*[1] [34], por Federico Terzi [35]. O projeto de Terzi também utilizava o mesmo módulo JY-61, era escrito em python e utilizava o *RaspberryPi* para processamento dos dados. O projeto consiste na utilização de um algoritmo de *Machine Learning*, mais precisamente Support Vector Machine para a classificação dos dados do acelerômetro em caracteres. A princípio era planejado utilizar o projeto de Terzi como base e adaptá-lo para que os caracteres que seriam a saída ativassem rotinas dentro do código do OLA, fornecendo assim um padrão de luz para cada movimento. Não foi possível realizar tal adaptação devido à problemas com bibliotecas. Um projeto foi então iniciado a partir do zero.

5.7. SVM

Baseando-se no projeto de Terzi, foi possível perceber a utilização de uma biblioteca denominada Scikit Learn (também conhecida como Sklearn) [36], projetada para *python*, com base em Numpy, Scipy e matplotlib, que são outras bibliotecas amplamente utilizadas em aplicações científicas em python. Focada em análises de dados e classificação, a SKlearn é uma compilação de vários algoritmos de *Machine Learning*, mas para este projeto foi utilizado a princípio apenas o SVM e mais futuramente a Regressão Logística.

[1] Tradução em português: Teclado de Movimento

Com todas as formulações matemáticas já embutidas em suas funções foi necessário apenas passar os dados estruturados de forma correta para que a biblioteca realize os cálculos necessários e formule um modelo para as variáveis em questão. Um ponto a se levar em consideração é que as formulações matemáticas são sensíveis a escala, ou seja elas perdem um pouco sua precisão para números grandes, por isto a própria biblioteca fornece funções para pré-processamento que neste caso consiste em transformar os dados em uma distribuição normal de mediana um e desvio padrão unitário, o que não chegou a ser necessário visto que o programa utilizado para ler os dados do acelerômetros já realizava o escalonamento, sendo necessário apenas remover esta parte da função para que os valores se encontrem entre 0 e 1, tornando o algoritmo mais preciso.

5.8. Construtor SVM

O primeiro passo na implementação do classificador SVM foi criar uma variável no código que será atribuído o construtor do classificador, isto é a variável que será referência para ser possível chamar as funções referentes ao algoritmo.

Na Figura 18 temos o construtor sendo criado com alguns atributos, porém tais atributos não apresentaram uma mudança significativa nos resultados, optando-se por permanecer com os atributos *default.* O construtor foi então atribuído a variável model_svm, desta forma para acessar qualquer função do construtor em questão basta utilizar por exemplo model_svm.*função(),* sendo *função()* a função desejada.

```
model_svm= svm.SVC(C=0.001, cache_size=1000,kernel='rbf')
```

Figura 18: Criação do construtor SVM, com parâmetros (fonte: o autor).

5.9. Estrutura dos dados

O formato de entrada que o classificador do SVM espera são duas matrizes, a primeira matriz de dados com duas dimensões, (onde cada linha é uma amostra e cada coluna é uma característica). A segunda matriz é o rótulo de cada linha, ou seja, qual classe cada amostra se encaixa. É importante estar atento às dimensões, todas as amostras precisam ter o mesmo número de características, e a matriz de rótulos precisa ser do mesmo tamanho do número de linhas da matriz de dados, ou seja, cada amostra (linha) da matriz de dados precisa ter um valor correspondente na matriz de rótulos. A imagem a seguir facilita o entendimento:

```
>>> X = [[1, 2], [2, 4], [4, 5], [3, 2], [3, 1]]
>>> y = [0, 0, 1, 1, 2]
```

Figura 19: Exemplo dos dados necessários para o treinamento do modelo SVM [37]

É possível observar duas matrizes “x” e “y”, sendo a matriz “x” a de dados, possuindo duas dimensões, com 5 amostras e 2 características, e a matriz “y” a matriz de rótulos possuindo 5 elementos, um para cada amostra.

Em seguida é chamada a função *fit(x,y),* que realiza a correlação dos dados com seus devidos rótulos, realizando todos os cálculos matemáticos para que esta separação seja o mais preciso possível. É basicamente esta função que treina o construtor, e gera o modelo que será comparado em previsões futuras.

A função *predict(z)* é utilizada para realizar a previsão em qual classe a variável "z" é mais provável de se encaixar. Ela funciona tomando uma variável com as mesmas dimensões dos dados utilizados para a aprendizagem, e retornando o número da classe correspondente. Se utilizarmos o exemplo acima e chamarmos a função *predict(z),* tomando z= [4,3], a função retornaria 1, pois é a classe que mais se aproxima dos valores que foram passados.

Como o SVM espera receber vetores de dimensões idênticas para que possa ser feita a classificação, a primeira tentativa foi criar uma matriz com 9 linhas, sendo uma linha para cada parâmetro, isto é três linhas para aceleração x, y e z, outras três para os valores de ângulos x, y e z, e mais três linhas para a aceleração angular também x, y e z. Onde cada coluna seria uma amostra (Tabela 1).

Tabela 1: Exemplo da estrutura dos dados (fonte: o autor)

Parametro	**Amostra_1**	**Amostra_2**		**Amostra_n**
Acc_x	0.2646	0.3678	...	0.1256
Acc_y	0.2646	0.3678	...	0.1256
Acc_z	0.2646	0.3678	...	0.1256
Ang_x	0.2646	0.3678	...	0.1256
Ang_y	0.2646	0.3678	...	0.1256
Ang_z	0.2646	0.3678	...	0.1256
VelAng_x	0.2646	0.3678	...	0.1256
VelAng_y	0.2646	0.3678	...	0.1256
VelAng_z	0.2646	0.3678	...	0.1256

Esta matriz era a representação de um único movimento, e para um bom aprendizado é recomendado utilizar em média 40 exemplos em cada movimento. Para seguir a estrutura do *fit(x,y)* é necessário ter um vetor de dados com *n* exemplos, e um outro vetor de *n* posições, sendo cada posição o rotulo da classe que o exemplo pertencia. Foi então criado um vetor de 40 posição onde cada posição era uma matriz representando um movimento, e outro vetor também de 40 posições contendo apenas o rotulo do movimento, por exemplo "1".

O grande problema disto é que resultava em uma matriz de 3 dimensões, não sendo legível para o algoritmo de classificação, que esperava uma matriz de apenas duas dimensões.

A solução foi então concatenar todos os valores em uma linha só, seguindo a seguinte estrutura:

Tabela 2: Dados concatenados em uma mesma linha(fonte: o autor).

0.2646	0.3678	...	0.1256	0.2646	0.3678	...	0.1256	...	0.1256
Acc_x				**Acc_y**				...	**VelAng_z**

Ou ainda no próprio código:

```
x=np.append(x,[Accx,Accy,Accz,Angx,Angy,Angz,Accax,Accay,Accaz])
```

Figura 20: Concatenação dos dados (fonte: o autor).

Desta forma foi possível reduzir a dimensão da matriz para apenas duas, onde cada linha ficou um movimento podendo ser atribuída a um rotulo.

5.10. Variança no tempo

A principal discrepância entre este projeto e o projeto de Terzi que foi tomado como base é que no projeto de Terzi havia um botão que sinalizava o início e o fim de cada movimento, facilitando a estruturação dos dados e por consequência a classificação do SVM. Infelizmente não foi objetivo deste projeto implementar um botão para definir início e fim de cada movimento, o intuito é realizar um mapeamento em tempo real.

Foi calculado que cada movimento levava em média 2 segundos para ser executado, então cada linha da matriz deveria ter dois segundos de registro de dados. Para evitar informações redundantes bem como facilitar o cálculo do SVM (que possui complexidade N^3), foi utilizada a função *sleep()*, que funciona realizando uma pausa no código. Desta forma menos dados e dados mais espaçados facilitam o cálculo e aumentam a precisão do algoritmo.

Ainda há o problema que o SVM espera vetores de tamanhos e dimensões idênticas para que possa ser feita a classificação, e este projeto é dinâmico, ou seja, não há uma forma de sinalizar o fim de um movimento e o início de outro. Assim sendo uma solução encontrada foi criar um *loop*, onde um conjunto com aproximadamente 40 movimentos (sendo um movimento 2 segundos de registro de dados do acelerômetro), é colocado em série para se registrar uma classe. Desta forma é possível registrar deslocamentos no tempo, bem como pedaços de movimentos que acabam por se situar entre um registro e outro.

O mesmo procedimento descrito foi implementado tanto no processo de aprendizado do classificador bem como no momento de se realizar a previsão.

5.11. Gatilhos DMX

Os valores retornados pela função *predict( )* são valores inteiros, correspondente a classe do movimento. Estes valores determinam qual *loop* de pacotes de dados DMX será transmitido via OLA(Figura 21). Cada posição do vetor data é um canal DMX, responsável por controlar uma característica do projetor, por exemplo o canal 1, ou a posição 0 do vetor(uma vez que a posição inicial de vetores é a 0), controla as cores, o canal 4 (posição 3 do vetor), é a função estrobo, e assim por diante. Cada projetor pode possuir funções diferentes em cada canal, por isto é de suma importância consultar o manual para poder se realizar a programação adequada. O projetor utilizado neste projeto é o *Triton Blue Spot sp 250 mini*. Por fim a função *wrapper.Run( )* fica responsável por realizar a transmissão dos dados para o projetor através do dispositivo USB previamente configurado no universo DMX criado no OLA.

```
if padrao1==1:
        data[1]=125
        data[2]=38
        data[9]=150
elif padrao1==2:    # o segundo ja e com o dimmer a 100%, cores alternantes e o efeito estrobbe
        data[3]=130
        data[1]=220
        data[0]=255
        data[9]=255
client.SendDmx(universe,data,DmxSent)
wrapper.Run() # envia os padroes para os dispositivos
```

Figura 21: Padrões de dados DMX, onde cada canal é uma característica do projetor(fonte: o autor).

5.12. Fluxograma do código

A estrutura final do código está representada na imagem a seguir (Figura 22), onde os dados gerados pelo acelerômetro são transmitidos via *bluetooth* para o Pymotion tracker que interpreta e concatena estes dados de forma que a Regressão logística consiga realizar a classificação. Esta classificação será utilizada na geração das mensagens DMX para que então o OLA estruture e envie para o projetor.

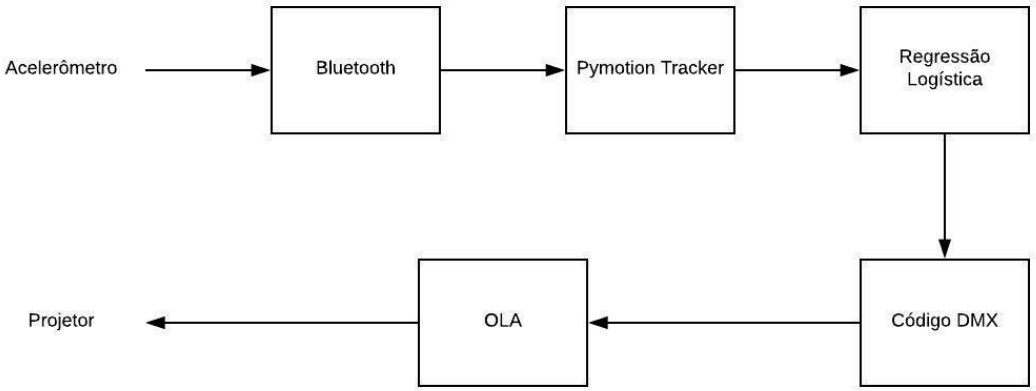

Figura 22: Fluxograma da estrutura do código (fonte: o autor).

6. Análise e Discussão de Resultados

Este capítulo apresenta os resultados e depois discute-os.

6.1. Introdução

Assim que todos os componentes do projeto estavam funcionando corretamente foi dado início ao processo de treinamento efetivo do classificador, para depois poder se realizar o mapeamento. Foi treinado dois movimentos: (i)um balançar leve da mão para representar uma parte calma da música, e (ii) um movimento mais brusco, um soco no ar, que neste caso representa uma parte animada da música. Ambos movimentos foram escolhidos principalmente pela distinção entre eles, facilitando assim sua classificação. O movimento mais calmo, dispara um padrão no projetor com o *dimmer* a 30%, luzes frias, e sem efeitos, já no movimento brusco o projetor ficam alterando suas cores, piscando em efeito estrobo, *dimmer* a 100% com sua luminosidade total, e um movimento de *shake*.

Em seguida o script de previsão foi rodado, obtendo uma classificação não satisfatória, com uma precisão de aproximadamente 30% dos valores corretos.

Um ponto que pode impactar no resultado final é o número de movimentos de cada classe bem como o número de amostras em cada movimento. Foi adotado como ponto de partida 45 movimentos por classe e 500 amostras em cada movimento. Desta forma, em um período de 2 segundos foram registrados 500 dados do acelerômetro, formando assim um movimento, e este procedimento foi repetido 45 vezes para formar uma classe.

Outras configurações foram aplicadas para verificar se a precisão do classificador sofria alguma alteração. Observou-se que aumentar o número de amostras em cada movimento pode diminuir a precisão do classificador. Uma hipótese para isto seria devido à quantidade de amostras redundantes que acabavam por confundir o classificador, bem como aumentava a complexidade dos cálculos de forma cúbica, reduzindo assim sua assertividade.

6.2. Problemas com o SVM

Devido à baixa precisão foi alterado os parâmetros de amostragem, bem como os parâmetros do construtor SVM. Variou-se a tolerância ao erro, o modelo do *kernel* e o *gamma*, que é um parâmetro para velocidade dos cálculos do hiperplano.

Entretanto a precisão dos resultados não parecia convergir, hora o construtor acusava uma precisão de quase 90%, hora apenas 30%.

Não havia uma consistência nos resultados, os mesmos parâmetros, treinados mais de uma vez apresentavam resultados diferentes. Aparentemente o problema era a estrutura dos dados.

Relembrando o projeto bem sucedido de Terzi [34], nota-se que uma característica deste projeto é que os dados são dinâmicos e, partindo desta característica, buscou-se na literatura uma resposta às dificuldades observadas na aplicação do SVM. De acordo com informações a respeito da escolha de algoritmos no Microsoft Azure [38] (uma plataforma desenvolvida pela Microsoft para implementação de *Machine Learning*) o SVM é apresentado como uma solução não propícia à aplicações com dados dinâmicos, sendo sugerida a Regressão Logística uma alternativa à este tipo de conjunto de dados.

6.3. Regressão logística, uma alternativa ao SVM

Observada a ineficiência do SVM na classificação de dados dinâmicos, a Regressão Logística foi implementada. A biblioteca Scikit Learn [36] também oferece um construtor de Regressão Logística, com a mesma estrutura do construtor SVM, não sendo necessário muitas alterações no *software* desenvolvido. Para comparar a eficiência de ambos classificadores, o classificador de Regressão Logística foi implementado junto ao código do SVM, de modo que os mesmos dados de aprendizagem são passados para ambos os construtores.

Conclusão

Por fim a Regressão Logística, pelo fato de ser um algoritmo de natureza probabilística, obteve um resultado mais satisfatório na classificação de dados dinâmicos. Com uma precisão adequada, e uma acurácia de aproximadamente 80%, a Regressão Logística acabou por ser utilizada na versão final do projeto. Com a limitação de ser necessário em média dois movimentos para se realizar a classificação corretamente.

7. Conclusões

Será apresentado aqui as conclusões do trabalho, bem como propostas para melhorias futuras.

7.1. Introdução

Devido à este ser um projeto inovador, não foi possível achar muitas fontes para se basear, o que acabou por acarretar os mesmos problemas de muitas ideias inovadoras. Mesmo com todos os problemas aqui apresentado no desenvolvimento deste protótipo, ainda foi possível obter um resultado satisfatório. Ainda que um pouco diferente do inicialmente planejado, o protótipo mostrou grande potencial para aplicações práticas.

A transformação da animação (movimento) de uma pessoa em padrões luminosos com certeza vai tornar mais imersiva e muito mais intensa a experiencia de qualquer pessoa que frequentar o espetáculo em questão.

7.2. Possíveis melhorias

Dos dois algoritmos utilizados na classificação do movimento, o SVM não obteve resultados satisfatórios, e a Regressão Logistica apresentou resultados esperados, porém com limitações tanto nos movimentos que estavam sendo treinados, bem como no tempo de resposta para a classificação, que levava em torno de 2 movimentos para se obter uma classificação adequada.

Algo que pode vir para sanar este problema é a utilização de Redes Neurais, que possuem um desempenho muito mais satisfatório em problemas dinâmicos, bem como a possibilidade de refinamento dos resultados com cada classificação bem-sucedida, tornando o algoritmo mais preciso de acordo com o passar do tempo, coisa que os classificadores atuais não são capazes. Mas é necessário mudar toda a estrutura dos dados para poder implementar as Redes Neurais, por isso será um aprimoramento futuro.

7.3. Propostas futuras

Apesar de ser apenas um protótipo há uma vasta gama de possíveis melhorias neste projeto. O próximo passo que pode ser executado é a implementação de um algoritmo de Transformada de Fourier para analisar o espectro de frequência da música que está sendo reproduzida. E desta forma, manipular as luzes de acordo com suas componentes espectrais que, aliado ao controle via movimentos do usuário, possibilitaria um ambiente ainda mais imersivo.

Outro aprimoramento é a integração de um magnetômetro para corrigir o *drifiting* causado pelo acelerômetro quando é obtido o descolamento utilizando apenas a aceleração. Esta solução possibilitaria mapear o deslocamento do dispositivo com mais precisão. Ainda nesta mesma linha, implementar um sistema de posicionamento, utilizando 3 transmissores de rádio espalhados pelo clube e realizando assim a triangulação do sinal enviado pelos 3 transmissores, calculando o tempo que cada sinal

leva para viajar do transmissor até o dispositivo. Com este posicionamento seria possível criar zonas especiais e padrões diferenciados nestas zonas.

Este projeto utiliza apenas um projetor o Triton blue spot sp 250 mini. Para que seja possível utilizar outros projetores e até mesmo mais do que um é necessário implementar o sistema RDM. Vale lembrar que o OLA já oferece suporte para tal tecnologia, sendo necessário apenas o hardware para que esta melhoria seja implementada.

7.4. Conclusões

Por fim, o projeto obteve os resultados esperados. Fornecendo ainda uma grande carga de conhecimento em Machine Learning, iluminação de palco, bem como python.

Fornece também uma base para qualquer projeto voltado para iluminação de palco, para que os espetáculos possam ficar cada vez mais artísticos, os eventos mais imersivos. Que assim as pessoas possam aproveitar uma experiencia mais intensa, e levar a lembrança de uma apresentação inesquecível para o resto de suas vidas.

Bibliografia

[1] R. Cadena, Automated Lighting: The Art and Science of Moving Light in Theatre, Live Performance, and Entertainment, Taylor & Francis, 2010.

[2] R. E. Dunham, Stage Lighting: Fundamentals and Applications, CRC Press, 2006.

[3] H. Heimbach , A. Klein, A. Lin, M. Lu e V. Zhuang, "Developing Theatrical Lighting Control with Arduino".

[4] Theatre crafts, [Online]. Available: http://www.theatrecrafts.com/pages/home/archive/equipment/detail/?id=7013. [Acesso em 24 abril 2018].

[5] USITT, "FAQ USITT," USITT, [Online]. Available: https://www.usitt.org/faq/#what-is-the-history-of-dmx512. [Acesso em 27 Abril 2018].

[6] BH Photo Video, [Online]. Available: https://www.bhphotovideo.com/images/images500x500/Hosa_Technology_DMX_306_5_Pin_XLR_Female_to_407855.jpg. [Acesso em 12 maio 2018].

[7] Maxim Integrated, "TUTORIAL 736 RS-485 (EIA/TIA-485) Differential Data Transmission System Basics," Maxim Integrated, [Online]. Available: https://www.maximintegrated.com/en/app-notes/index.mvp/id/736. [Acesso em 16 Maio 2018].

[8] KRON Medidores, "Conceitos Básicos de RS-485 e RS-422".

[9] Linear77. [Online]. Available: https://commons.wikimedia.org/wiki/File:DiffSignaling.png. [Acesso em 16 maio 2018].

[10] P. E. F. Magalhães, "Ethernet Stage Lightning Protocol: An Alternative System for professional Lightning Equipment with Legacy DMX Compatibility.," 2015.

[11] Laser Dj, [Online]. Available: https://www.laserdj.com.br/produtos/dmx-384-laserdj.jpg. [Acesso em 19 maio 2018].

[12] Avolites, [Online]. Available: https://www.avolites.com/Portals/0/CVStoreImages/Sapphire%20Touch%20-%20Front_900.jpg. [Acesso em 19 maio 2018].

[13] The Free On-line Dictionary of Computing, "Foldoc," [Online]. Available: http://foldoc.org/Application+Program+Interface. [Acesso em 2018 Setembro 25].

[14] Raspberry Pi, “Download,” Raspberry Pi, [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/. [Acesso em 05 outubro 2018].

[15] Raspberry Pi, “Foto,” [Online]. Available: https://www.raspberrypi.org/app/themes/mind-control/images/home-products-cta__image.png. [Acesso em 22 maio 2018].

[16] C. WoodFord, “Accelerometers,” *Explain That Stuff,* 2014.

[17] Bang Good, “JY-06 Module,” [Online]. Available: https://img.staticbg.com/thumb/water/oaupload/banggood/images/29/45/3481bf5b-105d-4557-80f0-ddc56a50874b.JPG. [Acesso em 22 maio 2018].

[18] E. Alpaydin, Introduction to machine learning, Massachussets: The MIT Press, 2010.

[19] V. CHERKASSKY e Y. MA, “Practical selection of SVM parameters and noise estimation for SVM regression,” 2004.

[20] H. Kandan, “Towards data science,” [Online]. Available: https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78. [Acesso em 22 maio 2018].

[21] L. WANG, Support Vector Machines: Theory and Applications, Springer, 2005.

[22] MIT Media, [Online]. Available: http://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/. [Acesso em 04 outubro 2018].

[23] M. Pal e M. Mather, “Support vector machines for classification in remote sensing,” 2005.

[24] D. G. Kleinbaum e M. Klein, Logistic Regression: A Self-Learning Text, 2010.

[25] S. S. Mangiafico, An R Companion for the Handbook of Biological Statistics, R companion, 2015.

[26] Etcher, [Online]. Available: https://etcher.io/. [Acesso em 05 outubro 2018].

[27] Open Light Project, “Linux Install,” [Online]. Available: https://www.openlighting.org/ola/linuxinstall/. [Acesso em 05 outubro 2018].

[28] CD MEX, [Online]. Available: https://http2.mlstatic.com/modulo-convertidor-usb-20-a-ttl-rs485-serial-ft232rl-cdmex-D_NQ_NP_610846-MLM25883649278_082017-F.jpg. [Acesso em 05 outubro 2018].

[29] M. Mitterdorfer, “Pymotiontracker,” [Online]. Available: https://github.com/fundiZX48/pymotiontracker. [Acesso em 05 outubro 2018].

[30] Linux Magazine, [Online]. Available: http://www.linux-magazine.com/var/linux_magazin/storage/images/issues/2017/197/command-line-bluetoothctl/figure-3/694900-1-eng-US/Figure-3_large.png . [Acesso em 06 outubro 2018].

[31] K. Woźnica. [Online]. Available: https://commons.wikimedia.org/wiki/File:Battery-lithium-cr2032.jpg. [Acesso em 06 outubro 2018].

[32] Gear Best, [Online]. Available: https://gloimg.gbtcdn.com/gb/pdm-product-pic/Electronic/2016/04/26/goods-img/1511578233835840038.JPG. [Acesso em 06 outubro 2018].

[33] Elecmaster, "JY-61 MPU6050 module User Manual by Elecmaster," [Online]. Available: https://pt.scribd.com/document/276841798/JY-61-MPU6050-module-User-Manual-by-Elecmaster. [Acesso em 05 outubro 2018].

[34] F. Terzi, "Gesture Keyboard," [Online]. Available: https://github.com/federico-terzi/gesture-keyboard . [Acesso em 06 outubro 2018].

[35] F. Terzi. [Online]. Available: http://federicoterzi.com.

[36] Scikit Learn, "Scikit Learn, Machine Learning in python," [Online]. Available: http://scikit-learn.org/. [Acesso em 22 outubro 2018].

[37] Scikit Learn, [Online]. Available: http://scikit-learn.org/stable/tutorial/basic/tutorial.html. [Acesso em 09 outubro 2018].

[38] Microsoft, "How to choose algorithms for Azure Machine Learning Studio," [Online]. Available: https://docs.microsoft.com/pt-pt/azure/machine-learning/studio/algorithm-choice. [Acesso em 22 outubro 2018].

Anexos

Código do script de aprendizagem

```
from sklearn import svm
from sklearn import preprocessing
import numpy as np
import motiontracker as mt
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import sys
import time


##Variaveis

rng=input('digite o valor do range :') ## o range sao quantas amostras eu vou ter

slp=3.0/rng #esse e o tempo de espera entre uma amostra e outra

svm_vector=np.array([[0 for i in range(rng*9)]])   # o vetor que vai receber as amostras, o tamanho dele e proporcional ao numero de amost

model_svm= svm.SVC(                                                                                                                                  )
model_LogR=Logistic

label=np.array([4]) #esse vetor vai guardar o rotulo de cada amostra, comecei com 4 so para inicializar, mas depois eu dispenso as primeir

mpu = mt.MotionTracker(bd_addr="20:17:12:04:51:13") # faz a conexao com o dispostivo bluetooth




try:
        mpu.start_read_data()
```

```
        time.sleep(3.5)
        print(slp)
        while (1):
                modelo=input('Digite o padrao :')  # esse e o rotulo do padrao que eu vou gravar a seguir
                if modelo==99:    # o padrao 99 finaliza o aprendizado
                        break
                else:
                        for j in range(45):    # a principio e treinado 45 vezes cada padrao
                                x=np.array([])
                                Accx=np.array([mpu.acc_x])
                                Accy=np.array([mpu.acc_y])
                                Accz=np.array([mpu.acc_z])
                                Angx=np.array([mpu.ang_x])
                                Angy=np.array([mpu.ang_y])
                                Angz=np.array([mpu.ang_z])
                                Accax=np.array([mpu.angv_x])
                                Accay=np.array([mpu.angv_y])
                                Accaz=np.array([mpu.angv_z])
                                for i in range(rng-1):
                                        Accx=np.append(Accx,[mpu.acc_x])
                                        Accy=np.append(Accy,[mpu.acc_y])
                                        Accz=np.append(Accz,[mpu.acc_z])
                                        Angx=np.append(Angx,[mpu.ang_x])
                                        Angy=np.append(Angy,[mpu.ang_y])
                                        Angz=np.append(Angz,[mpu.ang_z])
                                        Accax=np.append(Accax,[mpu.angv_x])
                                        Accay=np.append(Accay,[mpu.angv_y])
                                        Accaz=np.append(Accaz,[mpu.angv_z])
                                        time.sleep(slp)
                                label=np.append(label,modelo)
```

```
                                print(j)
                                x=np.append(x,[Accx,Accy,Accz,Angx,Angy,Angz,Accax,Accay,Accaz])
                                svm_vector=np.append(svm_vector,[x],0)
        X_train,X_test,y_train,y_test = train_t...                    _size=0.2) # esta funcao reparte meus dados em 80
        model_LogR.fit(X_train,y_train) # Treinamento do Logistic Regression
        model_svm.fit(X_train,y_train)  # Treinamento do SVM
        print(model.score(X_test,y_test),model1.score(X_test,y_test))  # exibo o score do meu modelo ou seja quantas vezes ele acertou
        joblib.dump(model_svm,'model.pkl')    # isso e para salvar o modelo svm em um arquivo para poder abrir no outro programa
        joblib.dump(model_LogR,'model1.pkl')  # e aqui salvamos o modelo da regressao logistica.
        print('JOB`S DONE!')


  except KeyboardInterrupt:
        mpu.stop_read_data()
        raise
```

Código do script de previsão

```

from __future__ import print_function
from sklearn import svm
import numpy as np
import motiontracker as mt
from sklearn.externals import joblib
from ola.ClientWrapper import ClientWrapper
import sys
import time
import array

mpu=mt.MotionTracker(bd_addr="20:17:12:04:51:13")

model_svm = joblib.load('model.pkl') # criacao do svm
model_LogR = joblib.load('model1.pkl') # criacao do  logistic regression


gatilho= True # Apenas para manter no loop
rng=250   # Rng e o sample rate, ou seja quantas amostras eu vou pegar em um segundo
slp=3.0/rng # o tempo entre uma amostra e outra
arry=9*rng  # o tamanho do vetor que recebera as amostras
universe=1  # o universo criado no OLA
data=array.array('B') # O pacote de dados que sera enviado via dmx, o parametro B representa o UNSIGNED


```

```
for i in range(11):
	data.append(0)   # inicializo o vetor data  com 0 nas 11 primeiras posicoes, pois o dispositivo utilizado neste projeto (Triton sp

global wrapper
wrapper = ClientWrapper()
client = wrapper.Client()


def DmxSent(status):
	if status.Succeeded():
		print("success")
	else:
		print('Error %s' %status.message, file=sys.stderr)
	global wrapper
	if wrapper:
		wrapper.Stop()

# o buffer vai receber em tempo real os valores de aceleracao

mpu.start_read_data()
time.sleep(2.5)
print('here we go!')
while gatilho: # loop do buffer, que continuara recebendo os valores e tentanto achar algum padrao correspondende
	buffer=np.array([]) # zera o buffer, para as amostras anteriores nao influenciar na atual
	Accx=np.array([mpu.acc_x])
	Accy=np.array([mpu.acc_y])
	Accz=np.array([mpu.acc_z])
	Acgx=np.array([mpu.ang_x])
	Acgy=np.array([mpu.ang_y])
	Acgz=np.array([mpu.ang_z])
	Acax=np.array([mpu.angv_x])
	for i in range(lng-1):
		Accx=np.append(Accx,[mpu.acc_x])
		Accy=np.append(Accy,[mpu.acc_y])
		Accz=np.append(Accz,[mpu.acc_z])
		Acgx=np.append(Acgx,[mpu.ang_x])
		Acgy=np.append(Acgy,[mpu.ang_y])
		Acgz=np.append(Acgz,[mpu.ang_z])
		Acax=np.append(Acax,[mpu.angv_x])
		Acay=np.append(Acay,[mpu.angv_y])
		Acaz=np.append(Acaz,[mpu.angv_z])
		time.sleep(slp)



	buffer=np.append(buffer,[Accx,Accy,Accz,Acgx,Acgy,Acgz,Acax,Acay,Acaz])
	padrao=model_svm.predict([buffer]) #previsao com o SVM
	padrao1=model_LogR.predict([buffer]) # previsao com o logistic regression, para comparacao de precisao
	print(padrao,padrao1)



	 # A partir do numero previsto e entao ativado um dos dois padroes de cores, o primeiro e com o dimmer a 30% cores mais calmas e sem
	 if padrao1==1:
		data[1]=125
		data[2]=38
		data[9]=150
	 elif padrao1==2:    # o segundo ja e com o dimmer a 100%, cores alternantes e o efeito estrobbe
		data[3]=130
		data[1]=220
		data[0]=255
		data[9]=255
	 client.SendDmx(universe,data,DmxSent)
	 wrapper.Run() # envia os padroes para os dispositivos
	 time.sleep(0.5)
```